Rapid Sim-to-Real Reinforcement Learning for Autonomous Drone Racing

Joel Klimont*, Jakob Buchsteiner, Alexander Lampalzer and Radu Grosu
Technical University of Vienna, Austria

http://www.imavs.org/

ABSTRACT

Fast sim-to-real iteration is important in autonomous drone racing, where each new track or vehicle requires adaptation under a strict wall-clock budget. We present a JAX-based reinforcement-learning pipeline that trains compact feedforward policies in a massively parallel environment. We identify a compact control-oriented acro-mode quadrotor model from short flight logs and randomize its transfer-relevant parameters. The model captures closed-loop body-rate and thrust response, battery-voltage effects, and direction-dependent drag. To suppress bang-bang control, we regularize action changes and large commands. Over a four-second rollout, this reduces the fraction of outputs above 95% of the maximum action magnitude from 50.0% to 1.1% while maintaining comparable track progress. On an NVIDIA GeForce RTX 5070 Ti, the fastest configuration processes 32 million environment steps per second. A 96-configuration study and a 32-seed evaluation show that rollout and optimizer choices strongly affect training reliability within five minutes. Two policies deployed on the A2RL Season 2 competition track reached speeds near $20\,\mathrm{m/s}$. These results demonstrate rapid policy training and real-world transfer using a compact dynamics model.

1 INTRODUCTION

Autonomous drone racing is a demanding benchmark for agile robotics because it combines high-speed motion, tight spatial constraints, limited onboard computation, and little tolerance for control error. The field has progressed from early academic racing challenges [1–4] to real-world systems operating above $20\,\mathrm{m/s}$ [5, 6]. Learning-based controllers have become competitive when simulation, objectives, and transfer assumptions are designed together [6–8].

Competition deployment requires fast adaptation. High-fidelity simulators can reduce model mismatch, but demand detailed models, greater implementation effort, and more computation. Compact control-oriented models can be estimated from short flight logs and simulated at high throughput, provided they retain the dynamics that govern transfer. This trade-off is central to racing, where every new track requires policy retraining and a new vehicle additionally requires model identification.

We identify a compact control-oriented quadrotor model from short flight logs and use it for competition-scale policy training. The policy outputs body-rate and thrust commands through the acro interface used by racing flight controllers. The simulator omits motor-level actuation and moment-based attitude dynamics and instead models the closed-loop response from policy commands to vehicle motion. We train with proximal policy optimization (PPO) [9]; its batched rollout and update structure maps well to a massively parallel JAX implementation [10] and supports stable training under short wall-clock budgets. Because the compact model omits inner-loop dynamics, the objective penalizes rapid action changes and persistently large deterministic commands that would otherwise produce bang-bang behavior at the deployed interface.

The main contribution is a practical sim-to-real workflow that combines short-log model identification, high-throughput policy training, configuration screening under a fixed wall-clock budget, and real-world validation. We evaluate 96 training configurations, repeat three representative settings over 32 seeds, quantify the effect of action regularization, and deploy two policies on the A2RL Season 2 competition track at speeds near $20\,\mathrm{m/s}$. The training environment and reported configurations are publicly available.[1]

2 RELATED WORK

Drone racing stresses perception, estimation, planning, control, and onboard computation simultaneously [2, 4]. Model-based systems remain strong baselines: model predictive contouring control optimizes progress under dynamic constraints [11], while sampling-based methods such as MPPI perform online trajectory optimization through parallel rollouts [12]. Our compact model serves a different purpose: it supports offline policy training and configuration selection, while deployment requires only a feedforward-network evaluation.

Recent reinforcement-learning systems have achieved highly competitive real-world performance. Swift demonstrated champion-level racing with simulation-trained policies and data-driven residual models [7]. Ferede et al. used domain randomization to transfer a motor-command policy

*Email address(es): joel.klimont@tuwien.ac.at

[1] Project repository: https://github.com/F-WuTS/jax-drone-racing-open.

http://www.imavs.org/

across substantially different quadrotors [8]. SkyDreamer and MonoRace address high-speed racing under realistic visual sensing constraints [5, 6]. We study a complementary state-based setting in which a deliberately simple simulator enables very high-throughput policy training while retaining sufficient fidelity for real-world deployment.

Direct training-time comparisons are difficult because control interfaces, tasks, hardware, and deployment criteria differ. Swift reports 10^8 environment interactions in approximately 50 min on an RTX 3090, followed by a 2×10^7-interaction fine-tuning stage [7]. Learning to Fly in Seconds reports direct-RPM trajectory-control transfer after 18 s of training on a consumer laptop and a Crazyflie nano quadrotor [13]. Ferede et al. train each racing-policy variant for 10^8 steps across 3-inch and 5-inch platforms [8], but do not report a directly comparable wall-clock time. For broad context, our $(6144, 64, 16, 4)$ configuration uses 6144 environments, 64 rollout steps, 16 minibatches, and four update epochs. It generates more than 5×10^9 environment transitions in five minutes, while the fastest tested configuration reaches 3.2×10^7 transitions per second on an RTX 5070 Ti. These figures are not a direct speed ranking; we instead ask whether a competition-scale gate-racing policy can be trained reliably within five minutes.

