Environment as Policy: Learning to Race in Unseen Tracks

Hongze Wang*, Jiaxu Xing*, Nico Messikommer, and Davide Scaramuzza

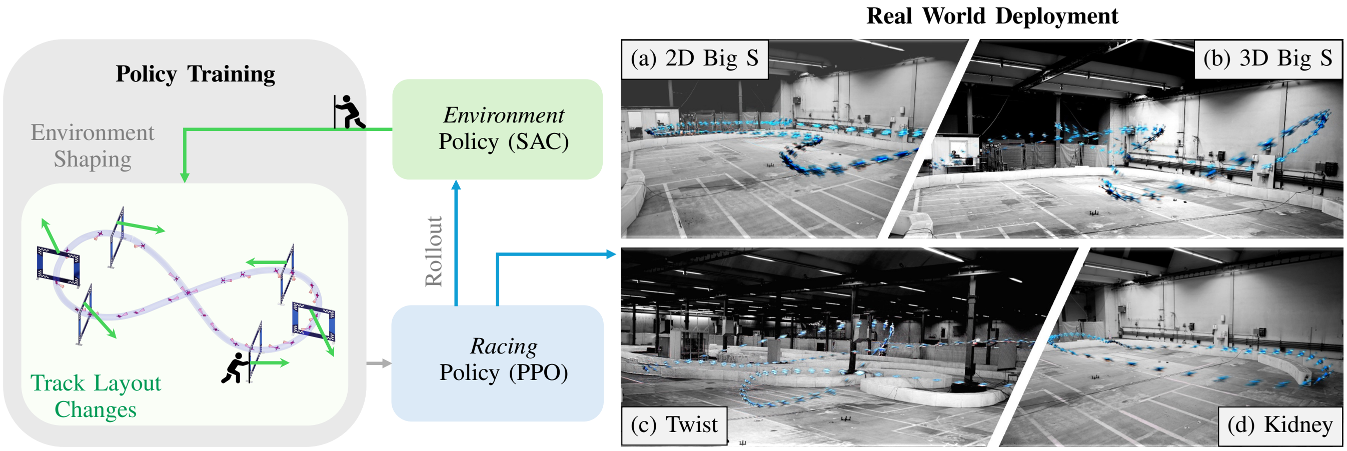


Fig. 1: In this work, we introduce an environment-shaping framework to improve the generalization of RL drone racing agents to diverse and unseen tracks without retraining. We use a Soft Actor-Critic (SAC) policy to adaptively shape the track layouts by generating difficult but achievable race tracks based on the racing agent's actual performance. The resulting *single* racing policy can fly in various unseen and challenging race tracks in the real world with competitive lap times.

Abstract*— Reinforcement learning (RL) has achieved outstanding success in complex robot control tasks, such as drone racing, where the RL agents have outperformed human champions in a known racing track. However, these agents fail in unseen track configurations, always requiring complete retraining when presented with new track layouts. This work aims to develop RL agents that generalize effectively to novel track configurations without retraining. The naïve solution of training directly on a diverse set of track layouts can overburden the agent, resulting in suboptimal policy learning as the increased complexity of the environment impairs the agent's ability to learn to fly. To enhance the generalizability of the RL agent, we propose an adaptive environment-shaping framework that dynamically adjusts the training environment based on the agent's performance. We achieve this by leveraging a secondary RL policy to design environments that strike a balance between being challenging and achievable, allowing the agent to adapt and improve progressively. Using our adaptive environment shaping, one single racing policy efficiently learns to race in diverse and challenging tracks. Experimental results validated in both simulation and the real world show that our method enables drones to successfully fly complex and unseen race tracks, outperforming existing environment-shaping techniques. Website:https://environment-as-policy.github.io/

I. Introduction

Reinforcement learning (RL) involves agents learning through trial and error by interacting with a pre-defined environment and maximizing the rewards based on these interactions. It has proven highly effective in various robotic control applications, demonstrating remarkable task performance across scenarios like dexterous manipulation [1], [2], [3], quadrupedal locomotion [4], [5], and agile quadrotor flight [6], [7], [8]. Previous research has shown that RL can even outperform human champions in the task of drone racing [9], [10], where an RL agent flies a quadcopter drone through a complex track at high speed, requiring quick decision-making and precise control.

However, while RL agents excel within the specific distributions they are trained on, they struggle with out-of-distribution configurations and may require retraining from scratch for even minor configuration changes [11]. To improve adaptability and generalization, extending the RL framework to train and perform across a broader range of distributions is essential, enabling agents to handle more diverse and dynamic environments effectively. However, training directly on a wide distribution of scenarios can significantly degrade the efficiency and effectiveness of the learning process [12]. As the distribution broadens, the complexity of policy exploration increases, making it harder for the agent to identify and implement effective solutions.

