IEEE Robotics and Automation Letters (RA-L) paper, presented at ICRA 2026, Vienna, Austria. Cite as RA-L paper.

What Matters in Learning a Zero-Shot Sim-to-Real RL Policy for Quadrotor Control? A Comprehensive Study

Jiayu Chen, Chao Yu, Yuqing Xie, Feng Gao, Yinuo Chen, Shu'ang Yu, Wenhao Tang, Shilong Ji, Mo Mu, Yi Wu, *Member, IEEE*, Huazhong Yang, *Fellow, IEEE*, and Yu Wang, *Fellow, IEEE*

***Abstract*—Precise and agile flight maneuvers are essential for quadrotor applications, yet traditional control methods are limited by their reliance on flat trajectories or computationally intensive optimization. Reinforcement learning (RL)-based policies offer a promising alternative by directly mapping observations to actions, reducing dependency on system knowledge and actuation constraints. However, the sim-to-real gap remains a significant challenge, often causing instability in real-world deployments. In this work, we identify five key factors for learning robust RL-based control policies capable of zero-shot real-world deployment: (1) integrating velocity and rotation matrix into actor inputs, (2) incorporating time vector into critic inputs, (3) regularizing action differences for smoothness, (4) applying system identification with selective randomization, and (5) using large batch sizes during training. Based on these insights, we develop *SimpleFlight*, a PPO-based framework that integrates these techniques. Extensive experiments on the Crazyflie quadrotor demonstrate that SimpleFlight reduces trajectory tracking error by over 50% compared to state-of-the-art RL baselines. It excels in both smooth polynomial and challenging infeasible zigzag trajectories, particularly on small thrust-to-weight quadrotors, where baseline methods often fail. To enhance reproducibility and further research, we integrate SimpleFlight into the GPU-based Omnidrones simulator and provide open-source code and model checkpoints.**

***Index Terms*—Reinforcement learning (RL), machine learning for robot control, aerial systems: applications.**

Received 17 December 2024; accepted 20 May 2025. Date of publication 29 May 2025; date of current version 5 June 2025. This article was recommended for publication by Associate Editor G. Pizzuto and Editor A. Faust upon evaluation of the reviewers' comments. This work was supported in part by the National Natural Science Foundation of China under Grant 62406159 and Grant 62325405, in part by the Postdoctoral Fellowship Program of CPSF under Grant GZC20240830 and Grant 2024M761676, and in part by China Postdoctoral Science Special Foundation under Grant 2024T170496. *(Jiayu Chen and Chao Yu contributed equally to this work.) (Corresponding authors: Chao Yu; Yu Wang.)*

Jiayu Chen, Yuqing Xie, Feng Gao, Yinuo Chen, Shilong Ji, Mo Mu, Yi Wu, Huazhong Yang, and Yu Wang are with Tsinghua University, Beijing 100084, China (e-mail: yu-wang@tsinghua.edu.cn).

Chao Yu is with Tsinghua University, Beijing 100084, China, and also with Beijing Zhongguancun Academy, Beijing 100094, China (e-mail: zoeyuchao@gmail.com).

Shu'ang Yu is with Tsinghua University, Beijing 100084, China, and also with Shanghai Artificial Intelligence Laboratory, Shanghai 200030, China.

Wenhao Tang is with Tsinghua Shenzhen International Graduate School, Shenzhen 518055, China.

For more details, visit our project website at https://sites.google.com/view/simpleflight/.

This article has supplementary downloadable material available at https://doi.org/10.1109/LRA.2025.3575011, provided by the authors.

Digital Object Identifier 10.1109/LRA.2025.3575011

I. Introduction

PRECISE and agile flight maneuvers are essential for UAVs, especially quadrotors, in applications such as package delivery [1], search and rescue [2], and infrastructure inspection [3]. Traditional control methods, whether model-based or model-free, often face limitations due to their dependence on flat trajectories adhering to actuation constraints [4], [5] or the need for accurate system modeling and nonconvex optimization solvers [6], [7], which can restrict policy flexibility. Recently, reinforcement learning (RL) has gained traction as a versatile and efficient alternative for quadrotor control [8], [9]. RL-based policies map observations directly to actions, bypassing the need for actuation constraints or precise system dynamics knowledge [10], enabling lower control latency and the potential for enhanced performance in quadrotor tasks.

A major challenge in RL-based quadrotor control is the sim-to-real gap, where policies trained in simulation often fail to perform stably in real-world deployment without additional fine-tuning. Despite numerous RL-based approaches, there is no unified understanding of the key factors for training robust, zero-shot deployable policies [11], [12], [13], [14], [15], [16]. For example, while reward functions are often designed to constrain control commands and enhance smoothness, the specific reward components critical for ensuring valid commands and task success remain unclear. domain randomization is widely employed to bridge the sim-to-real gap, but the extent to which it is necessary for effective quadrotor control remains unclear. Additionally, other factors influencing the training of robust RL-based policies remain underexplored.

This study explores crucial factors for developing robust RL-based control policies for real-world zero-shot deployment. We identify five key elements across input space design, reward design, system identification and training techniques: (1) integrating velocity and rotation matrix into actor's input, (2) adding time vector to critic's input, (3) employing action difference regularization as smoothness reward, (4) applying system identification to key dynamics parameters with selective randomization, and (5) utilizing large batch sizes. The first two factors optimize input space design for simulation learning, while the remaining three work together to bridge the sim-to-real gap. We implement these techniques in a PPO-based framework, *SimpleFlight*.

Our experimental validation on the Crazyflie 2.1 nano quadrotor demonstrates SimpleFlight's superior performance. The framework achieves over 50% reduction in trajectory tracking error compared to SOTA RL baselines without specialized algorithmic or architectural modifications. SimpleFlight