The sim-to-real gap remains central at high speed. Errors in thrust, drag, response delay, battery behavior, and state estimation can substantially change closed-loop behavior. Domain randomization addresses these errors by training over a distribution of model and observation parameters [7, 8]. Related identification work highlights thrust curves, aerodynamic drag, and actuator dynamics as important for real flight [14, 15]; we retain these effects in a compact control-oriented form. Continuous-control policies can also converge to saturated or bang-bang commands [16], and policy-smoothness regularization can reduce high-frequency actuation on real robots [17]. We apply this idea directly to the body-rate and thrust interface and evaluate the resulting command signals.

3 PROBLEM STATEMENT

We train a compact feedforward neural policy for high-speed autonomous drone racing. The track is represented as an ordered cyclic sequence of gates

$$\mathcal{G} = \{G_i\}_{i=1}^{N}, \tag{1}$$

where each gate has a pose and aperture radius. For the A2RL track, the sequence also contains two virtual gates near the chicane and ladder. In these sections, the next physical gate alone does not sufficiently constrain the intended route and permits shortcuts or trajectories that reduce visibility of subsequent gates. The virtual gates use the same transition and reward logic as physical gates, but do not represent physical obstacles. A lap is completed after the vehicle traverses the ordered sequence and crosses the first gate again. The objective is to minimize lap time while avoiding missed gates, track-boundary violations, and unsafe flight states.

The learned policy is deliberately track-specific: gate geometry enters both the observation and reward, so a materially different layout requires retraining. The short training budget is intended to make this adaptation practical. We focus on control learning; perception and state estimation provide the track-relative policy state but are outside the scope of this paper.

3.1 Learning Formulation

The real task is partially observable because of estimator uncertainty, actuation uncertainty, latency, battery variation, and unmodeled aerodynamics. During training, we approximate it as a randomized finite-horizon Markov decision process with noisy observations. The simulator state is

$$s_t = (x_t, i_t, h_t, \theta), \tag{2}$$

where x_t is the vehicle state, i_t the target-gate index, h_t recent actions, and θ sampled physical parameters. The policy receives a noisy observation $\tilde{o}_t$, samples an unconstrained Gaussian action, and applies a hyperbolic tangent,

$$\begin{aligned} &\hat{a}_t \sim \pi_\phi(\hat{a}_t \mid \tilde{o}_t), \\ &a_t = \tanh(\hat{a}_t) \in [-1, 1]^4, \end{aligned} \tag{3}$$

where a_t contains normalized roll-rate, pitch-rate, yaw-rate, and thrust commands. PPO optimizes

$$J(\phi) = \mathbb{E}\left[\sum_{t=0}^{H-1} \gamma^t r_t\right] \tag{4}$$

over randomized racing episodes.

4 METHODS

We describe the compact simulator, policy interface, reward design, and randomized PPO training setup. The environment is deliberately simpler than a motor-level quadrotor simulator, but retains the closed-loop response effects selected as most relevant for transfer.

4.1 Compact Quadrotor Model

The simulated state contains position, velocity, attitude, filtered command state, battery voltage, and body rates,

$$x_t = (p_t, v_t, q_t, u_t, V_t, \omega_t). \tag{5}$$

The bounded policy commands $a_t = \tanh(\hat{a}_t) \in [-1, 1]^4$ pass through a first-order response model,

$$u_t = u_{t-1} + \alpha(a_t - u_{t-1}), \qquad \alpha = 1 - \exp\left(-\frac{\Delta t}{\tau}\right), \tag{6}$$

with separate time constants for roll, pitch, yaw, and thrust. Filtered rate commands are scaled by the identified maximum

http://www.imavs.org/

Table 1: Model-identification errors on the chronological validation split. Rate errors are in deg/s and thrust errors in m/s^2.

Response	τ [ms]	RMSE	MAE	R^2
Roll rate	6.7	10.68	7.37	0.994
Pitch rate	12.1	12.94	9.61	0.978
Yaw rate	12.4	6.24	4.33	0.976
Thrust	16.8	0.29	0.16	0.993

body rates and integrated directly into the attitude. The simulator therefore omits individual motors, moment-based attitude dynamics, propeller aerodynamics, and the internal rate controller.

Collective thrust is computed from filtered thrust and battery voltage as

$$f_T = c_0 + c_1 u_{t,\mathrm{thr}} + c_2 u_{t,\mathrm{thr}}^2 + c_3 u_{t,\mathrm{thr}}^3 + c_4 V_t + c_5 u_{t,\mathrm{thr}} V_t. \tag{7}$$

The translational dynamics combine thrust, gravity, and aerodynamic drag,

$$\dot{p}_t = v_t, \tag{8}$$

$$\dot{v}_t = R(q_t)\begin{bmatrix}0 & 0 & f_T/m\end{bmatrix}^\top + \begin{bmatrix}0 & 0 & -g\end{bmatrix}^\top + f_D/m. \tag{9}$$

Drag uses an ellipsoid approximation of the vehicle body, making projected area direction-dependent, and battery voltage follows a linear discharge model. The result is a fast control-oriented approximation whose dominant uncertain parameters can be randomized during training. The exact nominal response, thrust, drag, and body-rate parameters are provided in the project repository.

