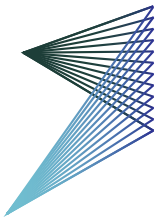
University of Zurich[UZH]



ROBOTICS & PERCEPTION GROUP

Jonathan Jacob Kolt Green

Zero-Shot Generalisation with Reinforcement Learning for Agile Drone Flight

Master Thesis

Robotics and Perception Group
University of Zurich

Supervision

Jiaxu Xing
Nico Messikommer
Angel Romero Aguilar
Prof. Dr. Davide Scaramuzza

October 2025

Contents

Abstract

Reinforcement Learning (RL) has unlocked new levels of capability for autonomous drone racing. However, it falls short on zero-shot generalisation (ZSG), as learned controllers typically cannot successfully race on unseen tracks. In this work we examine the problem of ZSG for autonomous drone racing, explore what contributes to improved generalisation, and propose a multi-task method that leverages parallelisation to train an RL agent capably of successful high-speed flight on complex unseen racetracks. Our approach outperforms the state-of-the-art for RL drone racing generalisation by a factor of 7.4, while only increasing laptimes by 15.8% on average. We demonstrate this both in simulation and by flying racetracks in the real world. We then extend this approach to pixel-based input, training an end-to-end controller that demonstrates, for the first time, vision-based ZSG for high-speed autonomous drone racing.

Nomenclature

Notation

a	action
c	action collective thrust
g	gate state
l	task switch evaluation window length
ℓ	clipped objective term over a given trajectory
q	task switching confidence score
r	reward
r_θ	PPO policy ratio
s	state
t	time
v	drone velocity
L^{CLIP}	PPO surrogate objective
N	number of tasks
N_g	number of gates
N_τ	number of environment timesteps per task
R	total reward
T_{test}	testing task set
T_{train}	training task set
$\hat{A}$	PPO advantage estimate
$\mathcal{A}$	action space
$\mathcal{B}$	concurrently sampled tasks
$\mathcal{M}$	drone Markov Decision Process
$\mathcal{O}$	observation space
$\mathcal{P}$	transition function
$\mathcal{R}$	reward function
$\mathcal{S}$	state space
$\mathcal{T}$	task space
Π	policy space
δp	vectors to four corners of gate
ϵ	PPO tuning parameter

γ	discount factor
λ	reward function weights
π	policy
ϕ	observation function
ρ	Spearman's rank correlation coefficient
τ	task parameters
θ	policy parameters
ω	drone angular velocity
ω^{ref}	action body rates
$\tilde{R}$	drone rotation

Acronyms and Abbreviations

RL	Reinforcement Learning
ZSG	Zero-Shot Generalisation
MDP	Markov Decision Process
UPOMDP	Underspecified Partially Observable Markov Decision Process
PPO	Proximal Policy Optimisation
MLP	Multi-Layer Perceptron
CNN	Convolutional Neural Network
SB	State-Based
VB	Vision-Based

Chapter 1

Introduction

Figure 1.1: Deployment of a single generalist policy on two different racetracks. Top: Twist (using simulated gates). Bottom: Figure8 (using real gates).

1.1 Introduction

Drone racing has recently emerged as a benchmark for agile robotics, pushing autonomous flight systems to their limits. Reinforcement Learning (RL) in particular has shown the ability to achieve human-level and even superhuman

performance when racing on fixed racetracks [16]. Compared to traditional model-based control approaches, RL can directly learn high-performance flight strategies through interaction with a simulated environment, achieving the rapid decision-making and precise manoeuvring required for high-speed drone racing.

However, existing RL-based racing agents exhibit a critical limitation: they fail to generalise. Controllers that excel on the racetracks they were trained on often crash immediately when deployed to new racetracks, typically failing to complete even a single lap. This behaviour stands in stark contrast to human pilots, both experts and non-experts alike, who can usually complete an unseen racetrack without crashing, even on their first attempt. The ability to fly safely and effectively without track-specific training or fine-tuning, which we refer to as zero-shot generalisation (ZSG), is therefore a fundamental capability for real-world deployment.

Prior work has taken steps toward improving ZSG for autonomous drone racing. Wang et al. [36] demonstrated an RL control pipeline that adaptively adjusts difficulty during training by iteratively updating the layout of the training racetracks. While this approach improved capability on unseen racetracks, it suffered from a major drawback: the learned agents could not fly competitively fast, trading away performance for generalisation. Beyond drone racing, several works in RL have explored training regimes that promote generalisation [2, 41], but these are typically limited to simple tasks, low-dimensional state spaces, or purely simulated environments [18]. Efforts to improve real-world RL generalisation in robotics either require online adaptation over multiple rollouts (e.g. Rapid Motor Adaptation in [20]), or address only narrow, low-dimensional task variations (e.g. terrain parameter particle filter in [21]).

In this work, we take a more systematic look at the ZSG problem in RL for drone racing. We propose a training method that combines large-scale parallelisation with adaptive task switching and informed task generation, enabling agents to continually encounter novel challenges while avoiding overfitting. Our approach yields strong generalisation across a wide range of unseen racetracks without sacrificing high-speed performance, achieving ZSG that is on par with state-of-the-art methods, despite requiring no fine-tuning on new tracks. We train a generalist policy with this method and deploy it to a real racing drone to confirm its efficacy (Figure 1.1).

To test the robustness of this recipe, we further validate it on a harder and more realistic setting: vision-based end-to-end control for drone racing. Here, the agent must learn directly from raw pixel inputs and minimal state information, a setting where existing approaches fail to generalise at all. Our method achieves successful zero-shot deployment on previously unseen tracks, demonstrating for the first time that RL-based vision policies can generalise for agile, high-speed drone racing.

Taken together, these results show that RL agents can achieve strong generalisation in high-speed drone racing without performance trade-offs, narrowing the gap between autonomous and human pilots. More broadly, they highlight a path toward robust, real-world deployment of agile autonomous drones.

1.2 Related Work

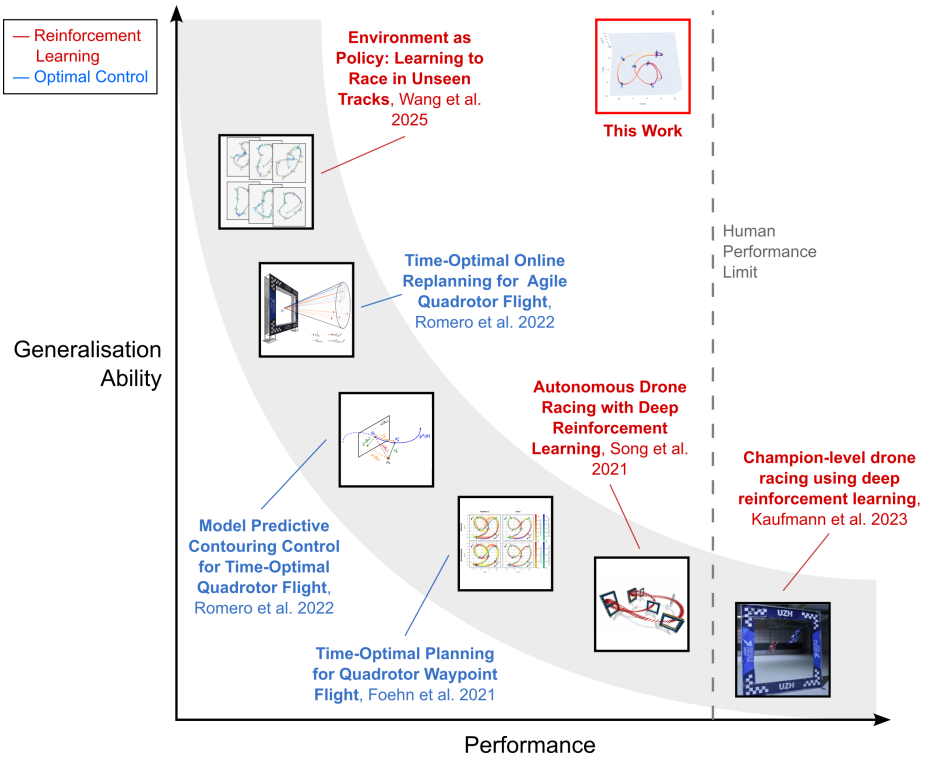


Figure 1.2: Comparison of performance vs generalisation trade-off with existing literature highlighting the key contribution of this work: best-yet generalisation without significantly reduced performance

RL has been widely adopted across a range of robotic tasks with a high level of success. Examples in which RL has been particularly effective for real-world robotics include legged locomotion [21], complex manipulation tasks [24], outperforming humans in video games [38], and interacting with complex human-centric environments [35]. However, these works involve controllers that are typically trained for and deployed on a single specialist task.

Recent advances in RL have extensively explored multi-task learning as a means to overcome overfitting to a single problem. The Paired Open-Ended Trailblazer (POET) framework [37] tackles this by employing evolutionary strategies to co-evolve a population of agents and tasks. Although effective at promoting diversity of ability, this approach requires maintaining separate policies for each task, limiting its scalability to continuous task distributions. A contrasting strategy involves environment shaping, where a single agent is trained in a dynamically evolving environment designed to maintain a strong learning signal [3]. However, such sequential curricula often suffer from catastrophic forgetting, as previously mastered tasks are replaced by new ones. Methods such as those proposed in [25, 15] address this limitation by reintroducing earlier environments to preserve prior knowledge. Despite these advances, the scalability of such methods to high-dimensional control tasks and their applicability to real-world

robotics remain largely untested.

To measure generalisation performance, several RL benchmarks have been established. These include frameworks with predefined task suites [40], randomly generated tasks [2], and procedurally generated environments [1]. Together, these benchmarks provide increasingly comprehensive assessments of how well agents transfer learned behaviours beyond their training distribution.

Within the domain of autonomous drone racing, a diverse body of work has emerged, spanning both optimal control and RL-based methods. Superhuman performance was first demonstrated in [16] using RL, although the policy was trained and fine-tuned on specific tracks, limiting generalisation. Alternative approaches have relied on time-optimal trajectory tracking [9, 26], which achieve fast lap times but require computationally expensive trajectory generation (ranging from seconds to hours) making them unsuitable for online adaptation. To mitigate this cost, [6] trained a neural network to approximate and then track optimal trajectories directly, though the approach instead required a large dataset of precomputed trajectories to train on. An alternative approach is to track a trajectory generated online using simplifying model assumptions, such as in [29, 28]. While this offers more generalisation, they still require per-track hyperparameter tuning for best performance. Other works have demonstrated agile vision-based flight without explicit state estimation [11], albeit only on a single racetrack. More recently, [36] achieved generalisation to unseen tracks with an RL-based racing agent, although at the cost of reduced peak performance compared to track-specific policies. A comprehensive comparison between optimal control and RL approaches for autonomous drone racing is provided in [32].

Chapter 2

Method

2.1 Problem Formulation

2.1.1 Background

In the RL framework, at every timestep t an agent in state $s_t \in \mathcal{S} \subseteq \mathbb{R}^n$ takes an action $a_t \in \mathcal{A} \subseteq \mathbb{R}^m$, receives a reward $r_t \in \mathbb{R}$, and transitions to the next state $s_{t+1} \in \mathcal{S}$, where $\mathcal{S}$ and $\mathcal{A}$ are the state and action spaces respectively. We define the probabilistic transition function $\mathcal{P} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ which maps a given state and action to the next state. The reward is computed by the reward function $\mathcal{R} : \mathcal{S} \to \mathbb{R}$. An action is determined by a policy $\pi \in \Pi : \mathcal{S} \to \mathcal{A}$. The framework can then be modelled as a conventional Markov Decision Process (MDP), defined by the 5-tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$ with a discount factor $\gamma \in (0, 1)$. Note that in this work we restrict our problem to state-based rewards. We consider the discounted infinite timeline problem where we seek to maximise the total reward $R(\pi, \mathcal{M}) = \sum_{t=0}^{\infty} \gamma^t r_t$. Thus, the goal is to find the optimal policy π^* that satisfies

$$\pi^* = \underset{\pi \in \Pi}{\operatorname{argmax}} \, \mathbb{E}_{s_0 \sim p(s_0)} \left[R(\pi, \mathcal{M}) \right]$$

for some initial state distribution $p(s_0)$. We model the policy as parameterised by a set of parameters θ, and denote a specific policy with π_θ.