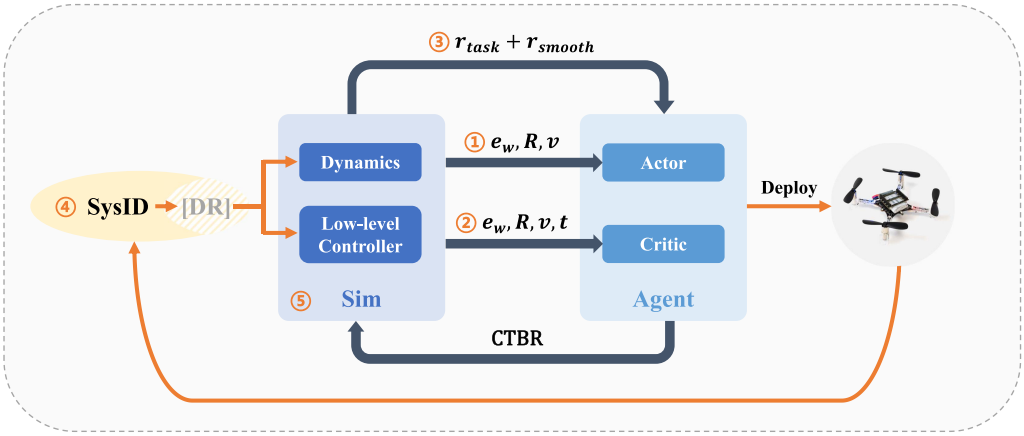


Fig. 1. Overview of SimpleFlight. We begin with SysID and selective DR for quadrotor dynamics and low-level control. Next, an RL policy is trained in simulation to output CTBR for tracking arbitrary trajectories and zero-shot deployed directly on a real quadrotor. The training framework focuses on three key aspects, i.e., input space design, reward design, system identification and domain randomization, as well as training techniques, identifying five critical factors to enhance zero-shot deployment.

successfully completes all benchmark trajectories, including challenging infeasible trajectories, outperforming other RL policies that fail under extreme conditions. Moreover, SimpleFlight demonstrates consistent performance across different quadrotor platforms, including the Crazyflie and our custom-built quadrotor, showcasing its strong generalization capability across varying models and sizes.

Furthermore, we integrate SimpleFlight into a high-parallel GPU-based simulator Omnidrones [17], and we open-source the code, model checkpoints, and benchmark tasks to ensure reproducibility. We believe that SimpleFlight will provide valuable insights to guide future research in RL-based quadrotor control. Our contributions can be summarized as follows:

- We investigate five key learning factors and develop a PPO-based training framework, SimpleFlight, for learning RL-based zero-shot sim-to-real policies.
- We conduct extensive real-world experiments on the Crazyflie to demonstrate the effectiveness of SimpleFlight. The policy derived by SimpleFlight is the only one capable of successfully completing all benchmarking trajectories, including both smooth and infeasible trajectories.
- SimpleFlight reduces trajectory tracking error by over 50% compared to SOTA RL baselines, despite not employing any tailored algorithmic or network architecture design.
- We integrate SimpleFlight into the high-parallel GPU-based simulator Omnidrones and open-source checkpoints to ensure reproducibility.

II. Related Work

A. General Approaches for Sim-to-Real Transfer in Robotics

Bridging the sim-to-real gap is a critical challenge in robotics, with system identification (SysID) being one of the most straightforward approaches [18]. While simulators cannot fully replicate real-world complexities, constructing accurate mathematical models within the simulator can significantly improve the real-world performance of reinforcement learning (RL) policies. However, the dynamic and time-varying nature of the real world, combined with factors such as friction and motor delays, makes strict calibration of simulator parameters through SysID challenging. To address these limitations, domain randomization (DR) has emerged as a promising technique [19], [20]. DR randomizes simulator parameters to cover a broader range of real-world conditions, thereby enhancing policy robustness during deployment.

DR techniques can be categorized into two types: dynamics randomization and visual randomization. Dynamics randomization, particularly effective for control tasks, improves policy stability by randomizing physical parameters such as object size, friction, mass, and damping coefficients [21], [22]. For example, in OpenAI's work on solving a Rubik's Cube using a robotic hand, randomization of these parameters enabled the policy to adapt to diverse real-world conditions. Visual randomization, on the other hand, is primarily used in vision-based tasks to address discrepancies in textures, lighting, and camera perspectives [19], [23].

In addition to DR, domain adaptation techniques have been widely adopted to minimize the distributional discrepancy between simulation (source domain) and reality (target domain) [24], [25], [26], [27], [28]. These techniques are particularly effective for vision-based robotics tasks. In this work, we focus on evaluating the impact of SysID and dynamics randomization on sim-to-real performance, leaving visual randomization and domain adaptation for future exploration.

B. Sim-to-Real Approaches for Quadrotors

In the field of quadrotors, significant advancements have been made in SysID and DR to bridge the sim2real gap. For SysID, Gronauer et al. [29] employ Bayesian optimization to automatically calibrate the simulator's dynamic parameters using real-world flight data. Bauersfeld et al. [30], [31] utilize residual models and real flight data to accurately model the aerodynamics of drones during high-speed flight, further enhancing the fidelity of simulation environments.

For DR, Molchanov et al. [22] randomize key dynamic parameters of the drone, enabling successful policy transfer from simulation to various types of quadrotors. Zhang et al. [32] combine adaptive control with dynamic parameter randomization, transferring policy to real-world environments with unknown disturbances. Additionally, some studies focus on improving RL policy performance in real-world scenarios by designing specialized observation spaces [16], action spaces [15], and incorporating supplementary training methods [33].

IEEE Robotics and Automation Letters (RA-L) paper, presented at ICRA 2026, Vienna, Austria. Cite as RA-L paper.

Despite these advancements, several critical challenges remain unresolved. First, there is no unified framework or standardized pipeline for training robust, zero-shot deployable RL policies. Existing approaches often rely on ad hoc designs for observation spaces, reward functions, and training strategies, making it difficult to generalize findings across different platforms and tasks. Second, while SysID and DR are widely used, their individual and combined effects on sim2real performance are not well understood. For instance, it remains unclear which dynamic parameters are most sensitive to SysID or DR, and under what conditions DR may hinder rather than improve performance. Third, many studies focus on specific aspects of the sim2real pipeline (e.g., observation design or parameter calibration) without considering their interplay, leading to suboptimal solutions. Finally, there is a lack of systematic evaluation of training strategies, such as batch size optimization, which can significantly impact sim2real performance without requiring additional modifications.

III. Preliminary

A. Problem Formulation

We formulate the quadrotor control problem as a Markov Decision Process (MDP). The MDP is defined as $M =< \mathcal{S}, \mathcal{A}, \mathcal{O}, P, R, \gamma >$, with the state space $\mathcal{S}$, the action space $\mathcal{A}$, the observation space $\mathcal{O}$, the transition probability $\mathcal{P}$, the reward function $\mathcal{R}$ and the discount factor γ. Denote the state and the observation at time step t as $(s_t, o_t) \in (\mathcal{S}, \mathcal{O})$. The goal of our work is to construct a policy π_θ parameterized by θ to output action $a_t \sim \pi_\theta(o_t)$ that performs precise and agile maneuvers to *track arbitrary trajectories*. The optimization objective is to maximize the expected accumulative reward $J(\theta) = \mathbb{E}[\sum_t \gamma^t R(s_t, a_t)]$.