Domain randomization is a commonly used technique to improve the learning capability of RL agents across a broader range of tasks [11], [13], [14], [15]. This approach varies the parameters of the training environment to expose the agent to a wide variety of potential deployment scenarios. By learning in diverse conditions, the agent develops robust policies less likely to overfit the specific characteristics of a single

* equal contribution. The authors are with the Robotics and Perception Group, Department of Informatics, University of Zurich, and Department of Neuroinformatics, University of Zurich and ETH Zurich, Switzerland (http://rpg.ifi.uzh.ch). This work was supported by the European Union's Horizon Europe Research and Innovation Programme under grant agreement No. 101120732 (AUTOASSESS) and the European Research Council (ERC) under grant agreement No. 864042 (AGILEFLIGHT).

environment. Domain randomization is often combined with curriculum learning, where the complexity and variability of training scenarios are incrementally increased [15]. By starting with less challenging environments and gradually introducing greater variability, the agent learns to perform well in a wide set of scenarios without becoming overwhelmed during the early stages of training. However, curriculum learning often depends on manual design, introducing human biases. This reliance on fixed, non-adaptive progressions can limit training diversity and restrict the agent's ability to generalize to new, real-world situations [16]. These effects are especially prominent in the challenging task of drone racing, where the platform's high agility and the need for rapid decision-making amplify the difficulties. Therefore, the challenge of agile drone racing on an unseen race track in real-world conditions remains largely unexplored.

To address this problem, we propose an automatic adaptive environment-shaping framework to enhance the agent's generalization ability to fly in unseen race tracks. This framework dynamically adjusts the environment curriculum based on the agent's learning progress (Fig. 1). The key idea is to create consistently challenging yet attainable environments, avoiding tracks that are too easy or overly difficult, which could hinder learning. We achieve this by leveraging a secondary RL agent to design environments that maintain a balanced difficulty level, allowing the agent to adapt and improve progressively. By automating the environment shaping, the system can more precisely tailor the learning environment to the agent's performance, reducing human bias and significantly enhancing the agent's ability to generalize across various unseen track configurations. In experiments validated both in simulation and the real world, we demonstrate for the first time a drone racing policy that can race in different unseen tracks. We show that our approach outperforms the existing environment-shaping approaches using the same number of actions and enables the racing policy to generalize to a diverse set of unseen and complicated tracks.

II. Related Works

Reinforcement Learning for Robotics Reinforcement Learning (RL) has been extensively applied in robotics to facilitate continuous control in complex and dynamic environments. Recent studies have achieved remarkable success in various robotic domains. These include quadrotor control [9], [10], [17], [8], [18], legged locomotion [19], [5], and manipulation [20], [21], showcasing significant advancements in the field.

Autonomous Drone Racing Autonomous drone racing is an emerging field in which drones must navigate through predefined waypoints in the shortest possible time. Success in this field requires integrating advanced hardware and sophisticated algorithms capable of perceiving the environment, planning optimal paths, and executing actions in real-time [22]. [10] demonstrated that RL controllers can outperform traditional optimization-based approaches, such as those proposed in [23], [24], for autonomous drone racing. This advantage arises because RL can effectively handle highly non-linear dynamical systems and complex objectives, which pose significant challenges for conventional optimization methods like Model Predictive Control (MPC) [25], [26]. Recent studies have shown that policies trained with deep reinforcement learning (DRL) can surpass even human world champions, employing vision-based state estimation for the RL controller. Furthermore, [27], [18] have demonstrated using RL to learn drone racing from image inputs directly.

Environment Shaping for Reinforcement Learning Environment shaping has been widely applied in reinforcement learning by designing a distribution of environments that progressively improves the agent's performance. Previous works like [28], [29], [30], [31] on environment shaping have been successful in simulation tasks like Bipedal Walker and maze games, where environments are easily parameterized for manual curriculum design. However, this process becomes more challenging for more complex real-world robotic tasks. Domain randomization is a common solution, where environments are sampled from a predefined range to enable robust task execution. For instance, [15], [14] used domain randomization to achieve dexterous manipulation in the real world. Additionally, [4] proposed an adaptive terrain curriculum using a particle filter to sample environment parameters, enabling quadrupedal locomotion across various terrains in real-world scenarios.

III. Methodology

To enable agile flight through unseen tracks, our approach introduces an *Environment Policy* that acts as a separate agent, guiding the racing policy's training across varied tracks. Both policies are employed in an alternating fashion during training: the environment policy creates tracks tailored to the drone agent's learning progress, enhancing the racing policy's robustness and speed. An overview of the method is visualized in Fig. 2. In the following, we introduce the racing policy in Sec. III-A and describe the proposed environment policy in Sec. III-B.

A. Racing Policy Training

The autonomous racing task can be framed as an optimization problem, where the objective is to minimize the time it takes for an agile quadrotor to pass through a predefined sequence of gates [22], as illustrated in Fig. 3. In this task, we define the observations as $\boldsymbol{o}_{\text{racing}} = \left[\boldsymbol{p}, \tilde{\boldsymbol{R}}, \boldsymbol{v}, \boldsymbol{\omega}, a_{\text{prev}}, \delta\boldsymbol{p}_1, \delta\boldsymbol{p}_2\right]$, where $\boldsymbol{p} \in \mathbb{R}^3$ denotes the drone's position, $\tilde{\boldsymbol{R}} \in \mathbb{R}^6$ is a vector comprising the first two columns of $\boldsymbol{R}_{WB}$ [32], $\boldsymbol{v} \in \mathbb{R}^3$ and $\boldsymbol{\omega} \in \mathbb{R}^3$ denote the linear and angular velocity of the drone, a_{prev} represents the previous action from the actor policy, and $\delta\boldsymbol{p}_1, \delta\boldsymbol{p}_2 \in \mathbb{R}^{12}$ represent the relative difference in position of the four next gate corners (4×3) in the world frame. Here, $\delta\boldsymbol{p}_1$ represents the difference of the four corners of the next gate to pass between the current quadrotor position, and $\delta\boldsymbol{p}_2$ represents the positional difference of the corners between the next gate to pass and the gate after the next gate to pass on the race