We identify the model from approximately five minutes of manually flown data spanning the battery-discharge range. After preprocessing, the body-rate dataset contains 40,389 samples and the thrust dataset 339,948 samples. Both use a chronological 70/30 training-validation split, yielding 28,272/12,117 rate samples and 237,963/101,985 thrust samples. For each rate axis, bounded nonlinear least squares estimates the first-order time constant by minimizing sample-wise setpoint-response residuals; a constant offset is computed from the mean residual. For thrust, bounded nonlinear least squares estimates the response time constant, while the six coefficients in Eq. (7) are solved by linear least squares for each candidate time constant to minimize body-frame vertical-acceleration error. Figure 1 compares measured and predicted body-frame vertical acceleration, and Table 1 reports validation errors. Additional validation plots and the exact nominal parameters are provided in the project repository.

4.2 Observation and Action Interface

On the real system, an EKF fuses IMU data and gate detections to estimate the vehicle state in the world frame.

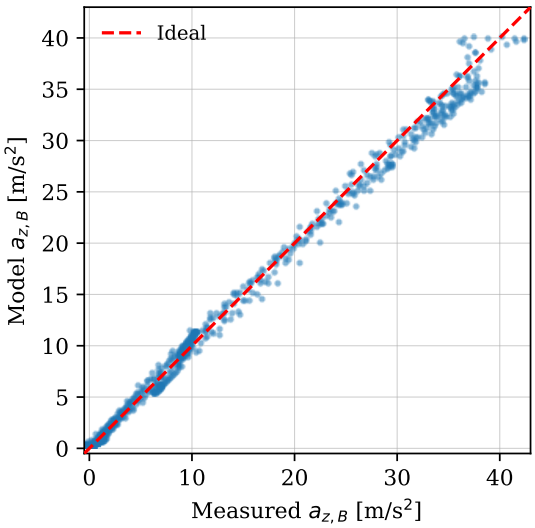
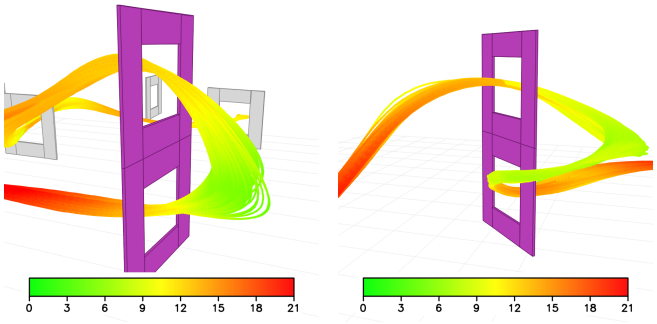


Figure 1: Measured and predicted body-frame vertical acceleration on the chronological validation split.



(a) Split-S (b) Ladder

Figure 2: Complex maneuvers on the A2RL Season 2 track. Color denotes policy speed in meters per second.

The racing environment converts this estimate into current-gate coordinates, keeping the policy input independent of the global track frame while preserving the same feature construction in simulation and deployment. The observation is

$$o_t = [p_t^{G_i}, v_t^{G_i}, \xi_t^{G_i}, \omega_t^{G_i}, u_t, a_{t-1:t-3}, V_t, d_{i,i+1}^{G_i}, d_{i,i+2}^{G_i}], \tag{10}$$

where $p_t^{G_i}$ and $v_t^{G_i}$ denote position and velocity in the current gate frame, $\xi_t^{G_i}$ is relative attitude, $\omega_t^{G_i}$ are gate-frame angular rates, u_t is the filtered command state, $a_{t-1:t-3}$ are the previous three actions, V_t is battery voltage, and the final terms describe the next two gate offsets. Gaussian observation noise approximates estimator uncertainty during training.

4.3 Gate Passing and Termination

Each gate defines a plane and aperture disk. A pass is registered only when the drone crosses the plane through the

http://www.imavs.org/

disk; crossings outside the disk are missed gates. Episodes terminate after a missed gate, flight-volume violation, battery or altitude safety violation, gate timeout, or maximum horizon. We do not add a separate gate-collision penalty beyond termination and the lost opportunity for further progress, since preliminary penalties encouraged overly conservative flight near gates, consistent with Ferede et al. [8].

4.4 Reward and Action Regularization

The task reward combines radial progress toward the active gate, a sparse gate-passing bonus, lap-time pressure, altitude and timeout penalties, and a distance-weighted gate-facing term. The latter discourages the vehicle from pointing away from a nearby gate and improves observability for the onboard EKF by keeping upcoming gates in view.