B. Quadrotor Dynamics

The quadrotor is assumed to be a 6 $^\circ$-of-freedom rigid body of mass m and diagonal moment of inertia matrix $\boldsymbol{I} = diag(I_x, I_y, I_z)$. The state space is 17-dimensinal and the dynamics are modeled by the differential equation:

$$\dot{\boldsymbol{x}} = \begin{bmatrix} \dot{\boldsymbol{p}}_{\mathcal{W}} \\ \dot{\boldsymbol{q}} \\ \dot{\boldsymbol{v}}_{\mathcal{W}} \\ \dot{\boldsymbol{\omega}}_{\mathcal{B}} \\ \dot{\boldsymbol{\Omega}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{v}_{\mathcal{W}} \\ \boldsymbol{q} \otimes [0, \boldsymbol{\omega}_{\mathcal{B}}/2]^T \\ \frac{1}{m} \boldsymbol{q} \cdot \boldsymbol{f}_{prop} \cdot \overline{\boldsymbol{q}} + \boldsymbol{g}_{\mathcal{W}} \\ \boldsymbol{I}^{-1}(\boldsymbol{\tau}_{prop} - \boldsymbol{\omega}_{\mathcal{B}} \times (\boldsymbol{I}\boldsymbol{\omega}_{\mathcal{B}})) \\ T_m(\boldsymbol{\Omega}_{cmd} - \boldsymbol{\Omega}) \end{bmatrix}, \tag{1}$$

where the quadrotor state $\boldsymbol{x}$ consists the position $\boldsymbol{p}$, the orientation $\boldsymbol{q}$ in quaternions, the linear velocity $\boldsymbol{v}$, the angular velocity $\boldsymbol{\omega}$ and the rotational speed of the rotor $\boldsymbol{\Omega}$. m denotes mass. The subscripts $\mathcal{W}$ and $\mathcal{B}$ represent the world and body frame. The frame $\mathcal{B}$ is located at the center of the mass of the quadrotor. The notation $\otimes$ indicates the multiplication of two quaternions. $\overline{\boldsymbol{q}}$ is the quaternion's conjugate. $\boldsymbol{g}_{\mathcal{W}} = [0, 0, -9.81 m/s^2]^T$ denotes earth's gravity. $\boldsymbol{f}_{prop}$ and $\boldsymbol{\tau}_{prop}$ are the collective force and the torque produced by the propellers. The quantities are defined as follows,

$$\boldsymbol{f}_{prop} = \Sigma_i \boldsymbol{f}_j, \boldsymbol{\tau}_{prop} = \Sigma_j \boldsymbol{\tau}_j + \boldsymbol{r}_{p,j} \times \boldsymbol{f}_j, \tag{2}$$

where $\boldsymbol{r}_{p,j}$ is the location of propeller j expressed in the body frame, $\boldsymbol{f}_j, \boldsymbol{\tau}_j$ are the forces and torques generated by the j-th propeller. The rotational speeds of the four motors Ω_j are modeled as a first-order system with a time constant T_m, where the commanded rotational speeds $\boldsymbol{\Omega}_{cmd}$ serve as the input. For the forces and torques generated by each motor, we adopt a widely used model from prior work [34], [35]:

$$\boldsymbol{f}_j = \left[0, 0, k_f \Omega_j^2\right]^T, \boldsymbol{\tau}_j = \left[0, 0, k_m \Omega_j^2\right]^T. \tag{3}$$

The thrust and torque are modeled as proportional to the square of the motor's rotational speed. The corresponding thrust and drag coefficients, k_f and k_m, as well as the motor time constant, T_m, can be determined using a static propeller test stand.

IV. SimpleFlight

A. Overview

In this section, we describe the details of the entire training framework SimpleFlight, as shown in Fig. 1. The core idea of learning a zero-shot sim-to-real RL policy involves two main aspects: improving policy performance in simulation and bridging the sim-to-real gap to minimize performance drop when deploying the policy in the real world.

To this end, we follow standard practices by first performing SysID to bridge the sim-to-real gap. We calibrate the quadrotor's dynamics by estimating four key parameters: mass m, inertia matrix $\boldsymbol{I}$, thrust coefficient k_f, and motor time constant T_m. These calibrated parameters are then used to model the quadrotor in the simulation. Following previous work, we adopt CTBR as the policy action space, a mid-level control command that has been shown to be more robust to sim-to-real gap [15]. To convert CTBR commands into four motor thrusts, a low-level controller is employed. Furthermore, we align the low-level controller in the quadrotor firmware with the one used in the simulator by calibrating its system response, thereby further bridging the sim-to-real gap.

Next, we train an RL policy in a simulator to enable the quadrotor to track arbitrary trajectories, which is then directly deployed on a real quadrotor without fine-tuning. The training framework emphasizes four critical aspects to enhance zero-shot deployment performance: input space design, reward design, system identification and domain randomization, as well as training techniques. Specifically, input space design aims to improve policy performance in simulation, while the other techniques are tailored to reduce the sim-to-real gap. From these three aspects, we identify and summarize five critical factors. The final design of SimpleFlight is detailed in the following sections, while comparisons with alternative configurations are presented in the experiment section.

B. Input Space Design

We employ a custom asymmetric actor-critic architecture in SimpleFlight. The actor takes as input the relative positions of the quadrotor to the next N reference trajectory points in the world frame $\boldsymbol{e}_{\mathcal{W}}$, the linear velocity $\boldsymbol{v}$, and the rotation matrix $\boldsymbol{R}$. This design allows the policy to perform long-horizon planning, which is particularly crucial for tracking infeasible trajectories with sharp turns. In practice, we set $N = 10$, with each point spaced by 0.05 s. The critic, on the other hand, receives the same inputs as the actor, augmented with a time vector $\mathbf{f}_t = [t, \ldots, t]^T \in \mathbb{R}^k$ as privileged information, where t denotes the discrete timestep in RL training. Since the training performance is not sensitive to the choice of k, we simply set $k = 1$ in our task. While for more complex tasks with high-dimensional

IEEE Robotics and Automation Letters (RA-L) paper, presented at ICRA 2026, Vienna, Austria. Cite as RA-L paper.

observations, we recommend to increase the dimension k if the value estimation proves insufficiently accurate. The time vector enables the critic to capture temporal information, improving its ability to estimate state values effectively. Both the actor and critic use three-layer MLPs, with an ELU activation function and a LayerNorm layer inserted between the two MLP layers to encode their inputs into a latent vector, respectively. The output dimension of the MLP layer is 256. For the actor, this vector parameterizes a Gaussian distribution to generate CTBR actions. For the critic, the vector is further fed into an MLP to produce the estimated state value.