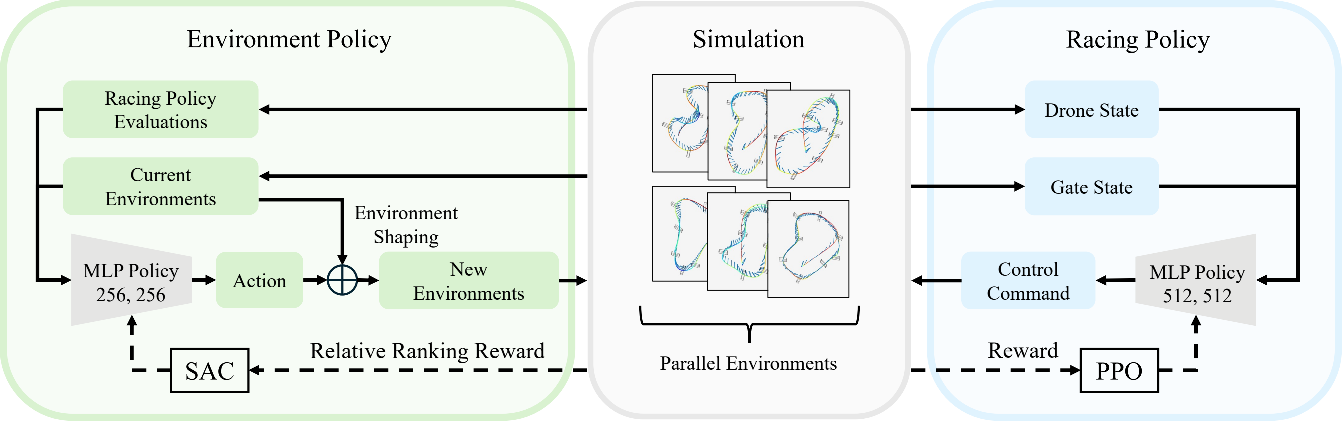


Fig. 2: Overview of the proposed method. In every N iteration, the environment policy *(left)* takes as input the information of the racing policy evaluations and the current environments. It generates actions to adjust the gate layouts independently for each parallel environment. The racing policy (right) utilizes the information about drone and gate states from these simulation environments to learn time-optimal drone racing strategies through an MLP.

track. The total reward at time t, denoted as r_t, consists of several components:

$$r_t^{\text{racing}} = r_t^{\text{prog}} + r_t^{\text{act}} + r_t^{\text{br}} + r_t^{\text{pass}} + r_t^{\text{crash}}, \tag{1}$$

where r_t^{prog} represents progress toward passing the next gate [33], r_t^{act} penalizes changes in actions from the previous time step, r_t^{br} discourages high body rates to ensure a stable flying behavior, r_t^{pass} is a binary reward for successfully passing the next gate, and r_t^{crash} is a binary penalty applied when a collision occurs, which also terminates the episode. The reward components are formulated as follows

$$\begin{aligned} r_t^{\text{prog}} &= \alpha_1 (d_{\text{Gate}}(t-1) - d_{\text{Gate}}(t)), \\ r_t^{\text{act}} &= \alpha_2 \|\boldsymbol{u}_t - \boldsymbol{u}_{t-1}\|, \\ r_t^{\text{br}} &= \alpha_3 \|\boldsymbol{\omega}_{\mathcal{B},t}\|, \\ r_t^{\text{pass}} &= \alpha_4 \quad \text{if robot passes the next gate}, \\ r_t^{\text{crash}} &= \alpha_5 \quad \text{if robot crashes (gates, ground).} \end{aligned} \tag{2}$$