To discourage control behavior that exploits omitted inner-loop dynamics, we penalize temporal changes in consecutive squashed sampled actions,

$$\tilde{r}_t = r_t - \lambda_{\Delta a} \left\| \tanh(\hat{a}_t) - \tanh(\hat{a}_{t-1}) \right\|_2^2, \qquad (11)$$

and add an ℓ_2 penalty on the squashed deterministic policy mean,

$$\mathcal{L}_{\text{actor}} = \mathcal{L}_{\text{PPO}} + \lambda_a \mathbb{E}_t \left[\left\| \tanh(\mu_\phi(o_t)) \right\|_2^2 \right]. \qquad (12)$$

The first term suppresses rapid command variation; the second discourages unnecessarily large mean commands and persistent saturation. Section 5 evaluates their combined effect on the deployed command interface.

4.5 Policy Optimization and Reset Distribution

We train PPO in the vectorized JAX environment with independently randomized initial states, observations, actions, and dynamics parameters. Actor and critic are feedforward MLPs with two 64-unit $\tanh$ hidden layers. The same architecture is used throughout the configuration study, seed study, and deployment. All reported timing experiments use an NVIDIA GeForce RTX 5070 Ti with 16 GB of VRAM.

Resets are sampled along a cyclic Catmull-Rom spline through the start position and gate centers. Position, speed, direction, and attitude are perturbed, and the target gate is assigned consistently with sampled progress. This broadens state-space coverage without requiring the deployed policy to track the spline. The exact reset and noise parameters are provided in the project repository.

Physical parameters are randomized independently per episode,

$$\theta_{\text{sim}} = \theta_{\text{nom}} \odot \kappa, \qquad \kappa_i \sim \mathcal{U}(\kappa_i^{\min}, \kappa_i^{\max}). \qquad (13)$$

Table 2 lists the randomized parameters. The distribution is intentionally low-dimensional and focuses on effects that strongly influence the closed-loop response. Mass and gravity remain fixed because mass is largely confounded with thrust scale through f_T/m, while gravity is known accurately.

Table 2: Domain-randomized model parameters. Scale factors are sampled independently per episode.

Parameter	Randomization	Description
$\tau_\phi, \tau_\theta, \tau_\psi, \tau_T$	$\mathcal{U}(0.8, 1.2)$	Command-response time constants
$\omega^x_{\max}, \omega^y_{\max}, \omega^z_{\max}$	$\mathcal{U}(0.8, 1.2)$	Maximum body rates
s_T	$\mathcal{U}(0.8, 1.2)$	Thrust-output scale
C_D	$\mathcal{U}(0.8, 1.2)$	Drag scale

5 EVALUATION

We evaluate the workflow in four steps. First, we compare rollout and optimizer settings under the five-minute budget. We then test repeatability across seeds and quantify the effect of action regularization. Finally, we examine real-world transfer and where the compact model remains informative despite its simplifications.

The track includes the split-S and ladder shown in Figure 2. The split-S combines a large heading reversal with vertically separated gate passages, while the ladder requires rapid altitude changes without sacrificing forward speed. We use passage of these elements as intermediate training milestones.

5.1 Five-Minute Training-Configuration Study

Figure 3 summarizes a 96-run study over 12 combinations of parallel-environment count and rollout length. The tested environment counts are 2048, 4096, 6144, 8192, 10240, and 12288; the rollout lengths are 64, 128, 256, and 512 steps. The project repository lists the exact pairings. Each rollout configuration is combined with 8, 16, 32, or 64 minibatches and either four or six optimizer epochs, while the model, network, reward, and nominal five-minute target remain fixed. The complete offline configuration study totals 480 nominal GPU-minutes when run sequentially and is not part of the per-track deployment cost.

The timed interval begins with the first JAX training call and includes compilation, rollout collection, PPO updates, and snapshot capture. Setup occurs beforehand; benchmarks, host logging, checkpoint serialization, and export occur afterward. The fastest configuration reaches 32 million environment steps per second on the RTX 5070 Ti, but does not achieve the highest lap-completion rate. Some lower-throughput configurations learn faster and more reliably within the same budget, showing that rollout length, batch structure, and update allocation matter at least as much as raw simulation speed.

Snapshots are evaluated with fixed-start and random-start benchmarks using 256 agents in the configuration study and 512 in the seed study. Each benchmark includes one nominal model, while the remaining agents independently sample all randomized model parameters; observation and action noise are disabled. Rollouts last at most 1000 steps, or 10 s at 100 Hz. We call a policy deployment-ready when at