2.1.2 Multi-Task Reinforcement Learning

While this captures the base case for RL well, there are two issues that we address to adapt this to the context of ZSG for drone racing.

Firstly, we typically do not have perfect state estimation; rather, we can only make some observation of our environment that contains (potentially incomplete and potentially noisy) information about our state. We can account for this by introducing an observation space $\mathcal{O}$ and an observation function $\phi : \mathcal{S} \to \mathcal{O}$ that lets our agent generate an observation $o_t \in \mathcal{O}$ at time t.

Secondly, our environment is not fixed. Since we are considering a multi-task context, there are a set of parameters that define our environment that can vary. Namely, in the context of drone racing, the position and orientation of the gates that define the racetrack can change. To properly parameterise our model to account for this, we introduce the task space $\mathcal{T}$, where for a given task a fixed parameter vector $\tau \in \mathcal{T}$ is selected.

Following [25] we can then model the problem as an Underspecified Partially Observable MDP (UPOMDP), defined by the tuple $(\mathcal{T}, \mathcal{O}, \phi, \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$. We use $\mathcal{M}_\tau$ to denote the environment and MDP induced by selecting a specific τ. We assume the environment is static, meaning that τ does not change during the agent's interactions with the environment. For generality we consider a continuous task distribution, rather than a discrete set of tasks. That is, we assume that the tasks that our agent will encounter in the real world can be modelled as a distribution over the task parameters $p(\mathcal{T})$. Given this distribution, and our augmented UPOMDP formulation, our objective becomes

$$\pi^* = \underset{\pi \in \Pi}{\operatorname{argmax}} \, \mathbb{E}_{s_0 \sim p(s_0),\ \tau \sim p(\mathcal{T})} \left[R(\pi, \mathcal{M}_\tau) \right]$$

In other words, we want to find the single policy that performs best in expectation across the distribution of possible tasks, not just any single task. Since our goal is ZSG, we cannot modify or update the policy with information during deployment. In the context of drone racing, we define a task to be a racetrack, which itself is composed of a fixed number of gates. Each gate has a four dimensional state vector $g = [x, y, z, \varphi]^T$ where x, y, and z are the cartesian coordinates while φ is the yaw. Note that we only provide one degree of rotational freedom for each gate as this is how drone racetracks are setup in real-world competitions. A track is then defined by its task parameter vector, which is the concatenated gate vectors in the order of the race, $\tau = [g_0^T, ..., g_{N_g-1}^T]^T$ for N_g gates. Hence we define the task space to be the set of all possible gate positions and yaws within a bounded volume for a fixed number of gates, so $\mathcal{T} \subset \mathbb{R}^{4N_g}$.

2.1.3 Measuring Generalisation

Generalisation in supervised learning is typically assessed by performance on a held-out test set. In RL however, there is no consensus on how to measure generalisation, as noted in [41, 18]. Different works employ different criteria, and the choice of metric often depends on the task. In this work, we restrict attention to zero-shot generalisation within the same task distribution, i.e. evaluation tasks are sampled from the same distribution as training tasks but are not encountered during training. Formally, let $T_{\text{train}} \subset \mathcal{T}$ denote the training set and $T_{\text{test}} \subset \mathcal{T}$ the disjoint evaluation set with $T_{\text{train}} \cap T_{\text{test}} = \emptyset$.

Several metrics have been proposed:

- Optimality gap: the difference between the agent's achieved return and the maximum achievable return, such as that used in [25]. While theoretically appealing, this requires computing or approximating optimal trajectories,

which is computationally infeasible for drone racing with a large number of racetracks.

- Generalisaton gap: the difference in average return between training and test tasks, as used in [1]:

$$\text{GenGap}(\pi) = \frac{1}{|T_{\text{train}}|} \sum_{\tau \in T_{\text{train}}} R(\pi, \tau) - \frac{1}{|T_{\text{test}}|} \sum_{\tau \in T_{\text{test}}} R(\pi, \tau).$$

  This metric is simple but problematic: it assumes returns are comparable across tasks. In drone racing, variations in track length and gate count can yield vastly different reward scales, making raw reward differences uninformative. Moreover, both highly capable and entirely random agents can exhibit small gaps by accruing similar rewards on test and train tasks.

- Success rate: the fraction of tasks successfully completed, such as that used in [2, 40]. This is intuitive for binary-completion tasks such as the hierarchical tasks in [4], but insufficient in drone racing where quantitative performance (lap time) matters as much as completion.

- Average reward: the average of the reward accumulated across all test tasks, such as that used in [5]. This does not account for success; it may be possible to have a high reward policy that cannot fly a single lap of the racetrack.

Given these limitations, we introduce Performance-Weighted Success Score, S_{pw}:

$$S_{pw}(\pi, T_{\text{test}}) = \frac{1}{|T_{\text{test}}|} \sum_{\tau \in T_{\text{test}}} \text{success}(\pi, \tau) \cdot \text{score}(\pi, \tau).$$

Here, success is a binary indicator of task completion, and score is a positive scalar reflecting quality of completion. This formulation is task-agnostic; success and score can be defined according to the domain.

For drone racing, we set

$$\text{success}(\pi, \tau) = \begin{cases} 1 & \text{if at least one lap is completed without crashing,} \\ 0 & \text{otherwise,} \end{cases}$$

and

$$\text{score}(\pi, \tau) = \begin{cases} 1/\text{fastest lap time}(\pi, \tau) & \text{if successful,} \\ 0 & \text{otherwise.} \end{cases}$$

This captures both feasibility (can the agent finish a lap?) and performance (how quickly?). While S_{pw} is not an absolute measure across different test sets, it provides a consistent and interpretable basis for comparing the ZSG ability of agents on the same T_{test}.

2.2 Approach

2.2.1 Training Pipeline

We adapt Proximal Policy Optimisation (PPO) to our parallelised multi-task paradigm to train a generalist agent. PPO is an on-policy actor-critic algorithm that optimizes a clipped surrogate objective to constrain the Kullback-Leibler divergence between successive policies [30]. This stabilises training by preventing destructive policy updates while maintaining sample efficiency and scalability. PPO has been widely adopted because it is both robust and effective [14].

While PPO is already well suited to a single-task parallelised training environment, we introduce additional logic to allow for multi-task training and task-switching. In contrast to conventional PPO, each of our environments contains a distinct task (in this case, each environment contains a different racetrack). Training on multiple tasks improves ZSG and mitigates catastrophic forgetting, shown formally in Section 2.2.2 and demonstrated empirically in 3.4.1.

Algorithm 1 Parallelised Proximal Policy Optimisation with Task Switching

Require: Task generator $\mathcal{G}$, task switching function Switch : $R_i \mapsto [0, 1]$, N environments, initial policy parameters θ, initial value function parameters ϕ, and learning rate η

Ensure: Optimised policy parameters θ^*

Initialise task parameters $\{\tau_i\}_{i=1}^{N} \sim \mathcal{G}$
Initialise environments $\mathcal{E}_i(\tau_i)$ for $i = 1 \dots N$
for each iteration **do**
    Collect trajectories λ_i from π_θ in all $\mathcal{E}_i(\tau_i)$
    for $i = 1 \dots N$ **do**
        $p_i \leftarrow \text{Switch}(R_i)$
        With probability p_i: $\tau_i \sim \mathcal{G}$, $\mathcal{E}_i \leftarrow \mathcal{E}_i(\tau_i)$
    end for
    Estimate advantages $\hat{A}_t$ and returns $\hat{R}_t$ from $\{\lambda_i\}$
    Form total PPO loss $L(\theta, \phi)$ with $\hat{A}_t$ and $\hat{R}_t$
    Update $(\theta, \phi) \leftarrow (\theta, \phi) - \eta \nabla_{\theta,\phi} L(\theta, \phi)$
end for

A "task controller" is used to maintain the set of current training tasks. At every policy update iteration, the task controller uses the vectorised rollout reward to compute the switching probability $p(\text{switch})$ for each task using the method described in 2.2.4. With this likelihood, the controller then updates the environment with a new task generated by the track generator described in 2.2.3. We describe the this process in Algorithm 1, where we use the spline-based track generator from Section 2.2.3 as our Task generator and the adaptive task switching logic from 2.2.4 as our Switch functions.

Additionally, we use domain randomisation to assist with reducing the sim-to-real gap during deployment. At the start of each episode we randomise the drone's dynamics and initial conditions. For details see Appendix A.3.

In practice, we implement this with the Flightmare simulator [31] to create and maintain our parallelised training environment, with Agilicious being used for

accurate drone dynamics modelling [8]. We train using a NVIDIA GeForce RTX 2080 Ti, with which we are able to achieve approximately 6.8×10^4 and 9.0×10^3 environment timesteps per second for the state- and vision-based agents respectively. Hyperparameter values used for training can be found in Appendix A.3.

2.2.2 Parallel Multi-Task Training

To generalise across a distribution of tasks, the agent must learn transferable skills rather than overfit to a single environment. In drone racing, this corresponds to training on multiple racetracks such that the policy can handle arbitrarily complex and realistic gate layouts. Prior approaches for multi-task RL such as [25, 15] are based on sequential curricula. These are, by their nature, susceptible to catastrophic forgetting, in which performance on earlier tasks degrades when new ones are introduced [10]. We demonstrate empirically that this is a particularly acute issue for drone racing in 3.4.1. To combat catastrophic forgetting we adopt a parallelised training paradigm in which rollouts are collected across multiple racetracks simultaneously and used jointly for policy updates.

Formally, the PPO surrogate objective for a single task is

$$L^{\text{CLIP}}(\theta) = \mathbb{E}_{(s,a)\sim\pi_{\theta_{\text{old}}}}\left[\min\Big(r_\theta(s,a)\hat{A}(s,a),\ \text{clip}(r_\theta(s,a), 1-\epsilon, 1+\epsilon)\hat{A}(s,a)\Big)\right],$$

with probability ratio $r_\theta(s,a) = \frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)}$, advantage function estimate $\hat{A}(s,a)$, and tunable hyperparameter ϵ. In practice, this expectation is approximated by the empirical average over a rollout with N samples:

$$\hat{L}^{\text{CLIP}}(\theta) = \frac{1}{N}\sum_{i=1}^{N} \ell(s_i, a_i; \theta),$$

where $\ell(s_i, a_i; \theta)$ denotes the clipped objective term. Mini-batch sampling provides an unbiased estimate of the empirical gradient of the PPO objective. In practice we use Adam [17], which adaptively rescales these stochastic gradient estimates, but the underlying mini-batch gradients remain unbiased estimates of the empirical average.