B. Environment Policy Training

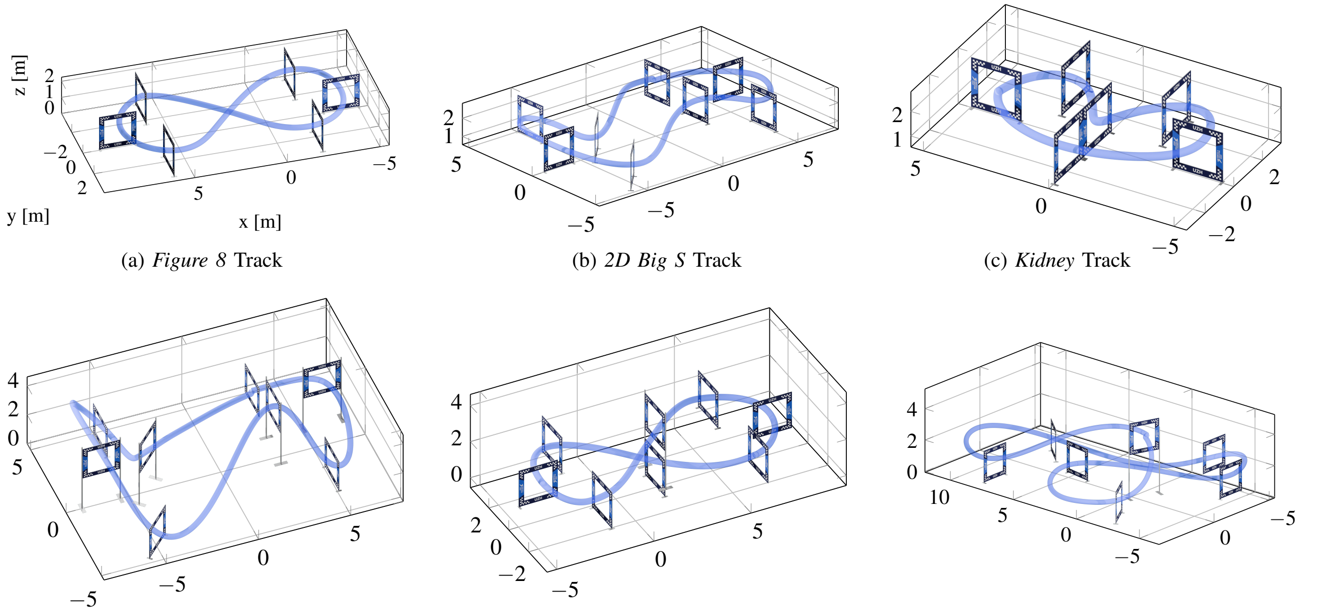
The track layout used for training the racing policy plays a critical role in shaping its flying capabilities while also impacting the overall training stability. If the tracks are too difficult, the agent will struggle to extract meaningful learning signals. Conversely, simple tracks fail to challenge the agent, limiting its ability to generalize to more complex environments. Thus, to ensure effective learning, the track difficulty must be *continuously adapted to the current capabilities of the agent*. To achieve this, we introduce a learned environment policy π_{env} that dynamically adjusts the race tracks. This adaptive approach allows the agent to consistently gather relevant and progressive learning experiences, optimizing its training stability and performance in diverse race tracks.

1) MDP Formulation: In our racing scenario, the states of the environment are represented by the position $\boldsymbol{p}$ and orientation $\boldsymbol{R}$ of each individual gate, such that the full gate state vector is given by $\boldsymbol{s}_{\text{gates}} = [\boldsymbol{p}_1, \boldsymbol{R}_1, \ldots, \boldsymbol{p}_N, \boldsymbol{R}_N]$, where N is the total number of gates in the environment. The environment policy leverages the observation $\boldsymbol{o}_{\text{env}} = [\boldsymbol{s}_{\text{gates}}, \boldsymbol{e}_{\text{racing}}]$, where s_{gates} represents the current states of all gates, and $\boldsymbol{e}_{\text{racing}}$ indicates the performance of the drone agent achieved in the track corresponding to the environment state s_{gates}. The evaluation performance vector $\boldsymbol{e}_{\text{racing}} = [\boldsymbol{e}_1, \ldots, \boldsymbol{e}_N]$ contains the gate-passing error for each of the N gates, where the error $\boldsymbol{e}_i$ is computed as the distance between the drone's position and the center of gate i at the moment the drone passes through the gate. The environment policy outputs actions $a = [\Delta \boldsymbol{p}_{\text{gates}}, \Delta \text{yaw}_{\text{gates}}]$, where $\Delta \boldsymbol{p}_{\text{gates}}$ specifies changes in gate positions, and $\Delta\text{yaw}_{\text{gates}}$ indicates changes in the yaw angles of the gates, both in the world frame coordinates. Hence, the transition dynamics $\mathbb{P}$ is naturally $s_{\text{gates}}^t = s_{\text{gates}}^{t-1} + a^{t-1}$. As training progresses, the training track layout reflects accumulated changes from the initial track, naturally producing various tracks.

Another key component of our framework is the development of a metric that precisely measures the effectiveness of the environment policy's actions on the performance of the racing agent. In our work, we propose a reward for the environment policy grounded in the *relative ranking* of the performance of the racing policy in different environments. Specifically, after each training phase, we rank all the parallel training environments according to the number of gates the agent successfully passed. The higher the ranking number, the worse the policy performs. The environment policy is then penalized for generating track layouts at the extremes of the ranking—those that are either too easy or too difficult. Additionally, we reward race tracks that fall within an intermediate range, where the agent demonstrates a degree of success but has not fully mastered the track. The reward is defined as follows:

$$r_t^{env} = \begin{cases} R\frac{N_{\text{env}} - \text{rank}}{N_{\text{env}} - r_{\text{upper}}} & \text{if } rank > r_{\text{upper}} \\ R & \text{if } rank \in [r_{\text{lower}}, r_{\text{upper}}] \\ R\frac{\text{rank}}{r_{\text{lower}}} & \text{if } rank < r_{\text{lower}} \end{cases} \tag{3}$$

where R represents a positive constant, r_{lower} and r_{upper} are the fixed thresholds determining the mid-range representing the tracks that are not too easy and not too hard, N_{env}



(a) *Figure 8* Track

(b) *2D Big S* Track

(c) *Kidney* Track

(d) *3D Big S* Track

(e) *3D Figure 8* Track

(f) *Twist* Track

Fig. 3: Visualization of the drone racing tracks used for the experiments, each characterized by varying levels of complexity. All the tracks maintain a consistent size scale, spanning widths from 8 meters to 16 meters.

represents the total number of parallel training environments. This formulation represents a smooth reward function that provides fine-grained feedback on the environment policy for each generated training track.

