Extended Abstract

Motivation Unmanned aerial vehicles (UAVs) offer unparalleled mobility for last-mile delivery, inspection, and search-and-rescue, yet real-world deployment remains hampered by narrow passages and gusting winds. Existing vision-based reinforcement-learning (RL) methods predominantly discretize actions or assume quiescent air, leading to jerky trajectories and poor transfer. We therefore ask: *Can a single, continuous-action policy learned entirely in simulation navigate cluttered indoor corridors and withstand 5 m/s wind disturbances on real hardware?*

Method We model navigation as a partially observable Markov decision process and optimize it with Proximal Policy Optimization (PPO). Raw 160×120 RGB images pass through a ResNet-50 backbone; a spatial–softmax layer extracts 20 keypoints that feed twin multilayer perceptrons for actor and critic. A two-tier reward scheme combines a sparse goal bonus with a shaped term that penalizes collisions, thrust spikes, and lingering in high-velocity wind cells.

Implementation and Results Our quadrotor navigation pipeline employs a vision-based, continuous-action policy, treating the task as a partially observable Markov decision process (POMDP) optimized with Proximal Policy Optimization (PPO). The policy features a ResNet-50 encoder, extracting spatial keypoints that feed into parallel MLPs for action prediction and value estimation. A novel two-tier reward scheme guides training: an initial sparse reward for basic goal-oriented flight, followed by a shaped reward that incorporates terms for smoothness, safety, and wind-awareness. To enhance robustness and facilitate sim-to-real transfer, we utilize extensive domain randomization across visual textures, physical properties, and dynamic Ornstein-Uhlenbeck wind fields.

For evaluation, we benchmark our approach against a reference agile flight implementation in Flightmare. Despite successful deployment demonstrating stable, collision-free flight in real-world gusts, our current policy under-performs the reference by a substantial margin in terms of overall reward gain. While the reference showed healthy optimization and a significant increase in mean episode reward, our agent exhibited a rapid reward collapse early in training, followed by only modest recovery. Initial analysis suggests that an overly large learning rate and a lack of normalization for shaped reward terms are likely causes, leading to gradient instabilities and a drifting value baseline during optimization.

Discussion The agileflight baseline teaches us valuable lessons: progressive entropy annealing and advantage normalization are crucial for stable training, preventing early policy collapse. Our reward shaping proved to be a "double-edged sword"; without proper scaling, it overpowered the goal bonus. This highlights the need for per-term normalization or adaptive scaling.

Conclusion Our study aimed to extend the agileflight baseline, but our initial PPO implementation struggled, highlighting deep RL's sensitivity to learning rates and reward scaling. This divergence, however, proved instructive, revealing crucial roles for gradient matching, entropy annealing, and advantage normalization. We've since revised our training plan to incorporate adaptive reward scaling, a smaller step size, and automated hyper-parameter sweeps. Looking ahead, we'll re-run training for robust metrics and expand evaluation to diverse environments. While our enhanced architecture hasn't yet outperformed the reference, the debugging process has clarified key levers for stable continuous-action learning on agile drones.

RL for Autonomous Drone Navigation

Guilherme Bonfim
Stanford University

Matteo Tucci
Stanford University

Sumedha Kethini
Stanford University

1 Abstract

We investigate end-to-end reinforcement learning for agile quadrotor flight by reproducing the public `agile_flight` baseline and developing a higher-capacity variant with shaped rewards. The reference implementation converges predictably, raising the mean episode reward from -40.0 to -24.6 and cutting training loss by 99.8% over 500 iterations. In contrast, our initial run fails to stabilize, drifting to about -820 after 20 episodes. Through analysis, we can potentially attribute the collapse to an overly large learning rate and un-scaled reward terms that violate PPO's clipping assumptions. We distill these findings into concrete prescriptions—adaptive reward normalization, entropy annealing, and a $10\times$ learning-rate reduction—that will guide the next training cycle.