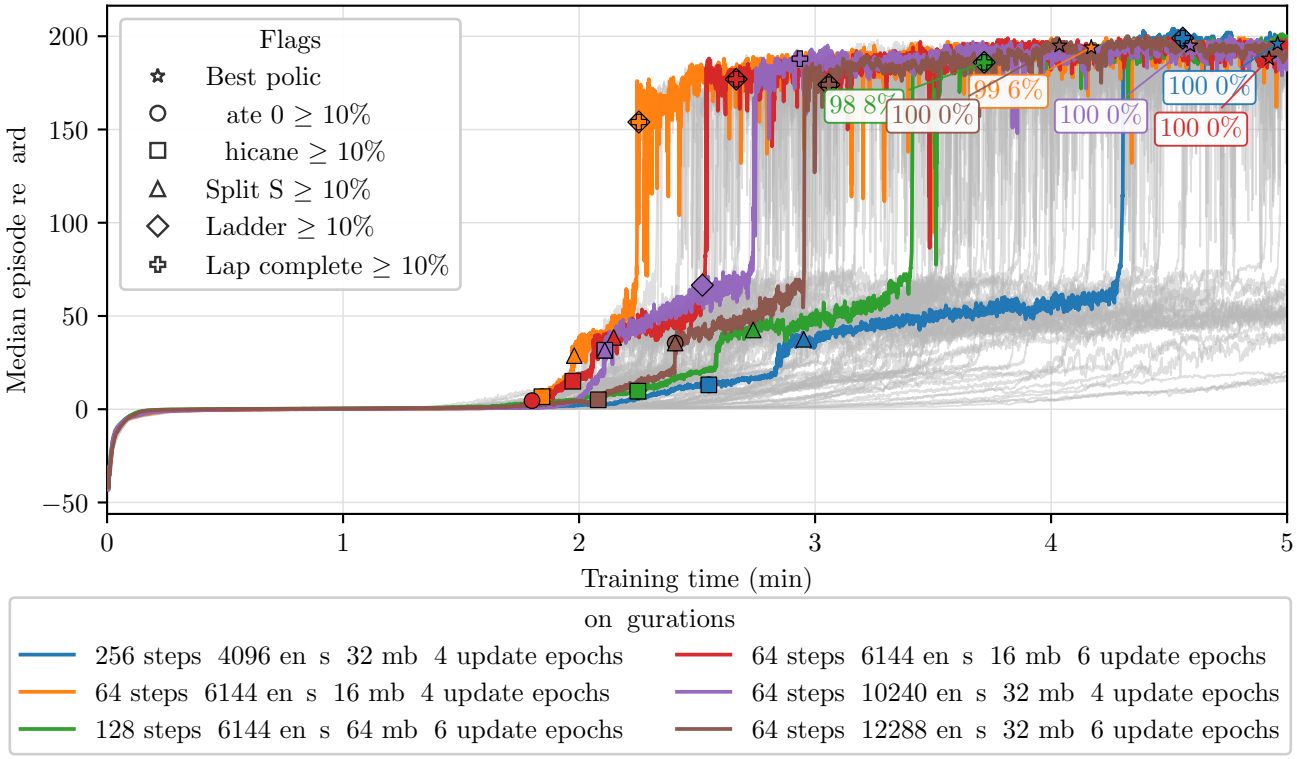


Figure 3: Training-configuration study over 96 PPO settings with a nominal five-minute target per run. Curves show median episode reward. Deferred domain-randomized benchmarks provide the track-section markers; stars identify snapshots selected by fixed-start lap completion. The six configurations with the highest fixed-start lap-completion rates are highlighted.

least 95% of fixed-start agents complete a lap. Figure 3 also marks the first snapshot for which at least 10% of random-start agents pass each indicated track section.

Several settings meet the 95% criterion within five minutes, whereas others make only partial progress or remain unreliable. After selecting a configuration through this offline study, the intended workflow uses a single nominal five-minute training run for a new track.

5.2 *Stability Across Seeds*

We repeat three configurations with 32 seeds each. Figure 4 shows that 8192×128 and 10240×64 learn faster and more reliably, whereas the failure-prone 4096×256 setting reaches the 95% criterion in only about 10% of seeds. The stronger results therefore do not depend on an isolated favorable seed.

5.3 *Effect of Action Regularization*

We compare weak and strong action regularization to assess whether the deployment-oriented objective suppresses bang-bang commands at the body-rate/thrust interface. Figure 6 shows the normalized policy outputs over the first four seconds of deterministic fixed-start rollouts. Both policies use the same model, seed, rollout configuration, and optimizer settings; only the action-delta, action-magnitude, and entropy coefficients differ.

The weak setting uses $\lambda_{\Delta a} = 0.001$, $\lambda_a = 0$, and entropy coefficient 10^{-3}; the strong setting uses $\lambda_{\Delta a} = 0.25$, $\lambda_a = 0.15$, and entropy coefficient 2.5×10^{-5}. The weakly regularized policy repeatedly switches between large commands, whereas the strongly regularized policy avoids rapid switching and persistent saturation. Over the four-second interval, the mean $\|a_t - a_{t-1}\|_2^2$ decreases from 4.237 to 0.00897, and the fraction of commands near saturation, defined by $|a_{t,j}| > 0.95$, decreases from 50.00% to 1.06%. This comparison evaluates the complete regularization setting rather than the contribution of each coefficient in isolation.