2) Environment Shaping: To accelerate data collection, we train our environment agent and the racing agent using multiple parallel simulation environments. To handle varying numbers of environments, the environment policy independently updates the track layout for each environment. Since all environments initially share the same track layout, the racing policy experiences a gradual increase in layout complexity, which facilitates smoother learning during the early stages of training. We use the notation $\text{Env}_i \leftarrow \text{Env}_{i-1} \oplus a_{i-1}$ to represent changes in the individual environment. For each new environment, the environment policy is trained for every N_{freq} racing policy update, where N_{freq} determines how much more frequent should π_{racing} than π_{env}. Since N_{freq} is typically much greater than 1 (i.e., $N_{\text{freq}} \gg 1$), the environment policy is updated less frequently than the racing policy. To handle this, we choose the off-policy RL algorithm Soft Actor-Critic (SAC) [34] to update the parameters of the environment policy. The complete framework containing the environment policy and the racing policy during the training is presented in Algorithm 1.

Algorithm 1 Adaptive Environment Shaping

```
Initialize: $\pi_{\text{agent}}$, $\pi_{\text{env}}$, and gate layout $\text{Env}_0$
for $i \leftarrow 1, 2, ..., N_{\text{epoch}}$ do
    Step 1: Generate New Environments
    if $i == 1$ then
        $a_0 \leftarrow$ random sample from the action space of $\pi_{\text{env}}$
        $\text{Env}_1 \leftarrow \text{Env}_0 \oplus a_0$
    else
        $\text{Env}_i \leftarrow \text{Env}_{i-1} \oplus \pi_{\text{env}}(\boldsymbol{s}_{\text{gates}}, \boldsymbol{e}_{\text{racing}})$
    end if
    Step 2: Train racing policy
    for $j \leftarrow 1, 2, ..., N_{\text{freq}}$ do
        Train $\pi_{\text{agent}}$ on updated $\text{Env}_i$
    end for
    Step 3: Evaluate and update environment policy
    Evaluate agent performance across tracks $\text{Env}_i$
    Compute relative ranking reward
    Update environment policy $\pi_{\text{env}}$
end for
```

IV. Experiments

In this section, we first introduce the detailed experimental setups in Sec. IV-A, followed by a discussion of the baseline approaches used for comparison in Sec. IV-B. Our experiments aim to address the following key research questions: (i) How does our approach generalize to unseen race tracks? (ii) How does our environment policy perform on different training configurations? (iii) Can our generalist racing policy even fly in dynamic race tracks with moving gates? (iv) Does our policy transfer to the real world?

A. Experimental Setup

Our training framework is built on the Flightmare simulator [36], with reward functions and PPO hyperparameters aligned with methods from previous research [9]. For our environment policy, we implemented a vectorized Gym [37] environment with the same number of environments as the agent, using a total of 100 environments. The environment policy uses the SAC implementation from Stable Baselines 3 [38], with fine-tuned parameters for our task setting. Specifically, we set the gradient steps of SAC to 100 to

Track Type	Track Name	Methods									
		Single-track RL		*RL w/o curriculum*		*Particle Filter*		*Domain Randomization*		*Ours*	
		SR [%]	LT [s]	SR [%]	LT [s]	SR [%]	LT [s]	SR [%]	LT [s]	SR [%]	LT [s]
2D	*Figure 8*	100.00	4.263	0.00	-	**100.00**	5.128	**100.00**	6.205	**100.00**	**4.746**
	Kidney	100.00	4.260	0.00	-	0.00	-	**100.00**	4.984	**100.00**	**4.943**
	Big S	100.00	9.245	0.00	-	0.00	-	0.00	-	**100.00**	**7.513**
3D	3D *Figure 8*	100.00	5.010	0.00	-	**100.00**	**5.654**	0.00	-	**100.00**	5.856
	3D *Big S*	100.00	10.187	0.00	-	0.00	-	0.00	-	**100.00**	**9.761**
	Twist	100.00	7.444	0.00	-	0.00	-	0.00	-	**100.00**	**10.199**

TABLE I: We compare the success rate (SR) and lap time (LT) of our method against four baselines. Six different unseen racetracks are evaluated in a realistic BEM simulation [35], including three 2D tracks and three 3D tracks. Here apart from the our and baseline approaches, we also include the *Single-track RL* as a reference, which is a vanilla PPO agent trained and tested on individual track.

accelerate the training of the environment policy and the entropy coefficient to 1.0 for better exploration.