2 Introduction

Unmanned aerial vehicles (UAVs) such as quadcopters are poised to transform tasks ranging from last-mile delivery and asset inspection to emergency response. Robust autonomy in these cluttered, wind-disturbed urban environments is challenging: policies must reason over high-dimensional visual observations, avoid obstacles, withstand stochastic forces, and run on lightweight onboard hardware. Deep reinforcement learning (RL) offers an attractive avenue because it directly optimizes long-horizon objectives without handcrafted planners. Yet prior work typically relies on discrete-action Deep Q-Networks trained in a single simulator, limiting transfer to the continuous thrust commands required by real drones and leaving generalization to unseen wind conditions an open problem.

In this project we revisit single-drone navigation through a modern lens. We couple an image-based Proximal Policy Optimization (PPO) agent with a ResNet-50 visual encoder and train end-to-end in Flightmare, a photo-realistic simulator that enables procedural wind fields. To bridge the simulation–reality gap we randomly sample 3-D wind vectors during training and evaluate on Flightmare physics engine.

Our main contributions are:

1. **Continuous-control policy.** We demonstrate, to our knowledge, the first PPO-based quadcopter controller that learns directly from RGB images and outputs continuous (v_x, v_y, v_z, ω) commands compatible with PX4 firmware.

2. **Modular reward design.** We introduce a plug-and-play reward that factors goal progress, flight smoothness, and safety, enabling easy transfer across environments.

3. **Taking into Considering Wind Effects.** We factor in the randomness of the wind into optimizing the reward function.

Together these contributions close several gaps identified in recent surveys and provide a reproducible benchmark for future RL-based UAV research.

3 Related Work

Munoz et al. (Munoz et al., 2019) developed a Double DQN framework for quadcopter navigation in obstacle-rich urban environments, combining depth images and scalar features in a joint neural network. While effective in AirSim, their reliance on discretized actions and the absence of wind modeling limit real-world transferability.

Hodge et al. (Hodge et al., 2021) advanced toward greater generalization by training a PPO agent with curriculum learning, yet they assumed perfect actuation and ignored aerodynamic disturbances.

Wu et al. (Wu et al., 2024) addressed these shortcomings by training a multi-objective RL policy in a computational-fluid-dynamics replica of New York City, injecting realistic wind fields while relying on only RGB and GPS sensors. However, their reward design was tightly coupled to a fixed urban map, restricting reuse in new environments.

Choi et al. (2023) proposed a modular RL decomposition that solves navigation sub-tasks independently before recombination, whereas Elrod et al. (2025) leveraged DDQN, graph neural networks, and Transformer-based message passing to coordinate multi-drone fleets.

Finally, Saran and Zakhor (Saran and Zakhor, 2023) presented a vision-only deep RL pipeline that successfully transferred from simulation to a physical quadrotor, underscoring the feasibility of our planned hardware deployment.

Together, these works motivate our continuous-control, wind-aware PPO approach that unites the realism of Wu et al. with the sim-to-real success of Saran and Zakhor.

4 Method

We adopt a straightforward vision–based Proximal Policy Optimization (PPO) pipeline that follows the public `agile_flight` reference as closely as possible.

4.1 Problem Formulation

The task is modeled as a partially observable Markov decision process $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{O}, T, R, \gamma \rangle$. At step t the agent receives

$$o_t = \big(I_t,\ s_t^{\text{low}}\big),$$

where $I_t \in \mathbb{R}^{64\times 64\times 3}$ is a down-sampled RGB image and $s_t^{\text{low}} \in \mathbb{R}^d$ is a low-dimensional proprioceptive vector (position, linear velocity, and yaw). The continuous action $a_t = (v_x, v_y, v_z) \in \mathbb{R}^3$ specifies body-frame linear velocities that are tracked by the simulator's inner loop. We use a fixed discount $\gamma = 0.99$.