5.4 *Real-World Racing Performance*

We deploy deterministic mean policies using the same state construction and body-rate/thrust interface as in training. The real system uses Weitflug, a research-oriented fork of Betaflight for agile autonomous flight [18]. The simulated trajectory in Figure 5 uses the noise-free simulator state; the real trajectory is reconstructed from the onboard EKF estimate recorded during competition. The policy and simulator run at 100 Hz. The EKF propagates IMU measurements at 1 kHz and uses camera-based gate observations for correction.

http://www.imavs.org/

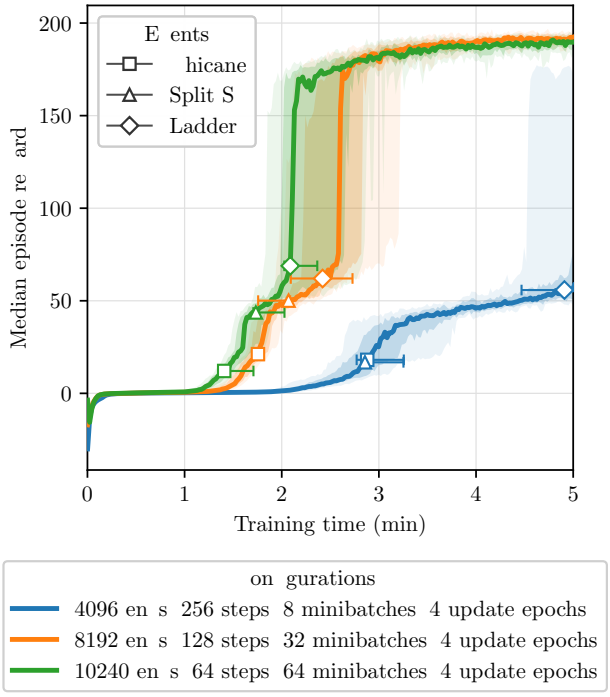


Figure 4: Seed-replication study for three representative PPO configurations. Curves and shaded quantile bands summarize median episode reward over 32 seeds per configuration; markers indicate benchmark track-section milestones.

The two policies with the highest fixed-start success rates were deployed in the Silver Cup finals. Each completed an uninterrupted two-lap run, one during a multi-drone race and the other during a timed race, with lap times agreeing within 0.1 s. The controller finished eight of ten race entries and won the Multi-Drone Silver Race [19]. The competition's VTX timing system recorded a best lap of 7.799 s, and the onboard EKF estimated a peak speed of approximately 20 m/s. The best human and autonomous laps on the same course were 6.283 s and 5.925 s, respectively.

As shown in Figure 5, the nominal and real racing lines agree well, with the largest deviation near the chicane. During this tight turn, the drone briefly points away from the next gate, reducing gate visibility and coinciding with a correction in the estimated trajectory. This behavior supports the distance-weighted gate-facing reward: progress alone is insufficient when the estimator loses informative visual references at critical track sections.

5.5 *Model Adequacy and Transfer*

This analysis evaluates robustness around the target vehicle and A2RL layout. The policy is track-specific, so a new layout requires retraining; a new vehicle also requires model identification. The five-minute workflow keeps this adaptation fast.

The largest sim-to-real mismatch occurs at the chicane in Figure 5. High speed, rapid direction changes, and reduced gate visibility make this section especially sensitive. In Figure 7, the randomized trajectories spread most strongly around the same sections, indicating that the compact simulator reveals transfer-sensitive regions despite omitting actuator-level dynamics.

The Catmull-Rom spline defines the reset distribution rather than a reference trajectory. The learned trajectories depart from it, especially around the ladder, showing that the spline broadens state coverage without imposing path following.

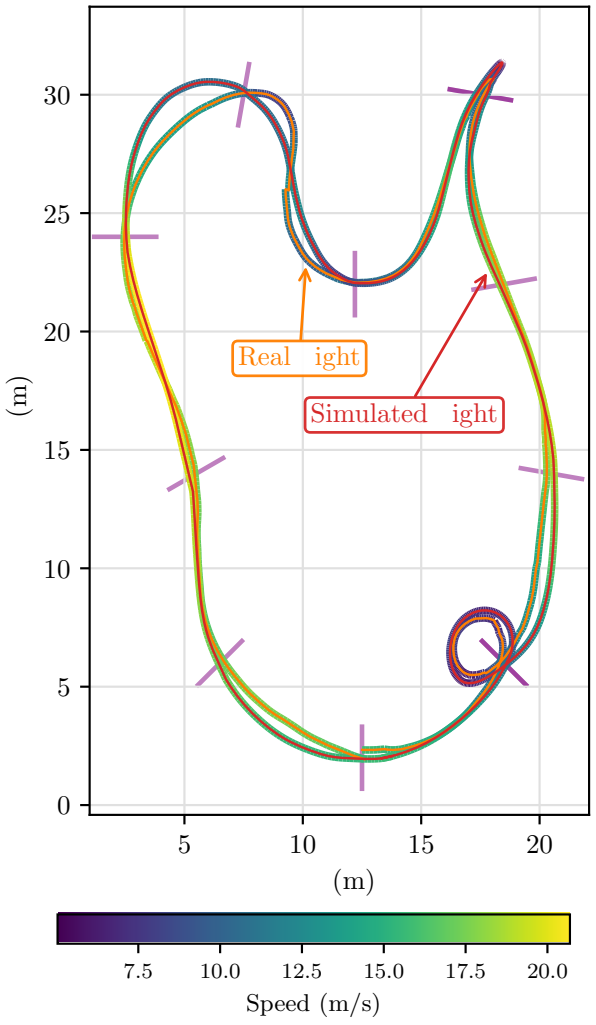


Figure 5: Top-down trajectory comparison for the same policy in nominal simulation and real flight. Color denotes speed; the real trajectory is reconstructed from the onboard EKF estimate.