Extending to the distribution of tasks $\mathcal{T}$, the objective becomes

$$L_{\mathcal{T}}^{\text{CLIP}}(\theta) = \mathbb{E}_{\tau\sim\mathcal{T}}\Big[\mathbb{E}_{(s,a)\sim\pi_{\theta_{\text{old}}},\tau}\big[\ell(s,a;\theta)\big]\Big].$$

In our parallelised setting, this is estimated by an empirical average across both tasks and rollouts:

$$\hat{L}_{\mathcal{T}}^{\text{CLIP}}(\theta) = \frac{1}{|\mathcal{B}|}\sum_{\tau\in\mathcal{B}}\frac{1}{N_\tau}\sum_{i=1}^{N_\tau}\ell(s_i^\tau, a_i^\tau; \theta),$$

where $\mathcal{B} \subseteq T_{\text{train}}$ is the set of concurrently sampled tasks and N_τ the number of transitions per task.

This formulation ensures that gradients reflect contributions from multiple tasks simultaneously. While this does not eliminate forgetting in principle, it mitigates it by preventing the policy from being updated exclusively with data from a single task. Furthermore, because each update is based on a larger and more diverse batch of samples, the variance of the gradient estimator is reduced, leading to more stable optimisation.

2.2.3 Generating New Tasks

Algorithm 2 Informed Racetrack Generation

Require: Maximum attempts M, number of control points N_c, number of gates N_g, world bounds B, constraints C, bounded control point distribution $\mathcal{D}$
Ensure: Valid track or failure

```
attempt ← 0
while attempt < M do
    p_i ~ D(B, C)   ∀i ∈ {1, ..., N_c}
    P ← {p_1, ..., p_{N_c}}
    S ← GenerateSpline(P)
    (G, Q) ← PlaceGates(S, N_g, C)
    valid ← (G, Q satisfy all constraints in C)
    if valid then
        return BuildTrack(G, Q, B)
    end if
    attempt ← attempt + 1
end while
return Failure("no valid track found")
```

A straightforward method for producing new training tasks is uniform random sampling from the task space. However, this has several drawbacks. First, uniform sampling does not respect the underlying structure of $\mathcal{T}$: feasible drone racing tracks are not distributed uniformly across the high-dimensional task space. Second, as dimensionality grows, maintaining sufficient sample density requires exponentially more samples, causing sparsity of exploration. Finally, many randomly sampled tracks may be either infeasible (violating physical constraints) or uninformative (providing little learning signal).

We therefore adopt an informed task generation approach that restricts sampling to a subspace of $\mathcal{T}$ defined by feasibility and diversity criteria. Specifically, tasks must (i) admit at least one smooth trajectory through all gates, (ii) satisfy physical and geometric constraints, and (iii) vary sufficiently in geometric properties to expose the agent to a wide range of flight dynamics. This process is formally defined and constraints elucidated in Appendix A.1.

Our method is spline-based. A set of control points $P = p_1, \ldots, p_{N_c}$ are sampled from a bounded distribution $\mathcal{D}(B, C)$. A B-spline S is fitted to these points, and gates are placed tangent to S at arc-length-spaced intervals. The spline defines a continuous reference trajectory through all gates. While this trajectory is not generally time-optimal, its smoothness guarantees at least one geometrically feasible flight path. Dynamic feasibility is approximated by enforcing curvature,

and spacing constraints C during spline construction and gate placement.

The algorithm is given in Algorithm 2, with more detail provided in Appendix A.1. By tuning spline parameters and control point distributions, we can generate novel racetracks that well approximate those seen in the real-world, thereby aligning procedurally generated tasks with deployment conditions.

2.2.4 Task Switching

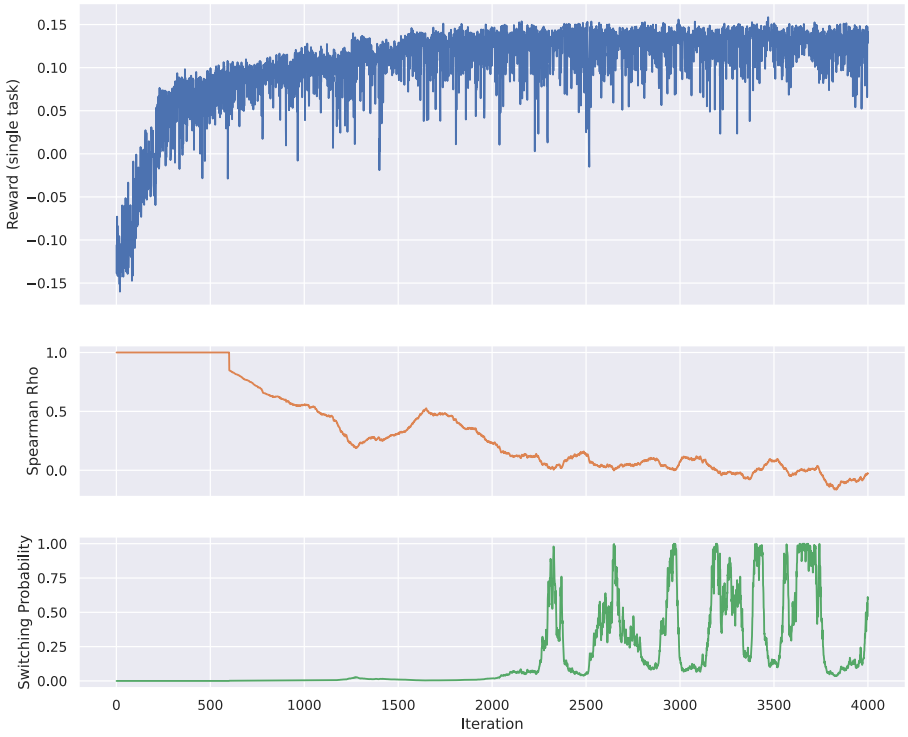


Figure 2.1: An example of how the switching score changes during training for a single task based on the rollout reward for that task. For this example $l = 600$ iterations and $\alpha = 1000$.

Assessing when a task should be replaced requires measuring whether it continues to generate useful learning signal. Using raw reward thresholds such as in [37] is unsuitable in our setting, since reward scales differ between tracks. Given the computational difficulty associated with computing time-optimal trajectories it is infeasible to normalise these reward scales a priori. Furthermore, task difficulty is non-stationary: as the policy improves, tasks within the training set transition from challenging to trivial.

Instead of absolute performance, we focus on performance improvement. If the expected return for a task ceases to change across recent updates, then that task is no longer providing gradients that yield meaningful policy updates. Since rollouts from multiple tasks are collected in parallel, global reward trends may obscure local convergence; task-specific analysis is required.

We model this as an online flatness detection problem. Let R_i^τ denote the cumulative rollout reward on task τ at policy iteration i. For each task, we consider a sliding window of length l which contains the last l cumulative rollout rewards. To measure trend, we compute Spearman's rank correlation coefficient ρ [13] between these cumulative reward values and their corresponding timesteps. Because the timesteps are monotonically increasing, this tells us:

- $\rho > 0$: upward trend (task still contributing improvements)
- $\rho < 0$: downward trend (performance degrading)
- $\rho \approx 0$: flat (little to no learning signal)

We found this to be more robust to noise than other flatness detection methods (alternatives outlined in Appendix A.2) because it does not rely on any explicit curve fitting or estimation but rather measures the overall trend present in the data.

To map this statistic into a switching score, we define a smooth heuristic function:

$$q([R_{i-l}^\tau, \ldots, R_i^\tau]) = \frac{1}{1 + \alpha\rho^2}$$

where $\alpha > 0$ controls sensitivity. This score $q \in (0, 1]$ is not a calibrated probability, but a monotonic confidence-like measure where higher values correspond to greater certainty that the reward curve is flat. Both positive and negative slopes reduce the score, since either indicates deviation from stagnation.

At each training iteration we compute the switching score for each task and it as the likelihood of switching to a new one. This mechanism ensures that training time is allocated to tasks whose returns are actively changing, rather than to those that have saturated. This process is visualised for a single task in Figure 2.1. Empirically we found $l = 600$ and $\alpha = 1000$ to be effective values, although varying these parameters had little impact on performance as long as l was sufficiently large.

We note that while p-values associated with ρ could also be used, they are unsuitable for online detection: a large p-value only indicates lack of evidence against the null hypothesis $\rho = 0$, not evidence for flatness itself. Our heuristic is therefore more appropriate for continual monitoring during training.

2.2.5 Model

Architecture

The controller follows an Actor-Critic design trained with the modified PPO algorithm described in Section 2.2.1. Both actor and critic networks comprise a Multi-Layer Perceptron (MLP) with two hidden layers of width 512 and LeakyReLU activations.

At each timestep, the environment provides an observation, and the agent outputs an action $a = [c, \omega^{\text{ref}}]$, where c is the collective thrust and ω^{ref} the body-rate setpoints for roll, pitch, and yaw.

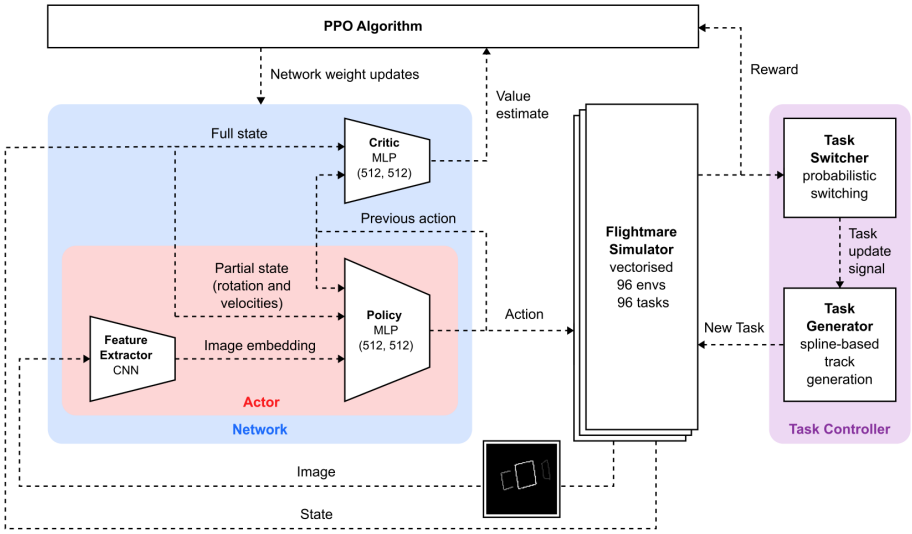


Figure 2.2: Diagram of model architecture for the vision-based generalist agent during training. Note that the state-based agent lacks the feature extractor, and instead the policy receives only the full state and previous action as its input.

Observations consist of two modalities: state and vision. The state observation is defined as

$$o^{\text{state}} = \left[\tilde{R}, v, \omega, a_{\text{prev}}, \delta p_1, \delta p_2\right],$$

where $\tilde{R} \in \mathbb{R}^6$ encodes the first two columns of the rotation matrix R_{WB} [42], $v \in \mathbb{R}^3$ and $\omega \in \mathbb{R}^3$ are the linear and angular velocities, and a_{prev} is the previous action. Gate-related information is provided by $\delta p_1, \delta p_2 \in \mathbb{R}^{12}$, which encode the relative positions of the drone to the four corners of the upcoming gate and from the upcoming gate to the subsequent gate, respectively.

The vision observation o^{image} is an 84×84 image processed by a Convolutional Neural Network (CNN) encoder into a 128-dimensional embedding o^{embed}. We use a CNN with three convolutional layers with channel sizes [32, 64, 64], kernel sizes [8, 4, 3], and strides [4, 2, 1].

While previous vision-based drone racing agents such as [11] have used black and white images, we instead use 8-bit greyscale. This allows us to encode gate ordering, which provides essential information for a generalist agent; without memorising the racetrack layout, the agent must be able to determine which is the next gate. Specifically, we make the next gate full brightness by setting the corresponding pixels to an intensity of 255, the following gate medium brightness with an intensity of 155, and all other gates low brightness with an intensity of 55 (Figure 2.3).