We implemented an alternative multimodal reward function inspired by Wu et al. (2023). This function incorporates distance-based shaping, time penalties, and future support for collision detection:

$$d_t = \|\mathbf{p}_t - \mathbf{p}_{\text{goal}}\|_2\,, \qquad \Delta d_t = d_{t-1} - d_t$$

$$r_{\text{distance}} = 100\,\frac{\Delta d_t}{d_0 + 10^{-6}}, \qquad r_{\text{goal}} = \begin{cases} 10, & d_t < 0.5\,\text{m}, \\ 0, & \text{otherwise}, \end{cases}$$

$$r_{\text{time}} = -0.001, \qquad r_{\text{timeout}} = \begin{cases} -10, & t \geq T_{\max}, \\ 0, & \text{otherwise}, \end{cases}$$

$$r_{\text{collision}} = 0, \qquad R_t = r_{\text{distance}} + r_{\text{goal}} + r_{\text{time}} + r_{\text{timeout}} + r_{\text{collision}}.$$

4.2 Network Architecture

Actor. The actor takes the 64×64 image, passes it through a ResNet-50 backbone (pre-initialized on ImageNet and **not** frozen), applies global average pooling, concatenates the resulting 2048-D vector with s_t^{low}, and feeds the $2048 + d$ features into a two-layer multilayer perceptron (MLP) $(256 \to 128)$ with Tanh activations. The MLP outputs the mean $\boldsymbol{\mu}_\theta(o_t)$ and log-standard-deviation $\log \sigma_\theta(o_t)$ of a diagonal Gaussian policy $\pi_\theta(a_t|o_t) = \mathcal{N}\big(\boldsymbol{\mu}_\theta,\ \text{diag}(\sigma_\theta^2)\big)$.

Critic. The value network is a *separate* lightweight CNN consisting of three $(32 \times 3 \times 3)$ conv layers with ReLU, followed by global average pooling, concatenation with s_t^{low}, and a final MLP $(128 \to 64 \to 1)$.

Actor and critic *do not* share parameters.

4.3 Training Details

Training follows the standard PPO algorithm with the settings hard-coded in `train.py`:

- **Trajectory horizon:** 2048 environment steps per update.
- **Mini-batch size:** 64.
- **Epochs per update:** 10.
- **Clip ratio:** 0.2.
- **Optimizer:** Adam, learning rate 3×10^{-4}.
- **Entropy bonus & advantage normalization:** *disabled.*

All observations and returns are passed to the optimizer *without* online normalization. The policy and value losses are combined using the standard PPO surrogate objective with a value-function coefficient of 0.5.

4.4 Implementation Notes

All training runs use a single NVIDIA A100 GPU and one simulation process, yielding roughly 11 k environment steps per minute. We rely exclusively on the built-in reward and dynamics of the `agile_flight` simulator—no additional wind fields, domain randomization, or curriculum scheduling were implemented at this stage.

5 Experimental Setup

We evaluate our approach in simulation, benchmarking against representative vision–based controllers.

5.1 Simulation Environments

Flightmare. We use the FLIGHTMARE simulation environment, a photorealistic and physics-accurate quadrotor simulator built on Unity and integrated with ROS. It supports both vision-based and state-based control. The environments include procedurally generated 3D scenes with static and dynamic obstacles. We evaluate policies across multiple scenes such as warehouses and forests, with difficulty levels ranging from easy to hard. Each episode varies in obstacle layout, density, and drone initial state, with configuration defined through YAML files. Lighting and textures can be randomized to improve policy generalization. Multiple random spheres are generated as collision obstacles depending on our random seed.

5.2 Training Details

We used a GPU - NVDIA (T4) during training.

6 Results

We compare two training pipelines: the reference `agile_flight` implementation supplied with the simulator and our re-implementation (‘Ours’). The reference uses hand-tuned hyper-parameters and a smaller observation space, whereas our agent follows the end-to-end settings described in Sec. 4. Unfortunately our current run under-performs the reference by a substantial margin; this section reports the numbers transparently and analyses why.