http://www.imavs.org/

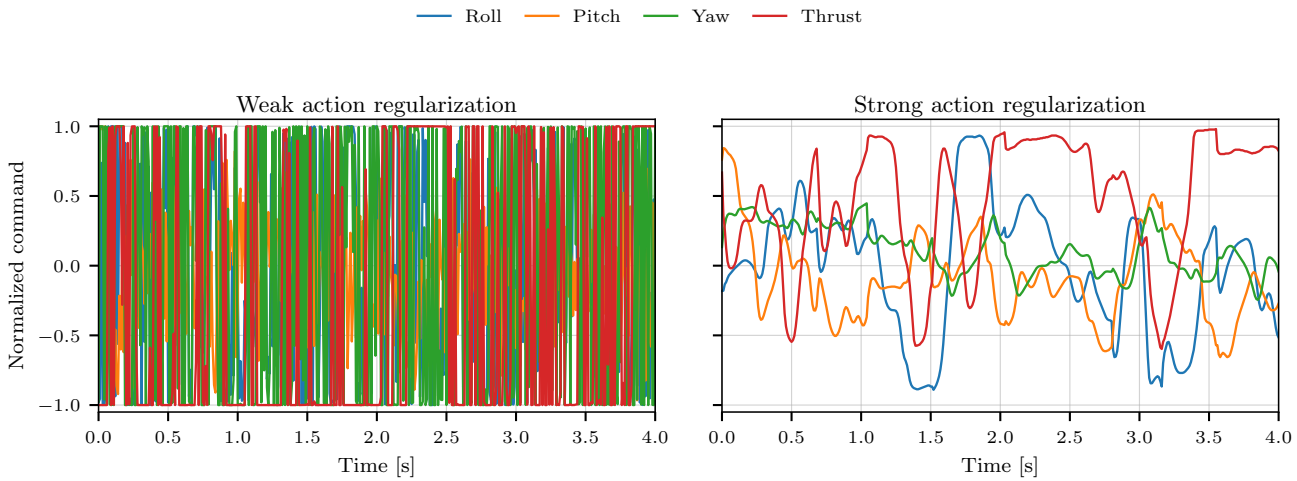


Figure 6: Normalized policy outputs over the first four seconds with weak (left) and strong (right) action regularization. Both policies reach a similar track position shortly after the split-S; stronger regularization reduces rapid switching and saturation.

http://www.imavs.org/

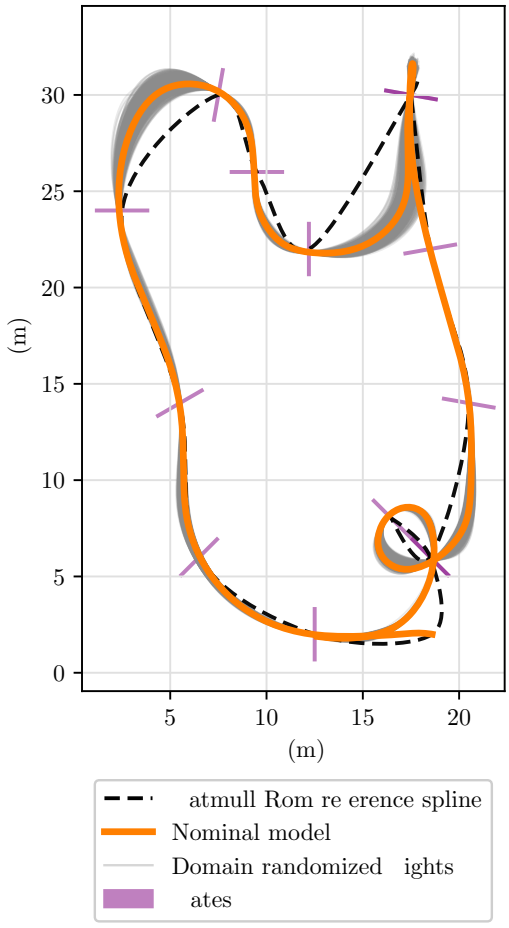


Figure 7: Nominal and domain-randomized fixed-start trajectories, with the reset spline and gate locations.

6 Conclusion

We presented a rapid sim-to-real workflow for autonomous drone racing using a compact body-rate/thrust model identified from short flight logs and massively parallel PPO in JAX. The fastest configuration reaches 32 million environment steps per second on an RTX 5070 Ti, but the 96-setting study and 32-seed analysis show that throughput alone does not determine learning quality under a five-minute budget. A matched comparison visibly suppresses rapid switching and reduces the fraction of commands near saturation, defined by $|a_{t,j}| > 0.95$, from 50.00% to 1.06%. Two selected policies transferred to the real drone, completed uninterrupted two-lap runs, and reached speeds near $20\,\mathrm{m/s}$.