During the training process, the racing policy is updated at every iteration, while the environment policy is updated every 100 iterations. We sometimes encounter environments getting stuck on infeasible tracks during training. To address this, if the evaluation SR stays at zero for three consecutive runs, we reset the environment to the initial race track. We selected an 8-gate oval track for the initial layout due to its simplicity, making it easier for the racing policy to learn and adapt quickly. For the environment policy's action space, we constrained the gate movements as follows: along the x and y axes, the range is $[-1.0, 1.0]\mathrm{m}$; along the z-axis, the range is $[-0.2, 0.2]\mathrm{m}$, and for the yaw, the range is $[-\frac{\pi}{30}, \frac{\pi}{30}]\ \mathrm{rad}$. For the relative ranking range, we set r_{lower} to the 50th percentile of all environments and r_{upper} to the 90th percentile. These hyperparameters are empirically found to lead to the best performance during the ablation experiments. Our method is trained for 800M data samples, which is equivalent to 12 hours of training using an NVIDIA TITAN Xp GPU.

To evaluate the performance of our approach, we tested the racing policy on several fixed racetracks with varying difficulties and complexities. We selected three 2D tracks, where all gates are positioned at the same height along the z-axis, including *Figure 8*, *Kidney*, and *2D Big S*. Additionally, we chose three 3D tracks, where the gates have varying heights along the z-axis, including *3D Figure 8*, *3D Big S*, *and Twist*, as shown in Fig. 3. All of these tracks are significantly different than the initial training track, with most having a different number of gates than during training. For the evaluation, we primarily used two metrics: success rate (SR %) and lap time (LT [s]). The success rate is calculated as the ratio of successful runs, where the drone passes all the gates without crashing, over the total number of trials. Lap time refers to the drone's total time to successfully complete a race track. These metrics are commonly used in drone racing and are essential in evaluating whether the policy enables fast and stable flight performance [27], [18].

B. Baselines.

To evaluate the generalization ability of our proposed methods, we compared them with three baseline methods: (i) RL policy without curriculum: This method shares the same initial environment setup as ours but does not use a curriculum for training. (ii) Domain randomization from [6]: In this approach, the initial environment is the same as in our method. However, environment policy actions are sampled randomly within the action space; we use the identical configurations from [6]. (iii) Particle Filter Curriculum from [4]: This method also uses the same initial environment as ours but relies on a particle filter to sample environments.

How does our approach generalize to unseen race tracks? We first evaluated our method on six unseen tracks to assess its generalization ability across tracks with varying numbers of gates and different layouts compared to those used during training. Table I presents the performance results. As can be observed, the reference single-track RL policies on the left side demonstrate the best performance in most experiments. This is because these policies overfit the specific track, allowing them to optimize and perform well in that particular environment. Consequently, the results from the RL method without a curriculum show that directly training on one track without a curriculum and testing on different tracks is not successful.

Additionally, we found that a simple curriculum has a limited effect on improving the racing policy's generalization ability. Methods such as domain randomization and particle filters can enhance the agent's generalization ability, allowing it to fly on some simple unseen tracks, such as *Figure 8* or *Kidney* tracks. However, since these methods do not take into account the continuous improvement of the agent's policy during training or the evolution of the environment, they fail to perform well in rather complicated unseen tracks, e.g., in *2D Big S* or *Twist* tracks. Only our proposed method demonstrated stable and successful flight across all six unseen tracks. Additionally, we observed that for most 2D and 3D tracks (excluding the *Twist* track), the policy's performance (lap time) remains close to that of the *single-track RL policy*, within a 1-second difference. This difference can be explained by the increased generalization of our policy, which did not memorize a single specific track. This confirms that our training approach maintains strong task performance while enhancing generalization.

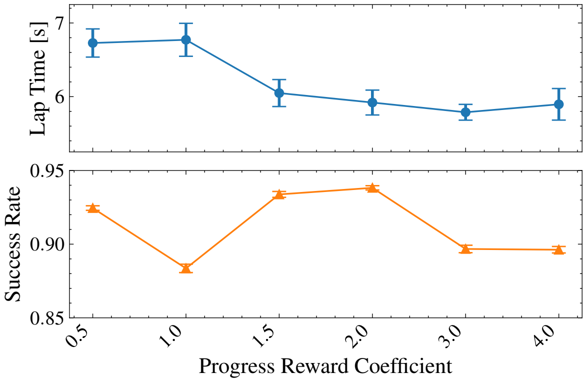


Fig. 4: Ablation study on the progress reward. Due to the fluctuations in evaluation results across different iterations, to fairly compare the performance of different coefficients, we take the average and variance of the success rate and lap time after the model has stabilized for comparison.

How does our environment policy perform on different training configurations? To assess the impact of different configurations on the performance of our environment policy in racing policy training, we performed an ablation study. Within the same parameter setting for the environment policy, we vary the progress reward coefficients α_1 from Equation 2 of the racing policy. To the variance in the results, we tested the different configurations on 100 randomly generated tracks, which were manually filtered for feasibility. As shown in Fig. 4, within a large range of parameters, increasing the progress reward coefficient helps the agent learn to fly faster, as evidenced by the decreasing lap times on specific tracks. At the same time, we observe no significant decrease in generalization, as the success rate remains fairly stable. This indicates the robustness of our framework, where a dedicated human-defined curriculum usually needs to be re-designed for different speed ranges.