For state-based training, both actor and critic receive o^{state}. For vision-based training, the critic again uses o^{state}, while the actor receives $[o^{\text{embed}}, \tilde{R}, v, \omega]$. Inclusion of $\tilde{R}$ ensures rotational stability in cases of partial gate visibility, and v, ω provide explicit velocity information otherwise unobtainable from the

embedding alone. This is necessary because the actor's network (an MLP) lacks the recurrency necessary to compare embeddings across multiple timesteps.

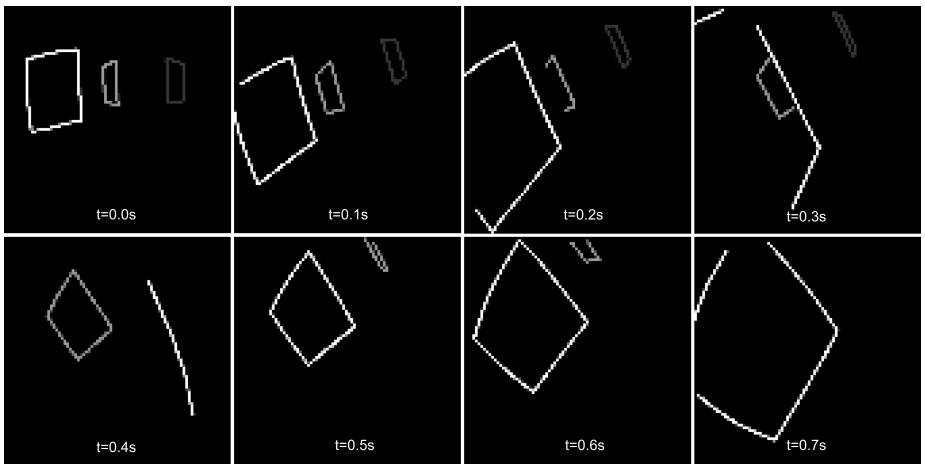


Figure 2.3: Image component of observation o^{image} for vision-based agent, showing gate order indicated by brightness. Note that between time t=0.4s and time t=0.5s, the drone crosses the upcoming gate and so the following gate switches to being the brightest.

Reward

At each timestep t, the reward is a weighted sum of components,

$$r_t = r_t^{\text{prog}} + r_t^{\text{pass}} + r_t^{\text{crash}} + r_t^{\text{rate}} + r_t^{\text{cmd}}$$

with coefficients λ listed in Table 2.1. Separate weightings are used for state-based (SB) and vision-based (VB) controllers.

The progress reward reduces sparsity by providing incremental feedback:

$$r_t^{\text{prog}} = \begin{cases} \lambda_{\text{prog}}(d_{t-1} - d_t), & \text{SB} \\ \lambda_{\text{prog}}\Big(\max(d_{t-1} - d_t, 0)\rho_t + \min(d_{t-1} - d_t, 0)\Big), & \text{VB}, \end{cases}$$

where d_t is the euclidean distance to the next gate centre and $\rho_t \in [0, 1]$, defined as the dot product between the normalised direction of the camera and the normalised normal to the gate plane, is a view-quality factor encouraging camera-gate alignment.

The gate-passing reward is triggered upon crossing the gate plane:

$$r_t^{\text{pass}} = \begin{cases} \lambda_{\text{pass}}\big(1 - e_t/w_g\big), & \text{SB} \\ \lambda_{\text{pass}} - \lambda_{\text{error}}\, e_t, & \text{VB}, \end{cases}$$

where e_t is the traversal error and w_g half the gate width.

Crashes incur a constant penalty $r_t^{\text{crash}} = \lambda_{\text{crash}}$.

To prevent aggressive manoeuvres detrimental to sim-to-real transfer, we penalise body-rate magnitudes:

$$r_t^{\text{rate}} = \lambda_{\text{rate}} \left\| \frac{\omega_{t,xy}}{\omega_{xy}^{\max}} \right\|^2.$$

Finally, to promote smooth and deployable control, the command penalty is defined as

$$r_t^{\text{cmd}} = \sum_{i \in \{\text{thrust}, xy, z\}} \lambda_{\text{diff},i} \Big(\lambda_{\text{linear}} |\Delta a_{t,i}| + (\Delta a_{t,i})^2 \Big),$$

where

$$\Delta a_t = \frac{a_t - \tilde{a}_t}{a^{\max}},$$

$a^{\max}$ is the maximum action magnitude, and $\tilde{a}_t$ is a first-order low-pass filtered action with a cutoff frequency of 6 Hz.

Term	SB	VB	Description
λ_{prog}	1.0	0.8	Progress towards next gate
λ_{pass}	1.0	4.0	Reward for successful gate traversal
λ_{error}	N/A	-1.0	Penalty for inaccurate traversal
λ_{crash}	-4.0	-4.0	Collision penalty
λ_{rate}	-0.001	-0.01	Penalty on large body rates
$\lambda_{\text{diff,thrust}}$	-0.0001	-0.05	Thrust smoothness penalty
$\lambda_{\text{diff},xy}$	-0.0002	-0.01	Roll/pitch smoothness penalty
$\lambda_{\text{diff},z}$	-0.0002	-0.01	Yaw smoothness penalty
λ_{linear}	0.01	0.01	Linear balancing factor

Table 2.1: Reward weights for state-based (SB) and vision-based (VB) controllers.

2.2.6 Deployment

We deploy the state-based model on real-world racetracks (Figure 1.1) with a custom quadrotor platform. A VICON motion capture system is used for precise state estimation. Inference on the control policy network is performed on a workstation and transmitted to the quadrotor, where the collective thrust and body rates are are converted to motor commands by a low level controller.

Chapter 3

Results

3.1 Generating Results

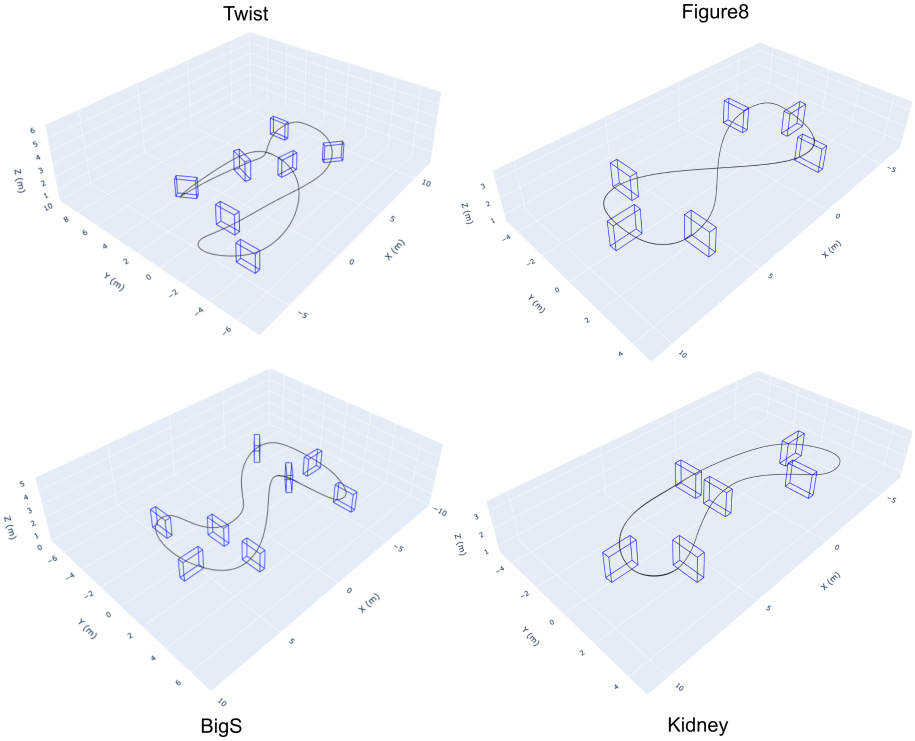


Figure 3.1: The four "core tracks" used to compare results

Measuring the ability of a generalised drone racing agent requires evaluating its performance on a large number of different racetracks, all of which are unseen (i.e. not present in the set of training tasks). Specifically, we use a test set of 40 racetracks: nine hand-designed and 31 generated with the spline-based method described in Section 2.2.3. Each agent is deployed 25 times per track for 1000

timesteps. The initial position is varied randomly within ±0.8 m in the x and y direction and ±0.6 m in the z direction to ensure robustness. Success is defined as a crash-free deployment in which at least one lap is completed, and lap time is reported as the mean over successful trials.

Furthermore, to provide more granular analysis and comparison between other approaches, we consider a set of four "core" racetracks (Figure 3.1). These racetracks (Figure8, BigS, Kidney, and Twist) are all human-designed and require a broad range of different manoeuvres to fly successfully. Thus they entail a representative set of tasks that can well capture the capability of an agent.

In the rest of this chapter we primarily refer to three types of agent: the "Generalist Agent", "Environment as Policy", and "Regular PPO". Environment as Policy refers to the method proposed in [36], and regular PPO is an agent trained on a single track with the original PPO algorithm [30]. The generalist agent is trained with the approach proposed in this work, i.e. the modified PPO described in Section 2.2.1, the adaptive task switching described in Section 2.2.4, the informed task generation described in 2.2.3, and a weight decay of 1.0×10^{-5} as outlined in 3.3.3. For consistency, the regular PPO agent is trained with the same observation as the generalist agent, as outlined in 2.2.5.

3.2 Performance

3.2.1 Speed

	Lap Time (s)		
Track	Generalist Agent	Environment as Policy	Regular PPO
Figure8	2.940	4.746	2.546
BigS	4.514	7.513	4.099
Kidney	3.202	4.943	2.340
Twist	4.746	10.20	4.102

Table 3.1: Average simulated lap times. Note that for Regular PPO a dedicated agent was trained for each track, while for the remaining agents a single policy was deployed across all tracks.

As shown in Table 3.1, the generalist agent achieves substantially lower lap times than the Environment as Policy baseline, with a mean improvement of 41.67% across the four core tracks. While it remains somewhat slower than training a dedicated PPO agent on each racetrack individually (on average 15.77% slower), these PPO agents do not generalise at all and exhibit poor robustness to minor track variations (see Section 3.4.3).

3.2.2 Generalisation

The generalist agent achieves markedly higher ZSG than prior approaches (Figure 3.2). Specifically, its maximum S_{pw} reaches 0.1652, compared to 0.0222 for Environment-as-Policy [36] and 3.250×10^{-4} for single-track PPO agents. This corresponds to a 7.4× improvement over the current state-of-the-art.

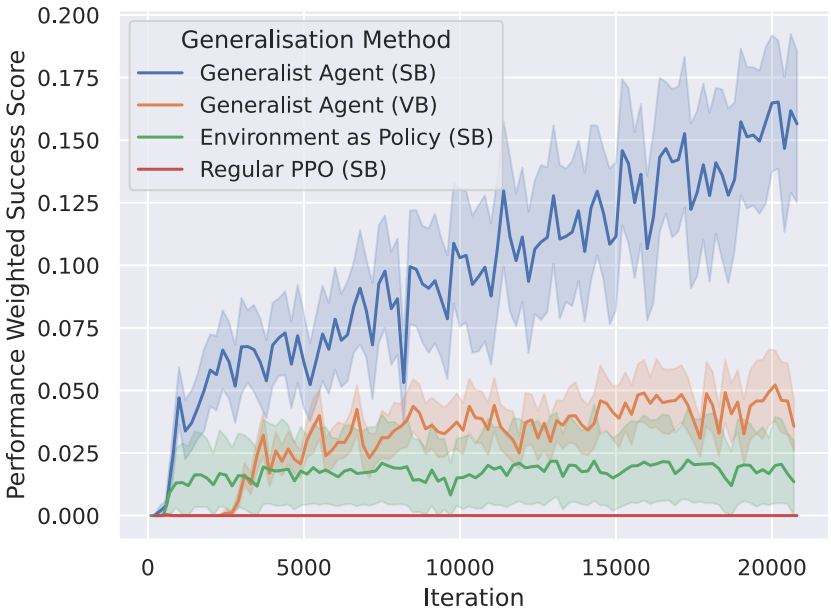


Figure 3.2: Comparison between our approach and current state-of-the-art, Environment as Policy [36]. SB denotes state-based and VB denotes vision-based. Note that a reduced maximum thrust of 6.0N was used for Environment as Policy to maintain stability during training, while all other runs where with a maximum thrust of 8.5N.