Factor 1: Utilizing the rotation matrix instead of a quaternion and incorporating linear velocity into the actor's input.

Factor 2: Adding a time vector to the critic's input to enhance its temporal awareness.

C. Reward Design

In simulation, RL policies often explore aggressive actions to optimize task performance. For instance, a policy might produce a collective thrust command that abruptly changes from maximum thrust at one timestep to 0 at the next. While such actions may execute without immediate instability in simulation, due to the sim-to-real gap, they become physically infeasible for real quadrotors and can lead to unstable behavior during deployment.

To address this, RL leverages reward design to regularize policy outputs. Existing studies incorporate auxiliary reward components to encourage smooth actions. In general, the reward function is defined as:

$$r = r_{task} + \lambda r_{smooth}, \tag{4}$$

where r_{task} represents the task-specific reward, and r_{smooth} is a smoothness reward designed to promote smooth actions. λ is the coefficient of the smoothness reward. We normalize both r_{task} and r_{smooth} to the range $[0, 1]$, allowing λ to represent the relative weight of the smoothness reward compared to the task-specific reward.

It is important to note the trade-off between r_{task} and r_{smooth}. While r_{smooth} discourages aggressive actions and enforces smoother commands, it can also restrict the quadrotor's ability to exploit agile maneuvers essential for tackling complex and challenging tasks. In SimpleFlight, we adopt the form $||\boldsymbol{u}_t - \boldsymbol{u}_{t-1}||_2$ as the smoothness reward, where $\boldsymbol{u}_t$ represents the policy's action, i.e., CTBR. We hypothesize that this design penalizes abrupt changes in policy output, serving as a soft yet direct constraint. This approach achieves a better trade-off between task completeness and action smoothness, enabling both stable and agile flight.

Factor 3: Incorporating regularization of the difference between successive actions as the smoothness reward.

D. System Identification and Domain Randomization

Our study explores the effects of system identification (SysID), domain randomization (DR), and their combination on sim2real performance with four key dynamic parameters, i.e., mass m, inertia $\boldsymbol{I}$, motor time constant T_m, and thrust coefficient k_f. We find that SysID is crucial for accurate sim2real transfer, particularly for measurable parameters like mass m and inertia $\boldsymbol{I}$, where DR is counterproductive, increasing training complexity and leading to suboptimal policies. For the motor time constant T_m, sim2real performance shows low sensitivity, and DR provides no significant benefits, only introducing unnecessary learning challenges. In contrast, the thrust coefficient k_f is highly sensitive, with deviations causing notable performance drops. However, DR significantly improves robustness.

Factor 4: Applying SysID for calibrating key dynamic parameters is crucial. DR exhibits selective effectiveness, improving performance for sensitive parameters like thrust coefficients while proving detrimental for insensitive or accurately measurable parameters.

E. Training Techniques

We highlight a often-overlooked training technique that significantly impact policy performance. Increasing the batch size during training without requiring additional modifications improves real-world performance despite having limited impact on simulation results. This benefit possibly arises from the enhanced data diversity generated by larger batch sizes, which strengthens the policy's generalization capacity.

Factor 5: Leveraging larger batch sizes during training.

V. Experiment

A. Experiment Setup

1) Benchmark Trajectories: We adopt two types of trajectories as benchmark trajectories: **smooth trajectories** (figure-eight and polynomial) and **infeasible trajectories** (pentagram and zigzag). The figure-eight and pentagram trajectories are deterministic, while the polynomial and zigzag trajectories are randomly generated for each trial. All trajectories start from the origin $(0, 0)$ with a fixed z-axis height. Examples of benchmark trajectories are shown in Fig. 2.

a. Figure-eight The figure-eight trajectory is a periodic smooth curve defined as $\boldsymbol{p}(t) = [\cos(2\pi t/T), \sin(4\pi t/T)/2, 1]$, where T represents the time required to complete one figure-eight lap. We test three velocities by varying T: 15.0 s (Slow), 5.5 s (Normal), and 3.5 s (Fast), corresponding to maximum velocities of 0.6 m/s, 1.6 m/s, and 2.5 m/s in the reference trajectories, respectively.

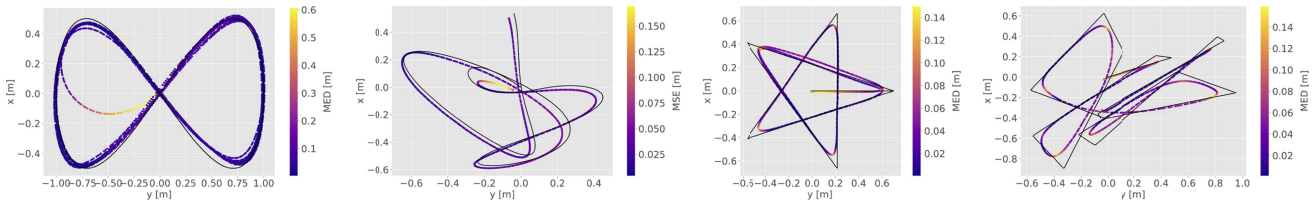
b. Polynomial The smooth polynomial trajectory consists of multiple randomly generated 5-degree polynomial segments. The duration of each segment is randomly selected between 1.5 s and 4 s. The velocity of the reference trajectories ranges from 0 to 1 m/s. To ensure smoothness, we enforce continuity of the first, second, and third derivatives at the junctions between consecutive segments.

c. Pentagram The pentagram is an infeasible trajectory where the quadrotor sequentially visits the five vertices of a pentagram at a constant velocity. We test two different velocities: 0.5 m/s and 1 m/s, marked as Slow and Fast, respectively.

d. Zigzag The zigzag trajectory is infeasible and is generated based on several randomly selected waypoints, with $x-$ and $y-$ coordinates distributed between -1 m and 1m. Consecutive waypoints are connected by straight lines, with time intervals randomly chosen between 1s and 1.5 s. The velocity of the reference trajectories ranges from 0 to 2m/s.

We note that pentagram and zigzag trajectories are challenging infeasible due to their sharp directional changes, i.e., infinite acceleration. Successfully tracking these trajectories requires quadrotors to perform long-horizon optimization and operate near the limits of their system performance, which is difficult for quadrotors with low thrust-to-weight ratios.

IEEE Robotics and Automation Letters (RA-L) paper, presented at ICRA 2026, Vienna, Austria. Cite as RA-L paper.



(a) Figure-eight (fast). (b) Polynomial. (c) Pentagram (fast). (d) Zigzag.

Fig. 2. Visualization of benchmark trajectories and corresponding trajectories followed using SimpleFlight. The reference trajectories are shown in black.