Can our generalist racing policy fly in dynamic race tracks with moving gates? To evaluate the generalization of our method, we further test the racing policy by deploying it on a dynamic track. In this scenario, several gates move at a constant speed in predefined directions. Notably, this dynamic setting was never included in the training configurations of the racing policy. As the track continuously changes, the racing policy's observations evolve over time. In this setting, only a highly robust control ability can finish the lap and prevent crashes. In this experiment, we chose the *Figure 8* track, where the third and fourth gates move dynamically along their y-axis at a constant speed of $0.6\,\mathrm{m\,s^{-1}}$ within the ranges of [-0.5, 0.5] m, [-1.0, 1.0] m, and [-2.0, 2.0] m. As can be observed in ??, we can see that the Domain Randomization [6] and Particle Filter [4] methods still achieve partial success when the gates move within the range of [-0.5, 0.5] m. However, both methods fail when the range increases to [-1.0, 1.0] m. In contrast, our method can handle perturbations up to [-2.0, 2.0] m. This further demonstrates our method's superior generalization ability, enabling it to adapt its actions as observations change and maintain stable performance even in the face of significant dynamic variations. Also, we test the current limit of our approach, where we further increase the gate moving speed to $0.75\,\mathrm{m\,s^{-1}}$. In this setting, our policy can still achieve more than 30% success rate.

Range [m]	**Methods**	SR [%]	LT [s]
Static	Single-track RL	100.00	4.263
	Particle Filter	100.00	5.128
	Domain Randomization	100.00	6.205
	Ours	**100.00**	**4.746**
$[-0.5, 0.5]$	Particle Filter	54.69	5.512
	Domain Randomization	93.75	6.152
	Ours	**100.00**	**5.388**
$[-1, 1]$	Particle Filter	0.00	-
	Domain Randomization	0.00	-
	Ours	**100.00**	**5.276**
$[-2, 2]$	Particle Filter	0.00	-
	Domain Randomization	0.00	-
	Ours (0.6m/s)	**100.00**	**5.329**
	Ours (0.75m/s)	**31.25**	**5.067**

TABLE II: Results of flying on a dynamic *Figure 8* racetrack. We compare the success rate (SR) and lap time (LT) of different methods.

Does our policy transfer in the real world? To validate the effectiveness of our proposed method, we conducted tests in real-world conditions. We used the Agilicious quadrotor platform [39] with precise state estimation provided by a VICON motion capture system, ensuring accurate inputs for the policy. The BetaFlight2 firmware was employed for low-level control to execute the collective thrusts and body rate commands. We performed nine laps on each of the six tracks (Fig. 3), demonstrating that our single racing policy can successfully navigate these previously unseen tracks in the real world with a success rate of 100%, as shown on the right side of Fig. 1. For further details, we invite readers to view the supplementary video.

V. Conclusion

In this work, we proposed an adaptive environment-shaping framework enabling, for the first time, a learned policy to race on unseen and dynamic tracks. Our method works by leveraging a secondary environment policy that shapes the training environments for a drone racing policy. Using our novel relative ranking reward, our environment policy can generate track layouts that are challenging but feasible based on the racing agents' performance. One single drone racing policy trained with our framework can race in various race tracks with different complexity with 100% success rate, whereas the state-of-art curriculum learning approach mostly cannot fly. Furthermore, we validated the policy's generalization ability by racing on a track with moving gates, where existing methods performed significantly worse at adapting to the fast-changing gate positions. We believe our work represents a significant advancement in enabling agile robots to achieve greater generalization and robustness in complex, dynamic, and open environments.

References

[1] O. M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, *et al.*, "Learning dexterous in-hand manipulation," *The International Journal of Robotics Research*, vol. 39, no. 1, pp. 3–20, 2020.

[2] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke, *et al.*, "Scalable deep reinforcement learning for vision-based robotic manipulation," in *Conference on robot learning*, pp. 651–673, PMLR, 2018.

[3] D. Kalashnikov, J. Varley, Y. Chebotar, B. Swanson, R. Jonschkowski, C. Finn, S. Levine, and K. Hausman, "Scaling up multi-task robotic reinforcement learning," in *Conference on Robot Learning*, 2022.

[4] J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter, "Learning quadrupedal locomotion over challenging terrain," *Science robotics*, vol. 5, no. 47, p. eabc5986, 2020.

[5] A. Kumar, Z. Fu, D. Pathak, and J. Malik, "Rma: Rapid motor adaptation for legged robots," *Proceedings of Robotics: Science and Systems (RSS)*, 2021.

[6] Y. Song, M. Steinweg, E. Kaufmann, and D. Scaramuzza, "Autonomous drone racing with deep reinforcement learning," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1205–1212, IEEE, 2021.

[7] N. Messikommer, Y. Song, and D. Scaramuzza, "Contrastive initial state buffer for reinforcement learning," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2024.

[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 international conference on robotics and automation (ICRA)*, IEEE, 2024.