Additionally, when our approach is extended to a vision-based observation, as described in 2.2.5, the agent achieves a maximum S_{pw} of 0.05219, demonstrating generalisation 2.3× better than Environment as Policy, despite the significantly more challenging observation format.

To ensure stability during training we had to limit the maximum thrust for the Environment as Policy agent to 6.0N, compared to the generalist agent's 8.5N. We note that the S_{pw} is not being artificially limited by lower achievable lap times due to a lower thrust because the generalist agent also achieves a higher average success rate than Environment as policy at every policy iteration.

3.2.3 Vision

As described in Section 2.2.5, we extend our pipeline to a vision-based agent to demonstrate the capability of the generalist agent. As noted in [11], flying directly from pixels without explicit state estimation is a much closer paradigm to human pilots and enables a more scalable approach to deployment in the real world. Figure 3.2 shows that our vision-based agent demonstrates a high capability with a maximum S_{pw} of 0.052188. While this is worse than the state-based generalist agent, it still significantly outperforms the state-based Environment as Policy, and, to the best of our knowledge, marks the first demonstration of

ZSG for vision-based high-speed drone racing.

3.3 Ablations

In the following section we quantify the effect of the strategies used to improve generalisation by selectively disabling them and measuring how this impacts performance.

3.3.1 Adaptive Task Switching

Switching Method	Max S_{pw}	Time to 50% of max S_{pw} (iterations)
No Switching	0.1004	2600
Random Switching	0.09746	1400
Adaptive Switching	0.1119	1600

Table 3.2: Generalisation performance comparison between different options for task switching

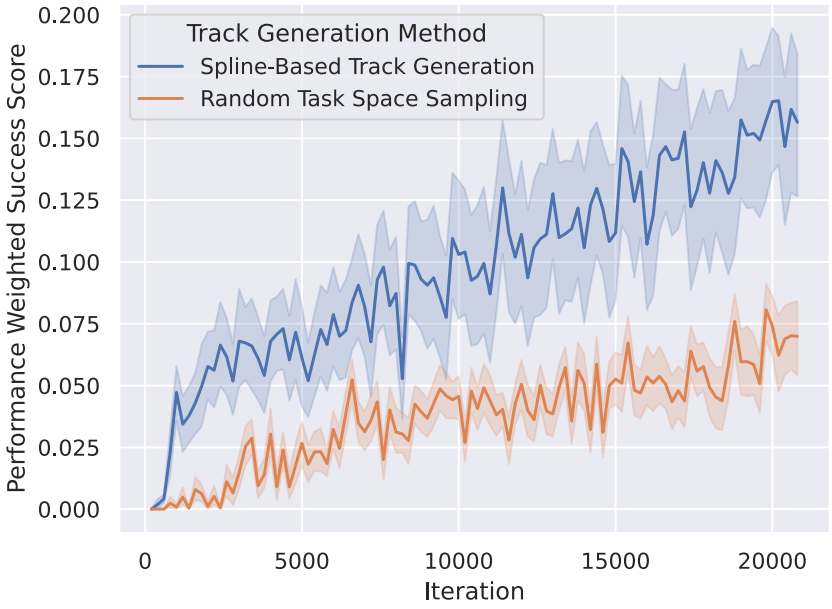


Figure 3.3: Comparison of the impact of track generation method on generalisation during training

We assess the impact of adaptive task switching by training one agent with no task switching at all, one with random task switching, and one with adaptive task switching. In all cases the agent is trained on 96 parallel environments, each with a different racetrack generated with the constrained spline method described in Section 2.2.3. When random task switching is used, at every iteration

there is a fixed probability of a task being switched out for a new one. A probability of 1.616×10^{-3} was used as this gave, in expectation, the same number of task changes as was observed when the adaptive task switching. This resulted in 3273 changes for the random switching and 3183 changes for the adaptive task switching, which, while not identical, was deemed close enough for a fair comparison. We disabled weight decay during this ablation to isolate only the impact of task switching.

Adaptive task switching enabled better generalisation, as measured with a higher maximum S_{pw}, while also providing a faster increase in generalisation during training (Table 3.2). Compared to no switching, random task switching causes a slight decrease in maximum generalisation of 2.963%. In contrast, adaptive task switching yielded an increase of 10.29%. Additionally, using track switching in general facilitates a quicker increase in generalisation during training, potentially increasing sample efficiency, with random and adaptive task switching achieving a 50% of their maximum S_{pw} in 53.85% and 61.54% of the time taken for the no switching approach to do the same.

3.3.2 Informed Task Generation

We assess the impact of informed task generation by training two agents utilising adaptive task switching, one with informed track generation and one with random sampling from the task space. Specifically, for informed task generation, the new racetracks are generated using the constrained spline method described in Section 2.2.3. Using informed task generation provides consistently better generalisation as measured by performance weighted success with a maximum score of $S_{pw} = 0.1652$, a factor of 2.05× better than the score of $S_{pw} = 0.08069$ achieved by random task space sampling (Figure 3.3).

3.3.3 L_2 Regularisation

L_2 regularisation (also referred to as weight decay) is a regularisation metric that prevents any single weight from deviating too far from zero by penalising the sum of the square of the weights. It is not commonly used in RL as it can decrease performance on the training dataset, however is widely used in other fields as a method for improving generalisation [23]. Some work suggests that it can also increase ZSG in RL [2]. We assess its impact on drone racing by measuring the generalist agent's S_{pw} after training with weight decays of 0, 1.0×10^{-6}, 1.0×10^{-5}, 1.0×10^{-4}, and 1.0×10^{-3} We found that it does significantly improves generalisation when well tuned, with the best value empirically demonstrated to be 1.0×10^{-5} However, values of 1.0×10^{-4} or larger can harm performance and even result in instability during training (Figure 3.4).

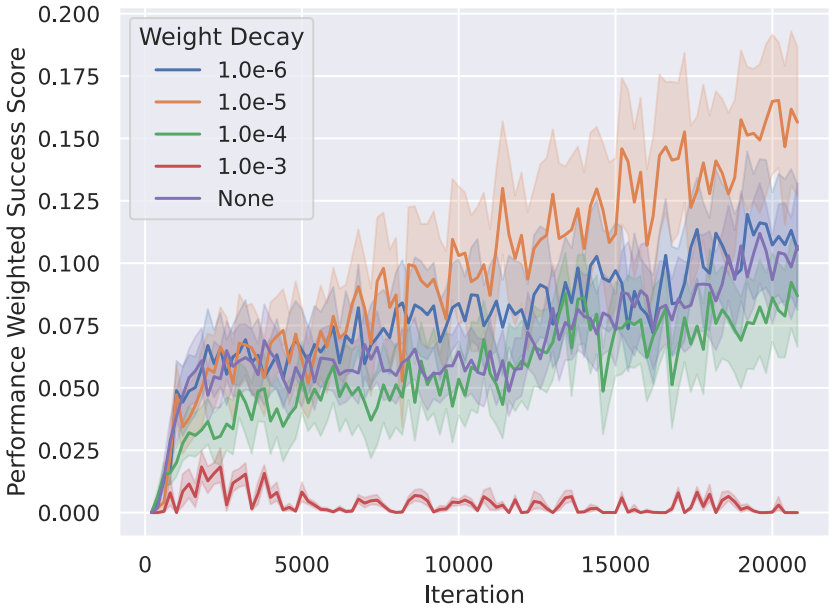


Figure 3.4: The impact of using different values for L_2 regularisation (weight decay) on generalisation during training, as measured by S_{pw}

3.4 Studies

In this section we perform an analysis of various aspects of RL controllers for drone racing, and explore how they impact ZSG.

3.4.1 Catastrophic Forgetting

A well documented issue with RL is catastrophic forgetting: when trained on new tasks, agents often lose the ability to solve tasks they previously mastered [34]. This problem is especially acute in sequential multi-task RL, as noted in [19]. To illustrate, we trained a racing agent on the four core tracks (in order: Figure8, BigS, Kidney, and Twist), each presented in isolation for 1.25×10^8 timesteps (5200 iterations). As shown in Figure 3.5, the agent learns to complete the active track with high reward and success, but as soon as training switches, success on earlier tracks collapses to zero and is never recovered. The agent is therefore only capable of racing on the most recent track, demonstrating the severity of forgetting and the instability of sequential training in this setting.

3.4.2 Sim-to-Real Gap

Across the four core racetracks, lap times in real-world deployment were 6.476% slower on average (Table 3.3). While this demonstrates a measurable sim-to-real gap, the magnitude remains modest, indicating that the learned policy

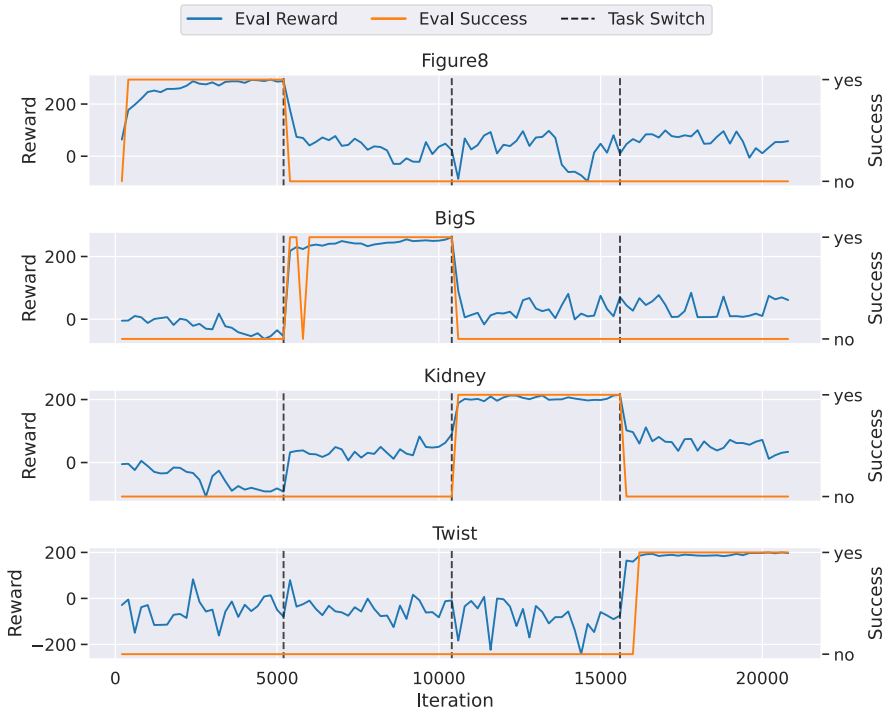


Figure 3.5: Ability to race on four different tracks during training. Results of evaluation on the tracks visualised with the training track switching every 1.25×10^8 timesteps (5200 iterations)

transfers effectively. Note that due to track limitations during deployment we use the "SplitS" track rather than Twist, which is a slightly modified form of Twist.