2) Training Details

We employ OmniDrones [17], a GPU-accelerated, highly parallel drone simulator, to train the quadrotor control policy using the on-policy PPO algorithm [36]. The simulator operates at 100 Hz with a timestep of 0.01 s. During training, we use a balanced mixture of smooth randomized polynomial trajectories and infeasible zigzag trajectories, generated under consistent benchmark rules. This setup makes figure-eight and pentagram trajectories out-of-distribution (OOD) test cases. The policy is trained for 15,000 epochs, with a single policy derived for all trajectories.

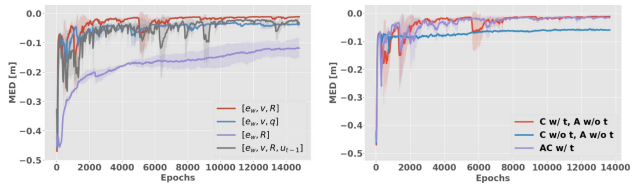
3) Evaluation Metric

We assess tracking performance using the Mean Euclidean Distance (MED) between the quadrotor's actual and target positions in the x- and y-axes, averaged over the entire trajectory. For the figure-eight trajectory, MED is averaged over ten repetitions per trial across three trials. For the pentagram trajectory, MED is computed over three single-run trials. For polynomial and zigzag trajectories, we generate five random trajectories of each type, repeating each twice, resulting in ten trials. MED values are averaged and reported as "mean (standard deviation)" in meters (m).

B. In-Depth Analysis on Key Factors

We systematically evaluate all proposed key factors through simulation and real-world experiments on figure-eight trajectories. For deployment, we use the Crazyflie 2.1 nano-quadrotor, with position, velocity, and orientation data provided by an OptiTrack motion capture system at 100 Hz. An offboard computer executes the policy, sending CTBR commands to the quadrotor via a 2.4 GHz radio at 100 Hz. Notably, we deploy a single randomly-selected policy (from three training seeds) due to the demonstrated robustness of our training process.

1) Factors 1&2: Input Space Design: Fig. 3 illustrates the training curves for different input space designs. For the common inputs of the actor and critic (Fig. 3(a)), $\boldsymbol{q}$ denotes the quaternion corresponding to the rotation matrix $\boldsymbol{R}$. Leveraging the relative positions $\boldsymbol{e}_{\mathcal{W}}$, linear velocity $\boldsymbol{v}$, and the rotation matrix $\boldsymbol{R}$ achieves the best tracking performance in simulation. We observe a training performance degradation (approximately 63.6%) when replacing $\boldsymbol{R}$ with $\boldsymbol{q}$. This is because representations for the 3D rotations are discontinuous in four or fewer dimensions, making it challenging for neural networks to learn, as also observed in graphics and vision studies [37]. Excluding the linear velocity $\boldsymbol{v}$ significantly degrades performance, underscoring its importance for predicting actions in agile quadrotor control. Including the previous action $\boldsymbol{u}_{t-1}$, as suggested in prior works [16], [31], results in a slight performance drop. We hypothesize this is because the RL policy continuously evolves during training, and including the previous action in the input introduces non-stationary into the environment, thereby reducing performance.



(a) Common inputs. (b) Impact of the time vector.

Methods	Figure-eight		
	Slow	Normal	Fast
AC w/ t	0.003(0.000)	0.010(0.002)	0.033(0.004)
C w/ t, A w/o t	**0.002(0.001)**	**0.006(0.001)**	**0.024(0.004)**

(c) The tracking performance on figure-eight with 10 laps.

Fig. 3. Training performance of input space designs. (a) For shared actor-critic inputs, the combination of relative positions $\boldsymbol{e}_{\mathcal{W}}$, linear velocity $\boldsymbol{v}$, and rotation matrix $\boldsymbol{R}$ achieves the lowest simulated MED. (b) Adding the time vector to inputs significantly improves tracking performance. (c) Including the time vector in actor inputs slightly degrades performance for long-duration trajectories.

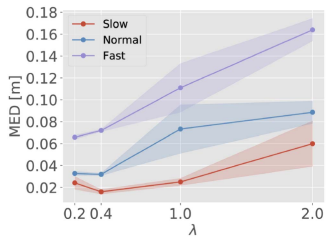


Fig. 4. Real-world performance of different λ on the figure-eight trajectory. We finally choose $\lambda = 0.4$.

Regarding the impact of the time vector (Fig. 3(b)), we evaluate three configurations: (1) time vector in both actor and critic (*AC w/ t*), (2) time vector only in critic (*C w/ t, A w/o t*), and (3) no time vector (*AC w/o t*). Results show that incorporating the time vector significantly improves tracking accuracy (*AC w/ t* and *C w/t, A w/o t*), as it enhances the critic's ability to capture temporal information and estimate state values. However, including the time vector in the actor (*AC w/ t*) can cause out-of-distribution (OOD) issues during long-duration flights, as the reference trajectory's timesteps may exceed the training trajectory's maximum length (Table III(c)). Thus, we include the time vector only in the critic to balance accurate value estimation with robust performance.

IEEE Robotics and Automation Letters (RA-L) paper, presented at ICRA 2026, Vienna, Austria. Cite as RA-L paper.

TABLE I
REAL-WORLD TRACKING PERFORMANCE OF DIFFERENT SMOOTHNESS COMPONENTS THAT ENCOURAGE VALID AND SMOOTH CONTROL COMMANDS

	Slow	Normal	Fast
Action Clipping	0.035(0.001)	0.077(0.016)	0.310(0.016)
Low-Pass Filter	∞	∞	∞
$\|\|\boldsymbol{acc}_t\|\|_2$	0.183(0.004)	0.329(0.004)	∞
$\|\|\boldsymbol{jerk}_t\|\|_2$	0.024(0.009)	0.047(0.005)	∞
$\|\|\boldsymbol{snap}_t\|\|_2$	0.026(0.002)	∞	∞
$\|\|\boldsymbol{u}_t\|\|_2$	0.044(0.003)	0.066(0.002)	0.110(0.027)
$\|\|\boldsymbol{u}_t - \boldsymbol{u}_{t-1}\|\|_2$	**0.016(0.002)**	**0.028(0.000)**	**0.051(0.002)**

The action-level regularization $\|u_t - u_{t-1}\|_2$ achieves the best tracking performance among all designs.