The policy is optimized for a specified track: a new layout requires retraining, and a new platform also requires model re-identification. The current experiments therefore validate rapid adaptation for one vehicle and track, not cross-track or cross-platform generalization. The action comparison also changes the action-delta, action-magnitude, and entropy coefficients together. Future work will separate these effects, study the trade-off between model fidelity and training speed, and repeat the complete workflow across additional tracks and platforms.

References

[1] Hyungpil Moon, Jose Martinez-Carranza, Titus Cieslewski, Matthias Faessler, Davide Falanga, Alessandro Simovic, Davide Scaramuzza, Shuo Li, Michael Ozo, Christophe De Wagter, Guido de Croon, Sunyou Hwang, Sunggoo Jung, Hyunchul Shim, Haeryang Kim, Minhyuk Park, Tsz-Chiu Au, and Si Jung Kim. Challenges and implemented technologies used in autonomous drone racing. *Intelligent Service Robotics*, 12(2):137–148, 2019.

[2] L. Oyuki Rojas-Perez and J. Martinez-Carranza. On-board processing for autonomous drone racing: An overview. *In-*

tegration, 80:46–59, 2021.

[3] Philipp Foehn, Dario Brescianini, Elia Kaufmann, Titus Cieslewski, Mathias Gehrig, Manasi Muglikar, and Davide Scaramuzza. AlphaPilot: Autonomous drone racing. *Autonomous Robots*, 46(1):307–320, 2022.

[4] Drew Hanover, Antonio Loquercio, Leonard Bauersfeld, Angel Romero, Robert Pěnička, Yunlong Song, Giovanni Cioffi, Elia Kaufmann, and Davide Scaramuzza. Autonomous drone racing: A survey. *IEEE Transactions on Robotics*, 40:3044–3067, 2024.

[5] Stavrow A. Bahnam, Robin Ferede, Till M. Blaha, Anton E. Lang, Erin Lucassen, Quentin Missinne, Aderik E. C. Verraest, Christophe De Wagter, and Guido C. H. E. de Croon. MonoRace: Winning champion-level drone racing with robust monocular AI. *arXiv preprint arXiv:2601.15222*, 2026.

[6] Aderik Verraest, Stavrow Bahnam, Robin Ferede, Guido de Croon, and Christophe De Wagter. SkyDreamer: Interpretable end-to-end vision-based drone racing with model-based reinforcement learning. *arXiv preprint arXiv:2510.14783*, 2025.

[7] Elia Kaufmann, Leonard Bauersfeld, Antonio Loquercio, Matthias Müller, Vladlen Koltun, and Davide Scaramuzza. Champion-level drone racing using deep reinforcement learning. *Nature*, 620(7976):982–987, 2023.

[8] Robin Ferede, Till Blaha, Erin Lucassen, Christophe De Wagter, and Guido C. H. E. De Croon. One net to rule them all: Domain randomization in quadcopter racing across different platforms. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6357–6363, 2025.

[9] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[10] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: Composable transformations of Python+NumPy programs, 2018.

[11] Angel Romero, Sihao Sun, Philipp Foehn, and Davide Scaramuzza. Model predictive contouring control for time-optimal quadrotor flight. *IEEE Transactions on Robotics*, 38(6):3340–3356, 2022.

[12] Michal Minařík, Robert Pěnička, Vojtěch Vonášek, and Martin Saska. Model predictive path integral control for agile unmanned aerial vehicles. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 13144–13151, 2024.

[13] Jonas Eschmann, Dario Albani, and Giuseppe Loianno. Learning to fly in seconds. *IEEE Robotics and Automation Letters*, 9(7):6336–6343, 2024.

[14] Jonas Eschmann, Dario Albani, and Giuseppe Loianno. Data-driven system identification of quadrotors subject to motor delays. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 8095–8102, 2024.

[15] Narendran Muraleedharan and Daniel S. Cohen. Modelling and simulation of UAV systems. In *Imaging and Sensing for Unmanned Aircraft Systems. Volume 1: Control and Performance*, pages 101–121. Institution of Engineering and Technology, 2020.

[16] Tim Seyde, Igor Gilitschenski, Wilko Schwarting, Bartolomeo Stellato, Martin Riedmiller, Markus Wulfmeier, and Daniela Rus. Is bang-bang control all you need? solving continuous control with bernoulli policies. In *Advances in Neural Information Processing Systems*, volume 34, pages 27209–27221, 2021.

[17] Siddharth Mysore, Bassel Mabsout, Renato Mancuso, and Kate Saenko. Regularizing action policies for smooth control with reinforcement learning. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1810–1816, 2021.

[18] Joel Klimont, Jakob Buchsteiner, Alexander Lampalzer, and Radu Grosu. Weitflug: A research-oriented fork of Betaflight for agile autonomous flight. In *Proceedings of the 17th International Micro Air Vehicle Conference and Competition*, Strasbourg, France, 2026. To appear.

[19] Abu Dhabi Autonomous Racing League. A2RL drone championship sets the pace for AI in autonomous flight, January 2026. Accessed July 24, 2026.

http://www.imavs.org/