[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] Y. Song, A. Romero, M. Mueller, V. Koltun, and D. Scaramuzza, "Reaching the limit in autonomous racing: Optimal control versus reinforcement learning," *Science Robotics*, p. adg1462, 2023.

[11] 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," 2017.

[12] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *Proceedings of the 26th annual international conference on machine learning*, pp. 41–48, 2009.

[13] J. Tobin, L. Biewald, R. Duan, M. Andrychowicz, A. Handa, V. Kumar, B. McGrew, J. Schneider, P. Welinder, W. Zaremba, and P. Abbeel, "Domain randomization and generative models for robotic grasping," 2018.

[14] OpenAI, M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, J. Schneider, S. Sidor, J. Tobin, P. Welinder, L. Weng, and W. Zaremba, "Learning dexterous in-hand manipulation," 2019.

[15] 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.

[16] Y. Kong, L. Liu, J. Wang, and D. Tao, "Adaptive curriculum learning," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 5067–5076, 2021.

[17] J. Hwangbo, I. Sa, R. Siegwart, and M. Hutter, "Control of a quadrotor with reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 2, no. 4, pp. 2096–2103, 2017.

[18] J. Xing, A. Romero, L. Bauersfeld, and D. Scaramuzza, "Bootstrapping reinforcement learning with imitation for vision-based agile flight," *arXiv preprint arXiv:2403.12203*, 2024.

[19] 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.

[20] Z. Fu, X. Cheng, and D. Pathak, "Deep whole-body control: learning a unified policy for manipulation and locomotion," in *Conference on Robot Learning*, pp. 138–149, PMLR, 2023.

[21] S. Gu, E. Holly, T. Lillicrap, and S. Levine, "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates," in *2017 IEEE international conference on robotics and automation (ICRA)*, pp. 3389–3396, IEEE, 2017.

[22] D. Hanover, A. Loquercio, L. Bauersfeld, A. Romero, R. Penicka, Y. Song, G. Cioffi, E. Kaufmann, and D. Scaramuzza, "Autonomous drone racing: A survey," *IEEE Transactions on Robotics*, 2024.

[23] A. Romero, S. Sun, P. Foehn, and D. Scaramuzza, "Model predictive contouring control for time-optimal quadrotor flight," *IEEE Transactions on Robotics*, vol. 38, no. 6, pp. 3340–3356, 2022.

[24] A. Romero, Y. Song, and D. Scaramuzza, "Actor-critic model predictive control," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 14777–14784, IEEE, 2024.

[25] D. Falanga, P. Foehn, P. Lu, and D. Scaramuzza, "Pampc: Perception-aware model predictive control for quadrotors," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1–8, IEEE, 2018.

[26] J. Xing, G. Cioffi, J. Hidalgo-Carrió, and D. Scaramuzza, "Autonomous power line inspection with drones via perception-aware mpc," in *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1086–1093, IEEE, 2023.

[27] I. Geles, L. Bauersfeld, A. Romero, J. Xing, and D. Scaramuzza, "Demonstrating agile flight from pixels without state estimation," in *Proceedings of Robotics: Science and Systems*, 2024.

[28] R. Portelas, C. Colas, K. Hofmann, and P.-Y. Oudeyer, "Teacher algorithms for curriculum learning of deep rl in continuously parameterized environments," 2019.

[29] J. Parker-Holder, M. Jiang, M. Dennis, M. Samvelyan, J. Foerster, E. Grefenstette, and T. Rocktäschel, "Evolving curricula with regret-based environment design," 2023.

[30] M. Jiang, M. Dennis, J. Parker-Holder, J. Foerster, E. Grefenstette, and T. Rocktäschel, "Replay-guided adversarial environment design," 2022.

[31] M. Dennis, N. Jaques, E. Vinitsky, A. Bayen, S. Russell, A. Critch, and S. Levine, "Emergent complexity and zero-shot transfer via unsupervised environment design," 2021.

[32] Y. Zhou, C. Barnes, J. Lu, J. Yang, and H. Li, "On the continuity of rotation representations in neural networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 5745–5753, 2019.

[33] Y. Song, M. Steinweg, E. Kaufmann, and D. Scaramuzza, "Autonomous drone racing with deep reinforcement learning," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1205–1212, 2021.

[34] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," 2018.

[35] L. Bauersfeld, E. Kaufmann, P. Foehn, S. Sun, and D. Scaramuzza, "Neurobem: Hybrid aerodynamic quadrotor model," in *Proceedings of Robotics: Science and Systems*, 2021.

[36] Y. Song, S. Naji, E. Kaufmann, A. Loquercio, and D. Scaramuzza, "Flightmare: A flexible quadrotor simulator," in *Conference on Robot Learning*, 2020.

[37] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016.

[38] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021.

[39] P. Foehn, E. Kaufmann, A. Romero, R. Penicka, S. Sun, L. Bauersfeld, T. Laengle, G. Cioffi, Y. Song, A. Loquercio, and D. Scaramuzza, "Agilicious: Open-source and open-hardware agile quadrotor for vision-based flight," *Science Robotics*, vol. 7, no. 67, 2022.