3.4.3 Fixed Trajectories

In contrast to trajectory-tracking controllers such as those in [29] and [9], which require a predefined reference trajectory, RL-based controllers learn policies directly from interaction without trajectory supervision. This removes the need for expensive trajectory optimisation. However, we hypothesise that in practice the learned policy degenerates into implicit trajectory tracking: the agent discovers a single high reward state-action sequence during training and subsequently overfits to it, rather than acquiring a generalisable control strategy.

We test this hypothesis with two experiments. In the first, an agent is trained on the Figure8 track and learns a trajectory that consistently clears all gates. At evaluation time we add an additional gate that lies on this trajectory, so that no deviation from the learned path is required. Despite being geometrically redundant, the presence of the new gate perturbs the agent's behaviour: the policy clears the inserted gate but then destabilises, leading to a crash within the following segment (Figure 3.6A). This shows that the learned policy is not

Track	Avg Lap Time, Simulated (s)	Avg Lap Time, Real-World (s)	Sim-to-Real Gap (% change in lap time)
Figure8	2.940	3.273	11.33
BigS	4.514	4.633	2.636
Kidney	3.202	3.511	9.650
SplitS	5.160	5.278	2.287

Table 3.3: Comparison between simulated and real-world performance of the generalist controller. Note that SplitS is a modified version of the Twist track.

conditioned on the geometry of the environment in general, but rather on the specific sequence of observations encountered during training.

In the second experiment, we use the same Figure8 training environment in which the initial manoeuvre required is a right-hand turn. We then evaluate the agent on the BigS track, where the first manoeuvre must instead be a left-hand turn. The agent nonetheless initiates a rightward turn, subsequently executing a repetitive loop around the starting gate and failing to progress (Figure 3.6B). This confirms that the policy has encoded the initial state-action mapping of the training trajectory, rather than learning a general strategy for negotiating unseen gate configurations.

These results suggest that policies trained with regular PPO on a single track, while not explicitly trajectory-conditioned, implicitly overfit to specific training trajectories. Without modifications for improved generalisation it appears that regular PPO for drone racing effectively reduces to trajectory memorisation.

3.4.4 Use As Foundation Model

Finally, we evaluate whether the generalist agent can serve as a foundation model for drone racing. Fine-tuning it on individual tracks reduced training requirements substantially when compared to the regular PPO approach: convergence was achieved in 34.96% of the iterations needed to train a PPO agent from scratch, while also yielding consistently lower lap times (Figure 3.7). The fine-tuning process involves an additional 1.0×10^8 timesteps of training with a reduced learning rate. This demonstrates that generalist pre-training provides an efficient starting point for task-specific adaptation.

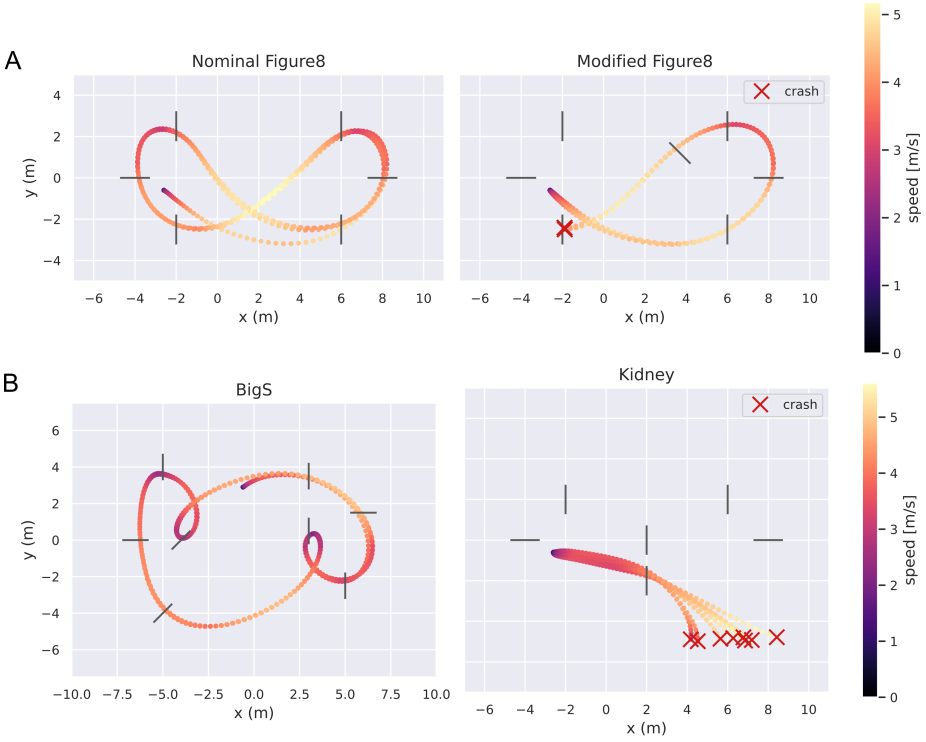


Figure 3.6: Demonstration of how a controller trained on a single track effectively memorises a single trajectory. In A, providing a small change to the observation by adding a gate along the existing trajectory induces a crash. In B, the agent turns right after the first gate on an unseen track and crashed because the first turn on the training track was a right-hand turns.

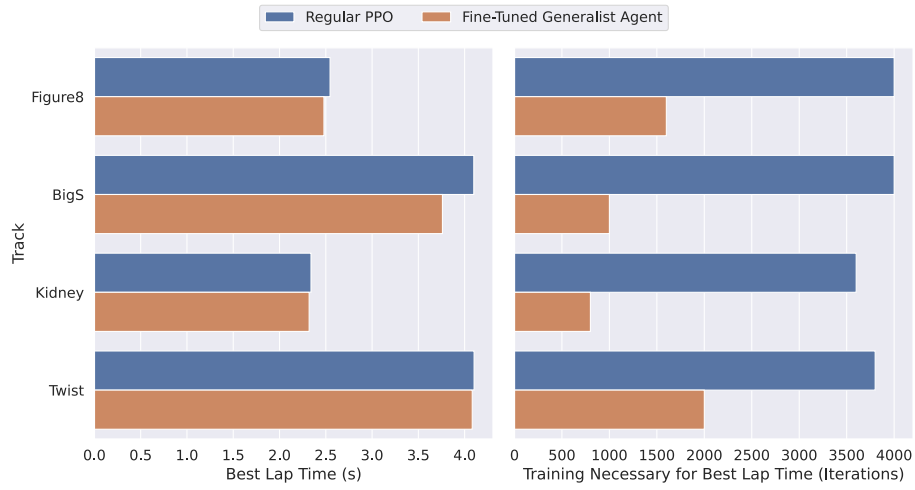


Figure 3.7: When used as a foundation model, the generalist agent can be fine tuned to equivalent or better performance than the regular PPO agent in fewer iterations.

Chapter 4

Discussion

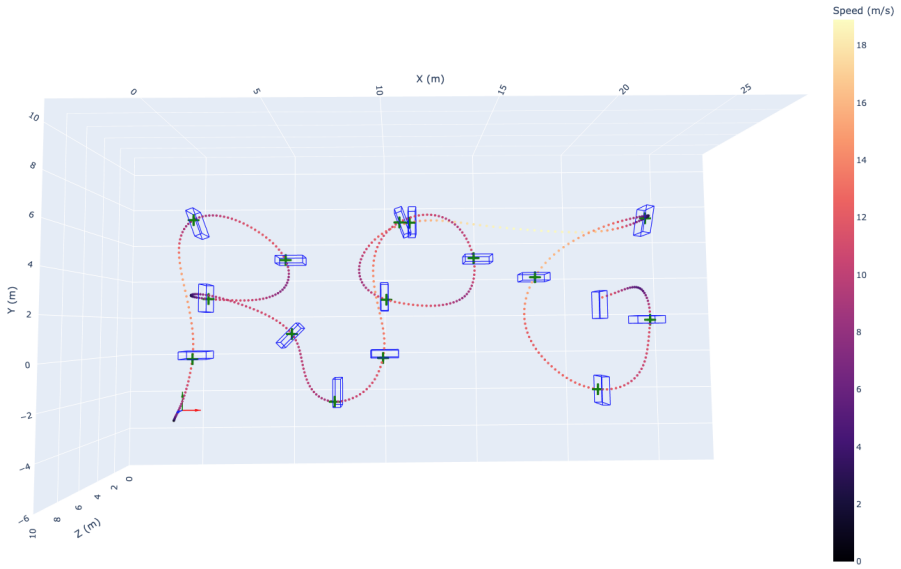


Figure 4.1: Our generalist agent can successfully race arbitrarily complex racetracks. This racetrack contains complex high-speed manoeuvres and was not present in the training set.

4.1 Key Questions

How effective is our approach? As visible in Secction 3.2, our approach sets a new standard for RL generalisation in autonomous drone racing by being capable of high-speed racing on a broad range of unseen tracks, including arbitrarily complex hand-designed ones (Figure 4.1). Additionally, it enables for the first time ZSG for a vision-based drone racing agent. Given the tendency of other learned models to overfit and the trajectory precomputation requirements

for optimal control-based methods, it is clearly our novel approach that enables this.

Our results in Section 3.2 show that the proposed model and training pipeline substantially advances ZSG for autonomous drone racing. Quantitatively, both our state- and vision-based generalist agents significantly outperform the Environment as Policy baseline, with an improvement of 7.4× and 2.3× respectively (Figure 3.2). These gains are accompanied by modest costs: the generalist agent is on average 15.8% slower than per-track regular PPO agents but is substantially faster than the Environment as Policy baseline with 41.67% lower lap times (Table 3.1). In hardware deployment the sim-to-real gap is small, with a mean lap-time increase of only 6.476% (Table 3.3), indicating that the policies we learn are not only better at ZSG in simulation but also transfer credibly to the real world. Taken together, these observations indicate that the method achieves a favourable mix of generalisation and speed: it significantly improves ZSG while maintaining competitive performance.

What are the key contributions that enable improved ZSG? Our ablations in Sections 3.3 and studies in Section 3.4 identify several interacting mechanisms:

- *Diverse, parallelised multi-task training.* Collecting rollouts from a large and diverse batch of tasks at every update reduces the tendency of the policy to overfit to a single trajectory and mitigates catastrophic forgetting relative to sequential curricula (Sec. 3.4.1).
- *Informed task generation.* Constraining the procedural generator to produce geometrically feasible and diverse tracks with a constrained spline-based sampling method yields a 2.05× improvement over uniform sampling (Figure 3.3), suggesting that task quality matters as much as quantity.
- *L_2 regularisation (weight decay).* A properly tuned weight decay (1.0×10^{-5} in our experiments) improves ZSG; larger values destabilise learning (Fig. 3.3), highlighting the sensitivity of regularisation in on-policy RL.
- *Adaptive task switching.* The flatness-detection switching rule based on Spearman rank correlation yields a reliable heuristic for retiring saturated tasks and provides a modest but measurable gain in ultimate S_{pw} and in the sample efficiency for generalisation (Table 3.2).
- *Architectural and observation design.* For vision-based control, the combination of a compact image embedding, a small set of state channels (rotation columns, velocity, angular rates) and the gate-brightness encoding (to indicate gate order) enabled stable, generalising policies where prior vision-based systems could only race a single track.

These components are complementary: diversity and informed generation expose the agent to representative geometry; regularisation and parallel updates reduce overfitting; adaptive switching focuses training capacity where it is still useful; and careful observation design ensures stable control from pixels.