2) Factor 3: Smoothness Reward Design: We evaluate various smoothness components commonly used in existing studies, with the real-world tracking performance summarized in Table I. Here, $\boldsymbol{acc}_t, \boldsymbol{jerk}_t, \boldsymbol{snap}_t$ represent the second, third, and fourth derivative of position at timestep t, respectively, and $\boldsymbol{u}_t$ denotes the policy's CTBR output at timestep t. Note that $||\boldsymbol{u}_t||_2$ penalizes desired angular velocity and thrust, indirectly constraining the third derivative of position, while $||\boldsymbol{u}_t - \boldsymbol{u}_{t-1}||_2$ penalizes angular acceleration and differential thrust, indirectly targeting the fourth derivative. The smoothness reward is defined as $r_{aux} = e^{-A}$, where A represents different forms of smoothness reward components. A grid search is conducted to optimize the hyperparameters for each component, and the best results are reported. In addition to reward design, we examine two alternative methods to encourage valid and smooth commands: "Action Clipping (AC)" and "Low-Pass Filter (LPF)". AC directly limits actions exceeding a predefined threshold, while LPF smooths the policy output as $\boldsymbol{u}'_t = \alpha \boldsymbol{u}_t + (1-\alpha)\boldsymbol{u}_{t-1}$, where α is the cutoff parameter. Among these, $||\boldsymbol{u}_t - \boldsymbol{u}_{t-1}||_2$ achieves the best real-world tracking performance. In contrast, non-reward methods like LPF fail to generalize across velocities, and AC limits agile maneuvers, resulting in suboptimal performance for high-velocity trajectories. Direct action constraints such as $||\boldsymbol{u}_t||_2$ outperform indirect kinematic constraints like $||\boldsymbol{jerk}_t||_2$ on fast trajectories, highlighting the effectiveness of action-level regularization for challenging tasks.

Since there exists a trade-off between the task reward and the smoothness reward, we conduct experiments on the smoothness reward coefficient λ, with the real-world tracking performance shown in Fig. 4. For $\lambda = 0.2$, not all fast trajectory experiments are successful; only results from successful trials are reported. As observed, larger λ can degrade tracking performance due to restricted agility, while smaller λ may lead to unstable flight. Based on these findings, we set $\lambda = 0.4$ for a trade-off.

3) Factor 4: SysID and DR: We analyze the effects of SysID and DR on tracking performance for figure-eight trajectories at normal velocity in Table II. We begin by calibrating four key dynamic parameters—mass m, inertia $\boldsymbol{I}$, motor time constant T_m, and thrust coefficient k_f—and report results using these calibrated values as "SysID". We then apply DR to each parameter, exploring two parameter randomization ranges: $[-10\%, +10\%]$ and $[-30\%, +30\%]$, denoted as "SysID+DR10%" and "SysID+DR30%," respectively. Our findings indicate that applying DR to approximately well-calibrated parameters generally does not improve sim-to-real transfer performance as it may increase learning complexity. In addition, introducing DR leads to slightly higher standard deviations, indicating that it can induce more unstable behavior. To further simulate inaccurate calibration, we introduce a $+30\%$ offset to each dynamics parameter, referred to as "Offset+30%." The results reveal varying performance sensitivity across parameters. Precise measurement is critical for sensitive parameters such as mass m and thrust coefficient k_f, while strict calibration is less essential for less sensitive parameters like inertia $\boldsymbol{I}$ and motor time constant T_m. We then apply DR to these offset values (termed "Offset+DR30%") and observe a noticeable improvement in the shifted thrust coefficient's performance, alongside comparable results for the other parameters. Therefore, for sensitive parameters, such as the thrust coefficient k_f, when accurate calibration is not feasible, DR can effectively enhance the robustness of the policy. For non-sensitive parameters, DR primarily increases the learning difficulty without providing performance benefits.

4) Factor 5: Effect of Batch Sizes: To evaluate the impact of the batch sizes, we test simulation and real-world performance using figure-eight trajectories (slow, normal and fast) via varying parallel environments. As shown in Fig. 5, increasing the batch size enhances real-world performance as simulation performance converges, with real-world results also stabilizing as the batch size grows further. Based on this finding, we recommend using larger batch sizes during training to enhance sim-to-real transfer.

C. Comparison of Other Methods

1) Baselines: While our paper focuses on robust RL policies for zero-shot real-world deployment (i.e. relative performance), we also show SimpleFlight's strong absolute performance. We benchmark two SOTA RL methods (DATT [11] and Fly [33]) on the Crazyflie and stress-test SimpleFlight against a well-tuned MPC method, PAMPC [38] on our custom quadrotor, named Air, which features a 250 mm arm length and is equipped with a PX4 flight controller and an Nvidia Orin processor.

a. DATT [11] is a feedforward-feedback-adaptive policy for CTBR command-based trajectory tracking, achieving SOTA performance over PID and non-linear MPC [39]. We retrain DATT on the standard Crazyflie 2.1 (body rate: $[-\pi, \pi]$rad/s, acceleration: $[0, 1.6\,\mathrm{g}]$) to ensure fair comparison, retaining its disturbance estimation for optimal performance. Deployment follows the same protocol as SimpleFlight.

b. Fly [33] proposes a high-speed simulator and RL-based framework for direct RPM control, enabling superior sim-to-real transfer. Using the released checkpoint, we perform onboard inference to evaluate its performance on benchmark trajectories, as it outperforms PID and prior RL policies.

c. PAMPC [38] is a non-linear MPC method that jointly optimizes perception and action objectives. By excluding the vision objective and fine-tuning cost function hyperparameters, we adapt it to ensure accurate tracking performance across all benchmark trajectories.

2) Real-World Experiments: We first report the trajectory tracking performance of SimpleFlight compared to the baseline methods across all benchmark trajectories on the Crazyflie, as shown in Table III. SimpleFlight significantly outperforms all baselines across benchmark trajectories on the Crazyflie (Table III), reducing MED by over 50% and achieving the lowest standard deviation, indicating superior stability. Fly [33] reliably tracks smooth trajectories at varying velocities but struggles with infeasible paths (e.g., fast pentagram and zigzag) due to limited

IEEE Robotics and Automation Letters (RA-L) paper, presented at ICRA 2026, Vienna, Austria. Cite as RA-L paper.

TABLE II
REAL-WORLD TRACKING PERFORMANCE OF SYSID AND DR ON THE FIGURE-EIGHT TRAJECTORY AT NORMAL VELOCITY

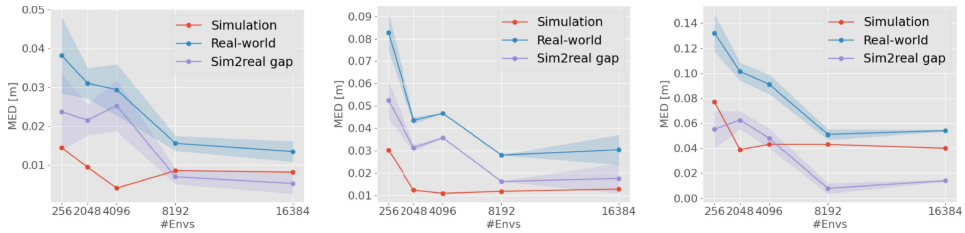
	Offset+30%	Offset+DR30%	SysID+DR30%	SysID+DR10%	SysID
Mass m	∞	∞	0.066(0.007)	0.041(0.006)	**0.028(0.000)**
Inertia $\boldsymbol{I}$	0.041(0.004)	0.046(0.002)	0.053(0.005)	0.036(0.001)	
Motor Time Constant T_m	0.036(0.002)	0.044(0.002)	0.057(0.012)	0.040(0.001)	
Thrust Coefficient k_f	0.107(0.013)	0.035(0.001)	0.050(0.005)	0.034(0.002)	

For sensitive parameters, such as the thrust coefficient k_f, when accurate calibration is not feasible, DR can effectively enhance the robustness of the policy. For non-sensitive parameters, DR primarily increases the learning difficulty without providing significant benefits.

TABLE III
SIMPLEFLIGHT DEMONSTRATES SUPERIOR REAL-WORLD PERFORMANCE ACROSS BENCHMARK TRAJECTORIES

Platform	Methods	Figure-eight			Polynomial	Pentagram		Zigzag
		Slow	Normal	Fast		Slow	Fast	
Crazyflie	Fly [33]	0.093(0.001)	0.181(0.004)	0.282(0.012)	0.289(0.042)	0.104(0.005)	∞	∞
	DATT [11]	0.050(0.009)	∞	∞	0.081(0.019)	0.055(0.004)	0.146(0.012)	0.114(0.019)*
	SimpleFlight (50Hz)	0.035(0.004)	0.056(0.003)	0.114(0.006)	0.042(0.007)	0.031(0.003)	**0.043(0.000)**	0.053(0.009)
	SimpleFlight (100Hz)	**0.016(0.002)**	**0.028(0.000)**	**0.051(0.002)**	**0.032(0.003)**	**0.024(0.001)**	0.045(0.002)	**0.052(0.003)**
Air	PAMPC [38]	**0.011(0.001)**	0.051(0.009)	0.117(0.001)	**0.028(0.006)**	**0.017(0.000)**	0.051(0.002)	0.064(0.007)
	SimpleFlight (Ours)	0.014(0.001)	**0.036(0.000)**	**0.077(0.001)**	**0.028(0.007)**	0.019(0.000)	**0.042(0.001)**	**0.046(0.007)**

On the Crazyflie platform, it reduces MED by over 50% compared to all baselines. On the Air platform, SimpleFlight shows comparable performance to Crazyflie results while outperforming finely-tuned PAMPC, confirming cross-platform generalization. * indicates that for DATT in the zigzag trajectory trials, 4 out of 10 attempts failed; the reported MED reflects the 4 successful trials.



(a) Slow. (b) Normal. (c) Fast.

Fig. 5. Effect of batch sizes on tracking performance on figure-eight trajectories. Increasing the batch size enhances real-world performance as simulation performance converges, with real-world results also stabilizing as the batch size grows further.

long-horizon reasoning. DATT [11] handles infeasible trajectories aggressively but fails in high-velocity tracking on low thrust-to-weight quadrotors. SimpleFlight excels in actuation constraint awareness, long-horizon reasoning, and optimization, particularly for sharp turns and complex maneuvers. At 50 Hz, SimpleFlight shows a minor performance drop compared to 100 Hz but still surpasses DATT, as higher-frequency control enables faster error correction.

We also conduct experiments on our own quadrotor platform, Air, to validate the effectiveness of SimpleFlight. The results demonstrate that our method achieves comparable performance on the Air platform to that on the Crazyflie, slightly outperforming the finely tuned PAMPC. This highlights SimpleFlight's ability to generalize across quadrotor models and sizes. We also provide flight videos on the Air platform on our website.

We remark that the comparison in Table III may not be entirely fair, as the policies are trained using different simulators, modeling approaches, and inputs/outputs. What we aim to convey here is that SimpleFlight, to the best of our knowledge, achieves the best control performance, despite not incorporating any algorithmic or architectural improvements. As a collection of proposed key factors, SimpleFlight can be integrated on top of existing quadrotor control methods.

VI. CONCLUSION

SimpleFlight is an RL framework for robust zero-shot deployment of quadrotor control policies. By incorporating five key factors including enhanced inputs design with velocity and rotation matrix, time vector for critic, regularization of the difference between successive actions as a smoothness reward, selected domain randomization with system identification, and large training batch sizes, it effectively bridges the sim-to-real gap. Evaluations on a Crazyflie quadrotor demonstrate over 50% lower tracking error compared to SOTA RL baselines.

REFERENCES

[1] J. Grzybowski, K. Latos, and R. Czyba, "Low-cost autonomous UAV-based solutions to package delivery logistics," in *Proc. Adv., Contemporary Control: Proc. KKA 20th Polish Control Conf.*, Łódź, Poland: Springer, 2020, pp. 500–507.

[2] J. Scherer et al., "An autonomous multi-UAV system for search and rescue," in *Proc. 1st Workshop Micro Aerial Veh. Netw., Syst., Appl. Civilian Use*, 2015, pp. 33–38.

[3] J. Nikolic, M. Burri, J. Rehder, S. Leutenegger, C. Huerzeler, and R. Siegwart, "A UAV system for inspection of industrial facilities," in *Proc. 2013 IEEE Aerosp. Conf.*, 2013, pp. 1–8.

[4] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2011, pp. 2520–2525.

[5] M. Faessler, A. Franchi, and D. Scaramuzza, "Differential flatness of quadrotor dynamics subject to rotor drag for accurate tracking of high-speed trajectories," *IEEE Robot. Automat. Lett.*, vol. 3, no. 2, pp. 620–626, Apr. 2018.

[6] D. Hanover, P. Foehn, S. Sun, E. Kaufmann, and D. Scaramuzza, "Performance, precision, and payloads: Adaptive nonlinear MPC for quadrotors," *IEEE Robot. Automat. Lett.*, vol. 7, no. 2, pp. 690–697, Apr. 2022.

[7] G. Williams et al., "Information theoretic MPC for model-based reinforcement learning," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2017, pp. 1714–1721.

[8] J. Hwangbo, I. Sa, R. Siegwart, and M. Hutter, "Control of a quadrotor with reinforcement learning," *IEEE Robot. Automat. Lett.*, vol. 2, no. 4, pp. 2096–2103, Oct. 2017.

[9] B. Kiumarsi, K. G. Vamvoudakis, H. Modares, and F. L. Lewis, "Optimal and autonomous control using reinforcement learning: A survey," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 6, pp. 2042–2062, Jun. 2018.

[10] C. Pfeiffer, S. Wengeler, A. Loquercio, and D. Scaramuzza, "Visual attention prediction improves performance of autonomous drone racing agents," *PLoS One*, vol. 17, no. 3, 2022, Art. no. e0264471.

[11] K. Huang, R. Rana, A. Spitzer, G. Shi, and B. Boots, "DATT: Deep adaptive trajectory tracking for quadrotor control," in *Proc. Conf. Robot Learn.*, 2023, pp. 326–340.

[12] F. Gao, C. Yu, Y. Wang, and Y. Wu, "Neural internal model control: Learning a robust control policy via predictive error feedback," *IEEE Robot. Automat. Lett.*, early access, May 23, 2025, doi: 10.1109/LRA.2025.3573169.

[13] Y. Song, M. Steinweg, E. Kaufmann, and D. Scaramuzza, "Autonomous drone racing with deep reinforcement learning," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2021, pp. 1205–1212.

[14] J. Eschmann, D. Albani, and G. Loianno, "Learning to fly in seconds," *IEEE Robot. Automat. Lett.*, vol. 9, no. 7, pp. 6336–6343, Jul. 2024.

[15] E. Kaufmann, L. Bauersfeld, and D. Scaramuzza, "A benchmark comparison of learned control policies for agile quadrotor flight," in *Proc. Int. Conf. Robot. Automat.*, 2022, pp. 10504–10510.

[16] A. Dionigi, G. Costante, and G. Loianno, "The power of input: Benchmarking zero-shot sim-to-real transfer of reinforcement learning control policies for quadrotor control," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2024, pp. 11812–11818.

[17] B. Xu, F. Gao, C. Yu, R. Zhang, Y. Wu, and Y. Wang, "*OmniDrones*: An efficient and flexible platform for reinforcement learning in drone control," *IEEE Robot. Automat. Lett.*, vol. 9, no. 3, pp. 2838–2844, Mar. 2024.

[18] K. Kristinsson and G. A. Dumont, "System identification and control using genetic algorithms," *IEEE Trans. Syst., Man, Cybern.*, vol. 22, no. 5, pp. 1033–1046, Sep./Oct. 1992.

[19] 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 *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2017, pp. 23–30.

[20] J. Tremblay et al., "Training deep networks with synthetic data: Bridging the reality gap by domain randomization," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. Workshops*, 2018, pp. 969–977.

[21] O. M. Andrychowicz et al., "Learning dexterous in-hand manipulation," *Int. J. Robot. Res.*, vol. 39, no. 1, pp. 3–20, 2020.

[22] A. Molchanov, T. Chen, W. Hönig, J. A. Preiss, N. Ayanian, and G. S. Sukhatme, "Sim-to-(multi)-real: Transfer of low-level robust control policies to multiple quadrotors," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2019, pp. 59–66.

[23] A. Loquercio, E. Kaufmann, R. Ranftl, A. Dosovitskiy, V. Koltun, and D. Scaramuzza, "Deep drone racing: From simulation to reality with domain randomization," *IEEE Trans. Robot.*, vol. 36, no. 1, pp. 1–14, Feb. 2020.

[24] Y. Ganin et al., "Domain-adversarial training of neural networks," *J. Mach. Learn. Res.*, vol. 17, no. 59, pp. 1–35, 2016.

[25] K. Bousmalis, N. Silberman, D. Dohan, D. Erhan, and D. Krishnan, "Unsupervised pixel-level domain adaptation with generative adversarial networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 3722–3731.

[26] X. Jing, K. Qian, T. Jianu, and S. Luo, "Unsupervised adversarial domain adaptation for sim-to-real transfer of tactile images," *IEEE Trans. Instrum. Meas.*, vol. 72, 2023, Art. no. 7503011.

[27] M. Long, Y. Cao, J. Wang, and M. Jordan, "Learning transferable features with deep adaptation networks," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 97–105.

[28] Y. Park, S. H. Lee, and I. H. Suh, "Sim-to-real visual grasping via state representation learning based on combining pixel-level and feature-level domain adaptation," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2021, pp. 6300–6307.

[29] S. Gronauer, M. Kissel, L. Sacchetto, M. Korte, and K. Diepold, "Using simulation optimization to improve zero-shot policy transfer of quadrotors," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2022, pp. 10170–10176.

[30] L. Bauersfeld, E. Kaufmann, P. Foehn, S. Sun, and D. Scaramuzza, "NeuroBEM: Hybrid aerodynamic quadrotor model," 2021, *arXiv:2106.08015*.

[31] E. Kaufmann, L. Bauersfeld, A. Loquercio, M. Müller, V. Koltun, and D. Scaramuzza, "Champion-level drone racing using deep reinforcement learning," *Nature*, vol. 620, no. 7976, pp. 982–987, 2023.

[32] D. Zhang, A. Loquercio, X. Wu, A. Kumar, J. Malik, and M. W. Mueller, "Learning a single near-hover position controller for vastly different quadcopters," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2023, pp. 1263–1269.

[33] J. Eschmann, D. Albani, and G. Loianno, "Learning to fly in seconds," *IEEE Robot. Automat. Lett.*, vol. 9, no. 7, pp. 6336–6343, Jul. 2024.

[34] F. Furrer, M. Burri, M. Achtelik, and R. Siegwart, "Rotors–A modular gazebo MAV simulator framework," in *Robot Operating System (ROS): The Complete Reference (Volume 1)*. Berlin, Germany: Springer, 2016, pp. 595–625.

[35] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "AirSim: High-fidelity visual and physical simulation for autonomous vehicles," in *Proc. Field Serv. Robot.: Results 11th Int. Conf.*, Springer, 2018, pp. 621–635.

[36] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017, *arXiv:1707.06347*.

[37] Y. Zhou, C. Barnes, J. Lu, J. Yang, and H. Li, "On the continuity of rotation representations in neural networks," in *Proc IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 5738–5746.

[38] D. Falanga, P. Foehn, P. Lu, and D. Scaramuzza, "PAMPC: Perception-aware model predictive control for quadrotors," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2018, pp. 1–8.

[39] G. Williams et al., "Information theoretic MPC for model-based reinforcement learning," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2017, pp. 1714–1721.