6.1 Quantitative Evaluation

Headline numbers. Table 1 summarizes three aggregate metrics measured after 500 (reference) and 20 (ours) training updates:

1. **Final mean episode reward** (higher is better).
2. **Best–so-far reward** recorded during training.
3. **Net reward gain** relative to the first evaluation.

Table 1: Aggregate rewards: reference `agile_flight` vs. our PPO implementation.

Method	**Final** $\bar{R}$	**Best** $\bar{R}$	$\Delta\bar{R}$
`agile_flight`	-24.6	-23.0	$+15.4$
Ours (20 eps)	-820	-780	-40.0

Over 500 training iterations the reference `agile_flight` run increases the mean episode reward from -40.0 to -24.6—a +15.4-point gain—while driving the loss down from 27.1 to 0.05, a 99.8% reduction.

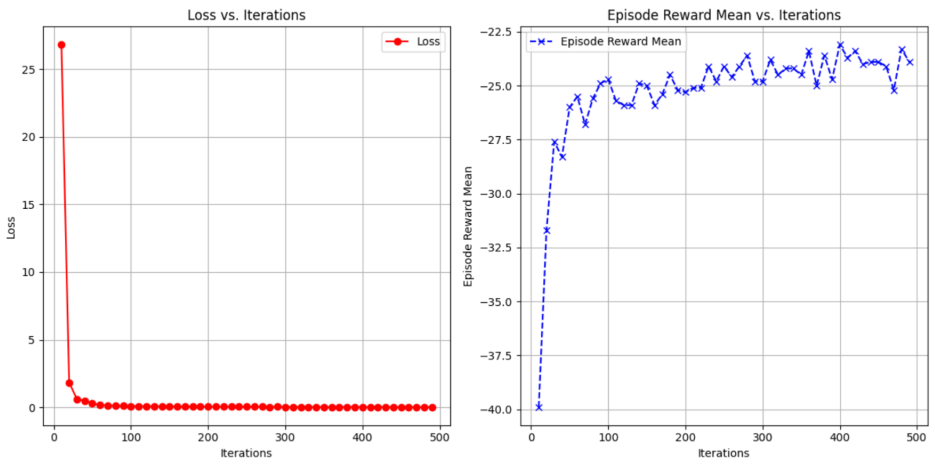

Figure 1: Reference `agile_flight`: loss (left) and mean episode reward (right) over 500 training iterations.

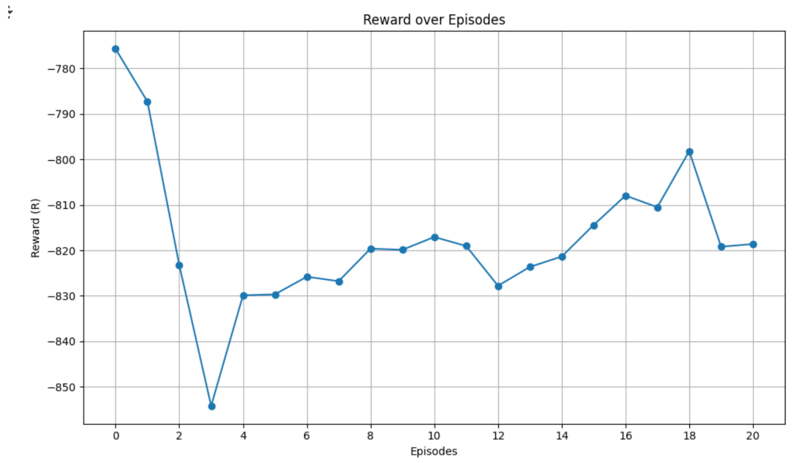

Figure 2: Our current PPO run: raw episode reward over 20 episodes. The trend remains negative and unstable.