What does this tell us about the trade-off between performance and generalisation? Despite being a widely acknowledged problem, poor ZSG in

RL is not well understood [18]. That being said, the empirical evidence in this work argues against a strict, binary trade-off between performance and generalisation. Our generalist policies achieve markedly better ZSG than prior methods with only modest degradation relative to highly overfit, single-track trained PPO agents. This suggests that - at least in the drone-racing domain - architectural choices, data diversity, and appropriately tuned hyper-parameters can shift the pareto-frontier: one can recover much of the performance of specialised controllers while gaining large improvements in generalisation. Moreover, the strong speedups observed when fine-tuning the generalist agent (convergence in 34.96% of the iterations needed from scratch on average) imply that generalist pre-training does not necessarily impose an irreducible cost: it can serve as an efficient starting point for specialised, high-performance policies.

How does this work extend to the wider field of RL? There are three broad takeaways. First, the combination of large-batch, parallel multi-task updates and informed procedural task generation scales to a high-dimensional task space and yields measurable ZSG gains even in the real world; this framework is applicable beyond racing and in robotic control in general where task geometry is structured but high-dimensional. Second, the switching mechanism provides a lightweight, online heuristic for curriculum management that does not require hand-designed difficulty metrics or expensive optimal-trajectory computations. Third, the success of a vision-enabled generalist highlights that it is possible to extend these improvements in ZSG to end-to-end policies for real robotic systems.

What are the limitations and failure modes? Despite the positive results, several important limitations remain:

- **In-distribution evaluation.** Our test set is sampled from the same generator family as training. Extrapolation to out-of-distribution track families (substantially different gate formats, dynamic gates, or novel obstacle types) is not yet characterised and must be explicitly evaluated.
- **Residual reliance on state cues for vision policies.** The vision-based agent currently requires a small amount of state information (absolute rotation, linear velocity, and angular velocity) to ensure stability; removing these cues exposes limitations of the current architecture as the training fails to converge. This indicates the need for some form of memory, either through frame stacking or recurrency.
- **Implicit trajectory memorisation.** The fixed-trajectory experiments (Section 3.4.3) show that even non-trajectory-conditioned RL can overfit to sequences of states encountered during training. While our generalist approach reduces this tendency, the risk of brittle, memorised behaviour in corner cases remains.
- **Robustness under environmental perturbations.** We have not tested real-world disturbances (such as wind gusts, sensor dropout); robustness to such perturbations should be probed more systematically.
- **Compute and sample efficiency.** Training with large numbers of parallel environments and many task changes is computationally demanding;

improving sample efficiency remains an important practical objective.

What are concrete paths to further improve ZSG? Based on the above, we recommend several complementary directions:

1. **Fast adaptation / meta-learning.** Adding a light-weight online adaptation mechanism (such as meta-learning [7] or fast context adaptation [43]) could close the remaining performance gap with single-track specialists while preserving ZSG for initial deployments.
2. **Continual-learning regularisers.** Methods such as Elastic Weight Consolidation [19] and gradient-projection techniques [39, 22] could be integrated to further ameliorate catastrophic forgetting in sequential or partially overlapping sets of training tasks.
3. **Richer temporal models.** Replacing or augmenting the actor with recurrent or attention-based architectures that allow for sequence modelling with understanding of temporal relationships should reduce reliance on explicit state channels by enabling implicit velocity and gate order estimation directly from observation. Promising architectures for this are LSTMs [12], or Transformers [33].
4. **Adversarial or diversity-driven task generation.** Incorporating adversarially generated tracks (aimed at exploiting policy weaknesses) or explicit coverage objectives for the task generator could produce more robust policies and provide a more rigorous out-of-distribution test-bed.
5. **Robustness testing.** Systematic evaluation under aerodynamic disturbances, sensor noise, and hardware variations will better quantify real-world readiness.
6. **Interpretability and behaviour analysis.** Tools for clustering qualitatively similar trajectories, visualising saliency in image embeddings, or extracting prototypical manoeuvres will help diagnose whether a given policy has learned a strategy or is memorising trajectories.

How does this approach apply to real-world contexts? The small sim-to-real gap and the vision-enabled results indicate practical promise: a generalist pre-trained policy can substantially reduce the effort of deployment in new venues and serve as a fast fine-tuning starting point. Given this, we believe that this work has significant potential to expand the capability of autonomous aerial agents in the real world. However, safety-critical deployment requires additional precautions: provably safe controllers, formal verification for critical flight envelopes, and confirmation of perception reliability under adverse conditions. From a systems perspective, reducing the need for external motion-capture systems by closing the remaining perception gaps (e.g. via temporal models or additional sensors such as event cameras) is a critical next step.

4.2 Conclusion

This work introduced a training paradigm that enables zero-shot generalisation in reinforcement learning for high-speed autonomous drone racing. By combin-

ing large-scale parallelised training, informed procedural task generation, adaptive task switching, and improved hyperparameters, we demonstrated that it is possible to train a single policy capable of robustly flying a broad range of unseen racetracks both in simulation and on real hardware without any task-specific fine-tuning. In doing so, we established a new benchmark for generalisation in this domain and presented, to our knowledge, the first demonstration of a vision-based drone racing agent that generalises zero-shot to unseen tracks.

The results suggest that the long-standing trade-off between performance and generalisation in RL for drone racing is not fundamental. While previous approaches tended to sacrifice speed for the ability to race unseen tracks (or vice versa), our generalist agent achieves competitive lap times while retaining strong ZSG. This finding indicates that careful attention to task diversity, environment design during training, and hyperparameter tuning can substantially improve generalisation without significantly eroding control performance. Moreover, the observed acceleration in fine-tuning from the generalist model implies that generalisation and adaptability can be mutually reinforcing rather than competing objectives.

The implications of this work extend to RL and robotics more broadly. Our method scales to a high-dimensional, structured task space and provides a practical recipe for producing policies that generalise across diverse conditions without resorting to online adaptation. The adaptive task-switching mechanism further introduces a lightweight means of managing curricula automatically, removing the need for manual tuning of task schedules. Together, these elements contribute toward a general framework for scalable and robust RL in varied, real-world settings.

Despite these advances, several open challenges remain. Our current evaluation focuses on within-distribution generalisation, and performance under out-of-distribution tasks remains untested. The vision-based agent also relies on partial state information, underscoring the need for improved architectures or richer perception models to achieve fully onboard autonomy. Addressing these limitations will be essential for real-world deployment, particularly in scenarios without motion-capture systems or with constrained sensing modalities.

Looking ahead, the generalist agent developed here could serve as a foundation model for aerial robotics: a pre-trained, broadly capable policy that can be rapidly fine-tuned for new environments or objectives. Integrating meta-learning and continual-learning techniques could enable such models to adapt online while retaining broad competence. Similarly, adversarial or diversity-driven task generation could strengthen robustness and provide a more rigorous test of ZSG.

In summary, this work demonstrates that through principled multi-task learning, RL can yield policies that are both fast and generalisable. By closing much of the gap between task-specific optimisation and true generalisation, we move toward the long-term goal of autonomous aerial agents that operate safely, efficiently, and intelligently in the open world.

Appendix A

Appendix

A.1 Spline-Based Track Generation

The track generation process constructs a smooth, closed trajectory defined by a periodic spline that passes through a set of spatially constrained control points. Evenly spaced gates are then placed along this trajectory, each oriented according to the local tangent direction of the spline. The resulting path ensures both geometric feasibility and spatial smoothness suitable for drone racing tasks.

Control Point Sampling. A set of N_c control points

$$P = \{p_1, p_2, \ldots, p_{N_c}\}, \quad p_i \in \mathbb{R}^3$$

is sampled uniformly within the bounded spatial region

$$B = [x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}] \times [z_{\min}, z_{\max}],$$

subject to the following constraints:

$$\text{(a) Horizontal spacing:} \quad \|p_i^{xy} - p_j^{xy}\|_2 \geq d_{\min}, \quad \forall i \neq j, \tag{A.1}$$

$$\text{(b) Boundary padding:} \quad x_{\min} + \delta_b \leq p_i^x \leq x_{\max} - \delta_b, \tag{A.2}$$

$$y_{\min} + \delta_b \leq p_i^y \leq y_{\max} - \delta_b, \tag{A.3}$$

where $p_i^{xy} = [p_i^x, p_i^y]^\top$, $d_{\min}$ denotes the minimum horizontal spacing, and δ_b the required padding from the environment boundaries. The control points are re-centered within B while preserving these constraints by translating all points by the average of their positions.

Periodic Spline Construction. A closed periodic B-spline $\mathbf{s}(t) : [0, 1] \to \mathbb{R}^3$ is fitted through the control points:

$$\mathbf{s}(t) = \text{B-spline}(P, s),$$

where s is a smoothing factor regulating curvature. The spline is defined by a tuple (t, c, k) obtained via `scipy.interpolate.splprep`, enforcing periodicity:

$$\mathbf{s}(0) = \mathbf{s}(1), \quad \mathbf{s}^{(n)}(0) = \mathbf{s}^{(n)}(1) \quad \forall n \leq k.$$

This corresponds to the GenerateSpline function in Algorithm 2.

Gate Placement. Let $L = \int_0^1 \|\mathbf{s}'(t)\|\, dt$ denote the total arc length of the spline. A set of N_g gates is positioned at approximately equal arc-length intervals:

$$g_i = \mathbf{s}(t_i), \quad \text{where} \quad \int_0^{t_i} \|\mathbf{s}'(t)\|\, dt = \frac{iL}{N_g}, \quad i = 0, \ldots, N_g - 1.$$

Each gate is assigned an orientation quaternion q_i derived from the tangent direction:

$$\mathbf{t}_i = \frac{\mathbf{s}'(t_i)}{\|\mathbf{s}'(t_i)\|}, \quad \theta_i = \arctan 2(t_i^y, t_i^x), \quad q_i = R_z(\theta_i),$$

where $R_z(\theta_i)$ represents a rotation about the yaw axis aligning the gate with the local trajectory direction. Gate altitudes are clamped within

$$z_{\min} + \tfrac{h_g}{2} \le g_i^z \le z_{\max} - \tfrac{h_g}{2}$$

to ensure the full gate geometry remains inside the environment bounds. This corresponds to the PlaceGates function in Algorithm 2.

Start Pose. The drone start pose is placed 2 m behind the first gate along its negative forward axis:

$$x_{\text{start}} = g_0 - 2.0\, \hat{f}_0,$$

where $\hat{\mathbf{f}}_1$ is the forward direction derived from q_0. The starting attitude is identical to that of the first gate, $q_{\text{start}} = q_0$, so that the drone begins pointing towards the first gate; this is necessary particularly for the vision-based agent to be able to see the first gate in its image observation.

Constraint Validation. A candidate track is accepted only if the following constraints are satisfied:

$$\begin{aligned}
\text{Gate spacing:} \quad & \|g_i - g_j\|_2 \ge d_g^{\min}, \quad \forall i \ne j, \\
\text{Boundary constraint:} \quad & x_{\min} + \delta_b \le g_i^x \le x_{\max} - \delta_b, \\
& y_{\min} + \delta_b \le g_i^y \le y_{\max} - \delta_b, \\
\text{Start position:} \quad & x_{\text{start}} \in [x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}] \times [z_{\min}, z_{\max}].
\end{aligned}$$

These constraints form the set C for Algorithm 2. If any of these constraints are violated, the generation process resamples control points and repeats until a valid configuration is found or the maximum number of attempts M (as defined in Algorithm 2) is exceeded.

A.2 Task Switching Metric Comparison

There are a number of alternatives for task switching to the Spearman-based confidence score proposed in Section 2.2.4. They primarily involve estimating the derivative of the reward curve and switching task when the derivative is close to zero. The discrete derivative can be computed directly from each task's

reward curve online, however given the extremely noisy nature of the reward curves (as visible in Figure 2.1) this does not produce a useful signal.