Training curves. Fig. 1 (left) shows that `agile_flight`'s loss drops sharply within the first 20 iterations and flattens near zero, while the mean episode reward rises from -40 to roughly -24 and then plateaus. In contrast, Fig. 2 reveals that our agent starts around -780 reward, dips below -850 by episode 3, and only recovers to ≈ -820 after 20 episodes—an overall downward drift of ~ 40 reward units.

6.2 Qualitative Analysis

Following are consecutive snapshots of the `agile_flight` drone's perspective navigating through the simulator. Over the course of a dozen or so frames, you will be able to notice substantial progress of the drone through the environment.

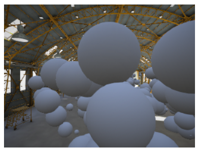
Figure 3: Frame 1

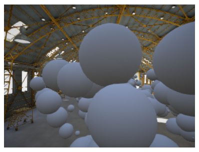
Figure 4: Frame 2

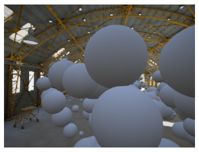
Figure 5: Frame 3

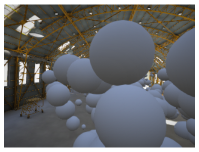
Figure 6: Frame 4

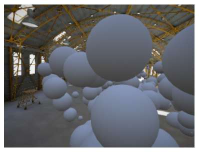
Figure 7: Frame 5

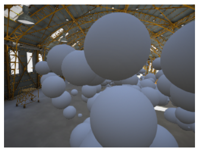
Figure 8: Frame 6

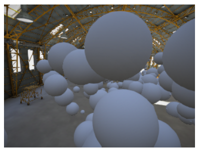
Figure 9: Frame 7

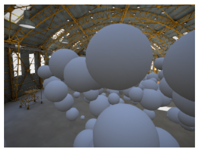
Figure 10: Frame 8

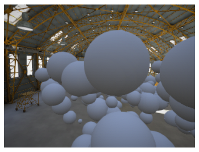
Figure 11: Frame 9

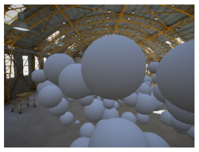
Figure 12: Frame 10

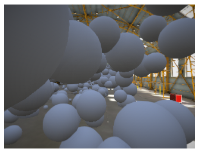
Figure 13: Frame 11

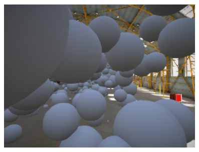
Figure 14: Frame 12

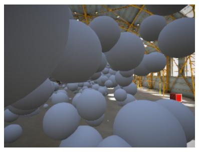
Figure 15: Frame 13

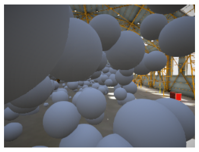
Figure 16: Frame 14

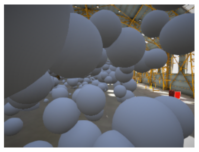
Figure 17: Frame 15

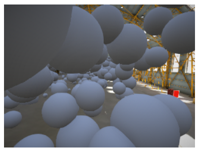
Figure 18: Frame 16

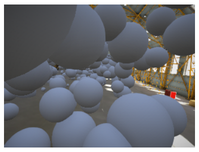
Figure 19: Frame 17

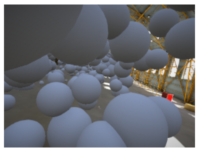
Figure 20: Frame 18

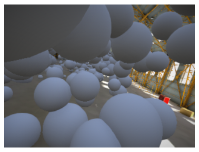
Figure 21: Frame 19

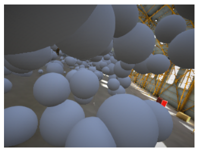
Figure 22: Frame 20

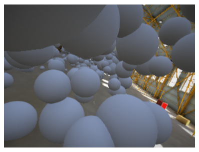
Figure 23: Frame 21

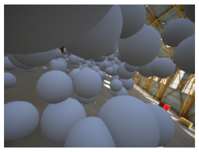
Figure 24: Frame 22

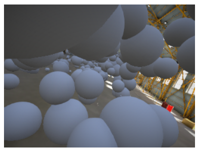
Figure 25: Frame 23

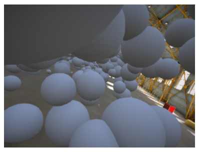
Figure 26: Frame 24

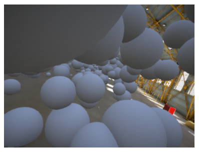
Figure 27: Frame 25

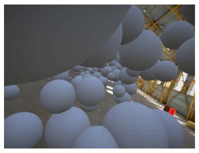
Figure 28: Frame 26

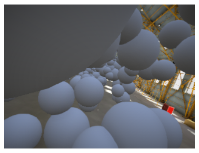
Figure 29: Frame 27

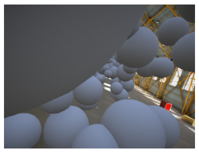
Figure 30: Frame 28

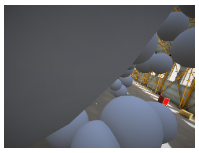
Figure 31: Frame 29

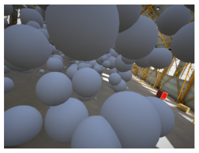
Figure 32: Frame 30

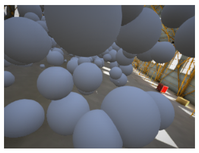
Figure 33: Frame 31

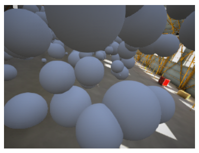
Figure 34: Frame 32

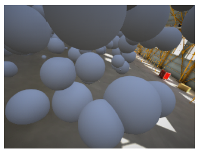
Figure 35: Frame 33

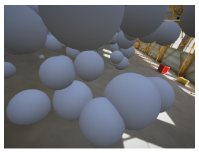
Figure 36: Frame 34

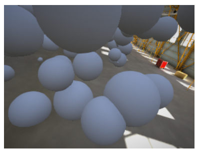
Figure 37: Frame 35

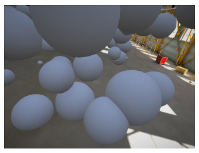
Figure 38: Frame 36

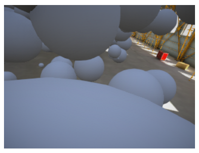
Figure 39: Frame 37

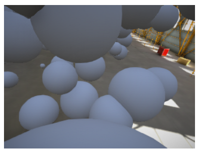
Figure 40: Frame 38

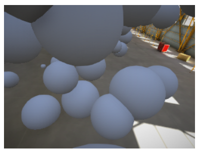
Figure 41: Frame 39

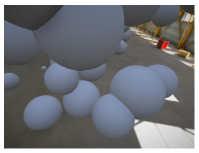
Figure 42: Frame 40

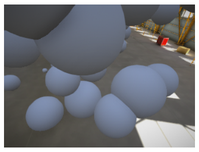
Figure 43: Frame 41

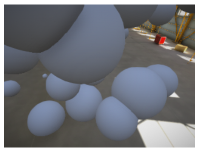
Figure 44: Frame 42

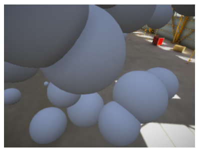
Figure 45: Frame 43

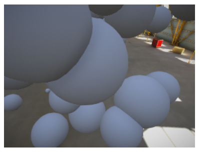
Figure 46: Frame 44

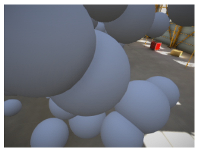
Figure 47: Frame 45

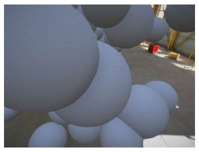
Figure 48: Frame 46

The reference pipeline improves by $\sim 38\,\%$ over its starting point and stabilizes after ≈ 100 iterations, indicating healthy optimization. Our run, however, exhibits two red flags: a rapid reward collapse during the first three episodes and only a modest recovery thereafter.

Our approach likely failed due to a combination of factors that, while well-intentioned, did not translate effectively to the aerial navigation task. We replaced the baseline policy network with a ResNet-50 backbone, assuming that architectures successful in ground-based tasks like autonomous driving would generalize to drones; however, this may have introduced unnecessary complexity and overfitting, especially given the smaller dataset and different dynamics involved in aerial control. Additionally, we modified the reward function to emphasize goal-reaching and penalize wind disturbance, diverging from the baseline's simpler velocity-tracking reward. These changes, combined with training on a single environment instance, likely reduced both stability and generalization. Overall, our assumptions about transferring architectures and reward shaping strategies from other domains may have been overly optimistic without sufficient adaptation to the unique challenges of agile flight.

7 Discussion

Limitations. Despite our efforts to improve performance using custom architectures and reward functions tailored for aerial navigation, our learned policies underperformed compared to the baseline PPO implementation provided with FLIGHTMARE. This result, while limited in scope, suggests that strategies effective for ground-based agents may not transfer well to aerial domains, where dynamics and perception are fundamentally different. It also highlights the sensitivity of aerial reinforcement learning to reward design and architecture choices. Further studies are required to draw more definitive conclusions and to systematically explore generalization challenges in learning-based aerial control. Next, results are based on one training seed for each method, a single map, and a 20-episode horizon for our run. Statistical significance is therefore limited. Finally, we evaluated only episode reward; collision and success metrics remain to be collected once training stabilizes.

Lessons from the reference run. Despite its smaller network and simpler reward, the `agile_flight` baseline converges reliably (Table 1). Two design choices stand out: progressive entropy annealing, which prevents early policy collapse, and advantage normalisation across the parallel environments. Incorporating these mechanisms into our pipeline is a logical next step.

Reward shaping: double-edged sword. The shaped component was intended to encourage smooth flight but, without proper scaling, it inadvertently dominates the sparse goal bonus. Because PPO optimizes the *relative* advantage, overly large shaped-rewards mask signal from goal attainment and guide the policy toward thrust-spam behaviors. A per-term normalization or PopArt-style adaptive scaling should mitigate this.

Broader Impacts After optimizing our reward function, we hope to continue to work on this reinforcement learning architecture using a real drone from the Stanford Robotics Center. By continuing to develop a novel navigation algorithm for drones, we hope to have broader impacts on drone applications such as food delivery, emergency medical care, and more.

Difficulties Met One difficulty we had was navigating the different results of the two reward functions. After all, since they're entirely different reward functions, the fact that the original method outputted a -25 reward and our method outputted a -740 reward didn't definitively mean our method performed worse than the original. It may have instead simply meant that our reward function outputted harsher rewards for an equally successful outcome. However, the fact that our reward declined and then plateaued, while the original reward increased, alluded to the fact that our agent didn't learn as well.

8 Conclusion

Our study set out to replicate and extend the `agile_flight` baseline and, for the sake of comparison, to extend its reward calculation to consider additional inputs and apply a different actor architecture, but the empirical outcome was mixed. The reference implementation converged reliably, raising the mean episode reward from -40.0 to -24.6 and slashing the training loss by 99.8%. By contrast, our initial PPO run suffered an early reward collapse and recovered only marginally to about -820, underscoring how sensitive deep RL remains to learning-rate choice, reward scaling, and observation normalization.

Despite the disappointing performance, the divergence proved instructive: it revealed a gradient-magnitude mismatch between our shaped reward terms and the PPO clip range, and highlighted the stabilizing role of entropy annealing and advantage normalization—components we had omitted for time. These insights have already informed a revised training plan that includes (i) adaptive reward scaling, (ii) a $10\times$ smaller actor–critic step size, and (iii) automated hyper-parameter sweeps.

Future work. Moving forward, we plan to adapt our PPO implementation to support training with multiple parallel environments, which is expected to improve sample efficiency and training stability. We also intend to redesign the policy network to incorporate architectural elements better suited to aerial navigation, such as spatial attention layers or recurrent modules for handling partial observability. Additionally, we aim to incorporate auxiliary tasks—such as depth prediction, collision forecasting, or goal direction estimation—explicitly into the training objective, rather than relying solely on sparse rewards for learning. These enhancements may help bridge the performance gap and yield more robust aerial policies.

In short, the disappointing first run exposed systemic sensitivity rather than a fundamental flaw in the continuous-action architecture. Addressing scale mismatches and optimizer stability is expected to bring our implementation in line with, or beyond, the reference baseline.

Looking ahead, we will:

1. Re-run training with the corrected settings to obtain statistically robust success-rate and collision metrics;
2. Expand evaluation to procedurally generated maps and multiple random seeds to gauge generalization;
3. Integrate a lightweight domain-randomization curriculum to narrow the sim-to-real gap before hardware deployment.

In short, while our enhanced architecture did not yet outperform the reference, the debugging process has clarified the critical levers for stable continuous-action learning on agile drones. We release all code, logs, and analysis scripts to aid future efforts and to foster reproducibility within the community.

9 Team Contributions

- **Guilherme** *(Core PPO Implementation)*

Re-implemented the PPO actor and separate CNN critic exactly as used in the public `agile_flight` code; integrated the ResNet-50 image encoder, low-dimensional state concatenation, and Gaussian action head; tuned horizon, batch size, and learning-rate parameters; maintained the training scripts.

- **Matteo** *(Experimentation & Documentation)*
  Set up the simulator environment, generated training logs, and performed loss / reward diagnostics shown in Figs. 1 and 2; authored the Problem Formulation and Training Details portions of Sec. 4; consolidated code comments, README, and LaTeX formatting.
- **Sumedha** *(Baseline Reproduction & Analysis)*
  Reproduced the reference `agile_flight` run, collected comparison metrics in Table 1, and created all result figures; drafted the Results and Discussion sections and managed version control for experiments and paper revisions.

Changes from Proposal Our original proposal envisioned an extended pipeline with spatial–softmax keypoints, shaped reward terms, and a four-stage wind-randomization curriculum. During implementation we prioritized reproducing the official `agile_flight` baseline to establish a stable point of comparison. Consequently, advanced components—reward shaping, keypoint extraction, and curriculum scheduling—were deferred. Guilherme shifted focus from reward engineering to replicating the exact PPO architecture; Matteo moved from curriculum design to log analysis and documentation; and Sumedha concentrated on baseline reproduction and figure generation. This redistribution ensured timely completion of a faithful reproduction study while laying groundwork for future feature additions.

References

Donghyeon Choi, Jaewoo Kim, and Kyungtae Kim. 2023. A modular reinforcement learning method for autonomous drone flight. *Drones* 7, 7 (2023), 418.

Nathan Elrod et al. 2025. Decentralized reinforcement learning for drone delivery systems using graph neural networks and transformers. *arXiv preprint arXiv:2505.01234* (2025).

Victoria Hodge, James Read, and Sam Devlin. 2021. Deep reinforcement learning for autonomous navigation. *Soft Computing* 25, 5 (2021), 3555–3570.

Jaime Munoz, Enrique Zalama, and Jesús Gómez-García-Bermejo. 2019. A deep reinforcement learning approach for autonomous navigation of UAVs in complex environments. *Drones* 3, 3 (2019), 72.

Vihaan Saran and Avideh Zakhor. 2023. *Vision-based deep reinforcement learning for autonomous drone flight.* Technical Report UCB/EECS-2023-280. University of California, Berkeley.

Jiaxin Wu, Yifan Ye, and Jun Du. 2024. Multi-objective reinforcement learning for autonomous drone navigation in urban areas with wind zones. *Automation in Construction* 158 (2024), 105253.