Finite differences can approximate the derivative of the discrete time series data. However, this is extremely sensitive to noise and was found to generate a discrete derivative of the reward curves that was rarely close to zero, even when the reward curve had visibly flattened. This was addressed with various smoothing methods, which removed noise before the reward gradient was estimated.

Exponentially Weighted Moving Average (EWMA) can smooth the time series by taking a weighted average of the most recent datapoints in a moving window, and the derivative then computed from this smoothed signal. The datapoints' weight decreases exponentially with time so that more recent data is weighted more heavily. This was found to be too sensitive to recent outliers, as a single episode with an abnormally high or low reward would significantly disrupt the gradient estimation.

Curve regression models the reward curve as a following a parametric function (such as a logistic curve) and uses a least-squares regression to fit the function to the data at every timestep. The derivative can then be computed analytically to avoid the error introduced by finite differences. While this worked well in some cases, it relies on an unfounded assumption about the underlying shape of the reward curve and hence failed completely for reward curves that deviated from the commonly observed sigmoid shape.

In practice, we found the Spearman-based confidence score to be the most robust to noise and the most effective across a range of different reward curve shapes.

A.3 Hyperparameters

We follow the nomenclature from the Stable Baselines 3 PPO implementation [27] for PPO-related hyperparameters. Reward function weight values can be found in Table 2.1. Track controller hyperparameters use the nomenclature introduced in Sections 2.2.4 and 2.2.3, and in appendix A.1. SB and SB denote state-based and vision-based respectively.

Parameter	Value (SB)	Value (VB)
Initial Log std	-0.5	0.0
GAE Lambda	0.95	0.95
gamma	0.995	0.995
n_steps	250	250
learning_rate start	3.0×10^{-4}	2.0×10^{-4}
learning_rate end	1.0×10^{-5}	1.0×10^{-5}
vf_coeff	0.5	0.5
max_grad_norm	0.5	0.5
batch_size	28000	24000
n_epochs	10	4
clip_range	0.2	0.2

Table A.1: PPO hyperparameters

Parameter	Value (SB)	Value (VB)
mass (kg)	0.05	0.05
inertia (kgm^2)	0.2	0.2
drag	0.5	0.2
thrust map	0.2	0.2
delay (s)	0.02	0.01
x position (m)	0.8	0.4
y position (m)	0.8	0.4
z position (m)	0.6	0.3
x velocity (m/s)	0.8	0.4
y velocity (m/s)	0.8	0.4
z velocity (m/s)	0.8	0.4
roll (deg)	20	10
pitch (deg)	20	10
yaw (deg)	20	10
roll rate (deg/s)	45	25
pitch rate (deg/s)	45	25
yaw rate (deg/s)	45	25

Table A.2: Initial condition domain randomisation hyperparameters

Parameter	Value (SB)	Value (VB)
l	600	600
α	1000	100
N_c	6	6
N_g	6	6
s	4.0	4.0
δ_b (m)	2.0	2.0
$d_g^{\min}$ (m)	2.0	2.0
M	100	100

Table A.3: Track controller hyperparameters

Bibliography

[1] Karl Cobbe, Christopher Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning.

[2] Karl Cobbe, Oleg Klimov, Chris Hesse, Taehoon Kim, and John Schulman. Quantifying generalization in reinforcement learning.

[3] Michael Dennis, Natasha Jaques, Eugene Vinitsky, Alexandre Bayen, Stuart Russell, Andrew Critch, and Sergey Levine. Emergent complexity and zero-shot transfer via unsupervised environment design.

[4] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking deep reinforcement learning for continuous control.

[5] Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Volodymir Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, Shane Legg, and Koray Kavukcuoglu. IMPALA: Scalable distributed deep-RL with importance weighted actor-learner architectures.

[6] Robin Ferede, Guido C. H. E. de Croon, Christophe De Wagter, and Dario Izzo. End-to-end neural network based quadcopter control. 172:104588.

[7] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks.

[8] Philipp Foehn, Elia Kaufmann, Angel Romero, Robert Penicka, Sihao Sun, Leonard Bauersfeld, Thomas Laengle, Giovanni Cioffi, Yunlong Song, Antonio Loquercio, and Davide Scaramuzza. Agilicious: Open-source and open-hardware agile quadrotor for vision-based flight. 7(67):eabl6259. Publisher: American Association for the Advancement of Science.

[9] Philipp Foehn, Angel Romero, and Davide Scaramuzza. Time-optimal planning for quadrotor waypoint flight. 6(56):eabh1221.

[10] Robert M. French. Catastrophic forgetting in connectionist networks. 3(4):128–135.

[11] Ismail Geles, Leonard Bauersfeld, Angel Romero, Jiaxu Xing, and Davide Scaramuzza. Demonstrating agile flight from pixels without state estimation.

[12] Sepp Hochreiter and JÃ¼rgen Schmidhuber. Long short-term memory. 9(8):1735–1780.

[13] Myles Hollander, Douglas A. Wolfe, and Eric Chicken. *Nonparametric Statistical Methods*. John Wiley & Sons. Google-Books-ID: Y5s3AgAAQBAJ.

[14] Julian Ibarz, Jie Tan, Chelsea Finn, Mrinal Kalakrishnan, Peter Pastor, and Sergey Levine. How to train your robot with deep reinforcement learning: lessons we have learned. 40(4):698–721. Publisher: SAGE Publications Ltd STM.

[15] Minqi Jiang, Edward Grefenstette, and Tim RocktÃ¤schel. Prioritized level replay.

[16] Elia Kaufmann, Leonard Bauersfeld, Antonio Loquercio, Matthias MÃ¼ller, Vladlen Koltun, and Davide Scaramuzza. Champion-level drone racing using deep reinforcement learning. 620(7976):982–987. Publisher: Nature Publishing Group.

[17] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization.

[18] Robert Kirk, Amy Zhang, Edward Grefenstette, and Tim RocktÃ¤schel. A survey of zero-shot generalisation in deep reinforcement learning. 76:201–264.

[19] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks. 114(13):3521–3526. Publisher: Proceedings of the National Academy of Sciences.

[20] Ashish Kumar, Zipeng Fu, Deepak Pathak, and Jitendra Malik. RMA: Rapid motor adaptation for legged robots.

[21] Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. Learning quadrupedal locomotion over challenging terrain. 5(47):eabc5986. Publisher: American Association for the Advancement of Science.

[22] Bo Liu, Xingchao Liu, Xiaojie Jin, Peter Stone, and Qiang Liu. Conflict-averse gradient descent for multi-task learning.

[23] Reza Moradi, Reza Berangi, and Behrouz Minaei. A survey of regularization strategies for deep models. 53(6):3947–3986.

[24] OpenAI, Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, Jonas Schneider, Szymon Sidor, Josh Tobin, Peter Welinder, Lilian Weng, and Wojciech Zaremba. Learning dexterous in-hand manipulation.

[25] Jack Parker-Holder, Minqi Jiang, Michael Dennis, Mikayel Samvelyan, Jakob Foerster, Edward Grefenstette, and Tim RocktÃ¤schel. Evolving curricula with regret-based environment design. In *Proceedings of the 39th International Conference on Machine Learning*, pages 17473–17498. PMLR. ISSN: 2640-3498.

[26] Chao Qin, Maxime S. J. Michet, Jingxiang Chen, and Hugh H.-T. Liu. Time-optimal gate-traversing planner for autonomous drone racing.

[27] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. 22(268):1–8.

[28] Angel Romero, Robert Penicka, and Davide Scaramuzza. Time-optimal online replanning for agile quadrotor flight. 7(3):7730–7737.

[29] Angel Romero, Sihao Sun, Philipp Foehn, and Davide Scaramuzza. Model predictive contouring control for time-optimal quadrotor flight. 38(6):3340–3356.

[30] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms.

[31] Yunlong Song, Selim Naji, Elia Kaufmann, Antonio Loquercio, and Davide Scaramuzza. Flightmare: A flexible quadrotor simulator.

[32] Yunlong Song, Angel Romero, Matthias MÃ¼ller, Vladlen Koltun, and Davide Scaramuzza. Reaching the limit in autonomous racing: Optimal control versus reinforcement learning. 8(82):eadg1462. Publisher: American Association for the Advancement of Science.

[33] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need.

[34] Nelson Vithayathil Varghese and Qusay H. Mahmoud. A survey of multi-task deep reinforcement learning. 9(9):1363. Number: 9 Publisher: Multidisciplinary Digital Publishing Institute.

[35] Dylan Vogel, Robert Baines, Joseph Church, Julian Lotzer, Karl Werner, and Marco Hutter. Robust ladder climbing with a quadrupedal robot.

[36] Hongze Wang, Jiaxu Xing, Nico Messikommer, and Davide Scaramuzza. Environment as policy: Learning to race in unseen tracks.

[37] Rui Wang, Joel Lehman, Jeff Clune, and Kenneth O. Stanley. Paired open-ended trailblazer (POET): Endlessly generating increasingly complex and diverse learning environments and their solutions.

[38] Peter R. Wurman, Samuel Barrett, Kenta Kawamoto, James MacGlashan, Kaushik Subramanian, Thomas J. Walsh, Roberto Capobianco, Alisa Devlic, Franziska Eckert, Florian Fuchs, Leilani Gilpin, Piyush Khandelwal, Varun Kompella, HaoChih Lin, Patrick MacAlpine, Declan Oller, Takuma Seno, Craig Sherstan, Michael D. Thomure, Houmehr Aghabozorgi, Leon Barrett, Rory Douglas, Dion Whitehead, Peter DÃ¼rr, Peter Stone, Michael Spranger, and Hiroaki Kitano. Outracing champion gran turismo drivers with deep reinforcement learning. 602(7896):223–228. Publisher: Nature Publishing Group.

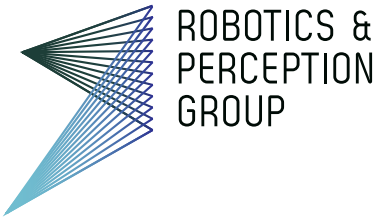
[39] Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. Gradient surgery for multi-task learning.

[40] Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Avnish Narayan, Hayden Shively, Adithya Bellathur, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning.

[41] Chiyuan Zhang, Oriol Vinyals, Remi Munos, and Samy Bengio. A study on overfitting in deep reinforcement learning.

[42] Yi Zhou, Connelly Barnes, Jingwan Lu, Jimei Yang, and Hao Li. On the continuity of rotation representations in neural networks.

[43] Luisa Zintgraf, Kyriacos Shiarlis, Maximilian Igl, Sebastian Schulze, Yarin Gal, Katja Hofmann, and Shimon Whiteson. VariBAD: A very good method for bayes-adaptive deep RL via meta-learning.



Title of work:

Zero-Shot Generalisation with Reinforcement Learning for Agile Drone Flight

Thesis type and date:

Master Thesis, October 2025

Supervision:

Jiaxu Xing
Nico Messikommer
Angel Romero Aguilar
Prof. Dr. Davide Scaramuzza

Student:

Name:	Jonathan Jacob Kolt Green
E-mail:	gjonatha@ethz.ch
Legi-Nr.:	23-942-014

Statement regarding plagiarism:

By signing this statement, I affirm that I have read the information notice on plagiarism, independently produced this paper, and adhered to the general practice of source citation in this subject-area.

Information notice on plagiarism:

https://www.ius.uzh.ch/dam/jcr:00000000-100f-6f7c-ffff-ffff869554ef/LK_Plagiarism.pdf

Zurich, 7. 10. 2025: