Extended Abstract

Motivation Autonomous drone navigation in obstacle-rich environments is essential for real world applications like package delivery, search and rescue, and inspection tasks. Collisions with unexpected obstacles pose a significant challenge to achieving reliable performance in these scenarios. The goal of this project is to train a single autonomous quadrotor (modeled as a Crazyflie 2.0×) to navigate from a fixed start point to a fixed goal at (5, 5, 1) while avoiding 3 static cube obstacles in the PyBullet simulator. My personal motivation for this project stems from my collaboration with my brother to build an autonomous rescue drone system to deliver emergency medical assistance to help deliver medication to cardiac arrest patients when they are home alone.

Method To address the challenge of sparse rewards in continuous action spaces, I implemented several reward shaping strategies designed to guide the learning process. The initial reward function was sparse: $+100$ for reaching the goal and ˇ100 for crashes. I then iteratively experimented with shaping variants, including small alive bonuses, potential-based shaping terms $(+\Delta d)$, and velocity-alignment bonuses conditioned on how close the obstacle was. Additionally, I incorporated a time penalty term to discourage passive hovering. To make the training more time-efficient, I used curriculum learning by starting training with easier goal positions and progressively increasing the difficulty to the final goal at $(5, 5, 1)$. I utilized Soft Actor-Critic (SAC) as the main reinforcement learning algorithm, taking advantage of its off-policy capabilities and entropy-based exploration to manage sparse rewards. The state vector included position, velocity, angular velocity, and a 3D goal-relative offset to inform the agent of its target location.

Implementation The training environment was built using the gym-pybullet-drones simulator. Each experiment ran for 1 million steps per shaping variant, and evaluation was performed across 100 episodes. Metrics included average episode reward, average episode length, and success rate. My implementation pipeline included automatic logging of training curves (ep_rew_mean and ep_len_mean) and TensorBoard integration for real-time monitoring. I tuned the reward shaping terms iteratively, guided by observed training instabilities and reward magnitudes.

Results Experiments with purely sparse rewards failed to produce meaningful learning, with PPO and SAC agents either hovering until timeout or crashing quickly. SAC initially showed promise with sparse rewards qualitatively but ended up with a low success rate (0%). Incorporating alive bonuses or potential-based shaping on their own was insufficient as agents either prioritized alive time or exhibited unstable reward spikes. The best performance (32% success rate) came from a carefully balanced reward function combining distance bonuses, potential-based shaping, and conditional velocity-alignment bonuses. Even this best variant showed slow, incremental improvements, with frequent failures. Quantitative metrics revealed that while the best shaping variant significantly outperformed the sparse baseline, absolute success rates remained pretty low due to the inherent difficulty of the task.

Discussion My results highlight the importance of combining multiple shaping signals to guide exploration in sparse-reward, continuous-action tasks. Reward shaping and curriculum learning proved essential, but not sufficient on their own to fully solve the obstacle avoidance problem. Overly strong shaping terms could destabilize the critic or overshadow the goal bonus. Too small crash penalties failed to deter reckless policies, while overly large penalties caused divergence in the value estimates. Curriculum learning helped mitigate gradient instability, while the goal-relative offset in the observation was crucial to provide directional cues. These findings emphasize the challenge of balancing exploration and exploitation in complex RL environments.

Conclusion This project demonstrated that standard RL methods such as PPO and SAC both struggle in sparse-reward, continuous obstacle-avoidance tasks without reward shaping and curriculum learning. My crafted reward functions, combined with curriculum learning, improved performance significantly but still fell short of fully solving the task. This highlights the need for continued research into potentially a adaptive shaping strategy or hybrid model-based planning to achieve safe and reliable obstacle avoiding flight. This project was a small step towards developing autonomous rescue drones capable of delivering medical assistance in real-world home environments, the ultimate goal I aim to achieve with my brother.

Learning Obstacle-Avoiding Drone Navigation with PPO and SAC

Claire Du
Department of Mechanical Engineering
Stanford University
clairedu@stanford.edu

Abstract

In this project, I investigate the problem of training a single autonomous drone, specifically a Crazyflie 2.0×, to navigate from a fixed starting position at to a goal position at while avoiding 3 diagonal obstacles in a simulated environment built using the `gym-pybullet-drones` framework. The problem is challenging due to the sparse nature of the reward signal and the continuous action space, which makes exploration and learning particularly difficult. Throughout this project, I tested Proximal Policy Optimization (PPO) and Soft Actor-Critic (SAC) algorithms. To improve learning, I designed several reward-shaping strategies, including potential-based shaping, alive bonuses, conditional velocity-alignment, and a time penalty term. These were combined to provide more frequent learning signals and encourage safe, directed progress toward the goal. While training metrics showed improvement with shaped rewards, evaluation results remained disappointing, with the agent still failing to achieve successful navigation in any episode. These results underscore the challenge of applying reinforcement learning to sparse-reward, obstacle-avoidance tasks with continuous actions. Despite extensive reward shaping, the agent could not generalize to safe and reliable navigation.

1 Introduction

Autonomous drone navigation in cluttered environments is an essential capability for real-world applications such as package delivery, search and rescue, and infrastructure inspection. My personal motivation for this project is that my brother and I are building an autonomous drone system that can deliver medication to cardiac arrest patients when they are home alone. This system aims to revolutionize emergency medical response by enabling autonomous drones to navigate complex indoor environments and deliver life-saving medication and equipment directly to patients before paramedics arrive. Overall, in all these scenarios, drones must efficiently reach specified goals while avoiding obstacles that could lead to mission failure or damage. In this project, I focus on the challenge of training a single autonomous drone, modeled as a Crazyflie 2.0×, to navigate from a fixed start position at $(0, 0, 1)$ to a fixed goal position at $(5, 5, 1)$ in a simulated environment built using the `gym-pybullet-drones` framework. The environment contains three obstacles placed at $(1.5, 1.5, 1)$, $(2.5, 2.5, 1)$, and $(3.5, 3.5, 1)$, forming a diagonal barrier that the drone must avoid.

Objectives. The primary objective of this project was to train a reinforcement learning (RL) agent capable of successfully navigating to the goal while avoiding the obstacles. The project goal is to evaluate the performance of different RL algorithms (PPO and SAC) in this environment and to investigate whether reward shaping techniques could improve learning efficiency and policy performance. Ultimately, the final goal was to design a reward function and training strategy that enables the agent to consistently reach the goal without collisions.

Research Questions. To address the objectives, the project was guided by the following research questions:

1. Can a standard on-policy RL method such as PPO learn to navigate this environment effectively, given the sparse reward structure?
2. Does an off-policy algorithm like SAC, which benefits from entropy-based exploration, perform better in this sparse-reward, continuous-action setting?
3. How does reward shaping influence learning outcomes in this task and what types of shaping terms are most effective in encouraging obstacle avoidance and goal-reaching behavior?

2 Related Work

Prior research in reinforcement learning for drones has explored navigation in structured or simplified environments. Hwangbo et al. (2017) and Saleh et al. (2018) focus on open-air flight or predefined racing gates, where drones avoid minimal obstacles. These approaches assume a clear path and do not account for static obstacles that appear in cluttered spaces in the real world. My project differs by placing static obstacles in the drone path, which requires learning about true obstacle avoidance from scratch.

Obstacle avoidance has often been addressed using model-based controllers or explicit perception modules. For example, Liu et al. (2019) and Panerati et al. (2021) integrate collision detection through model-predictive control or vision CNNs. While these methods enhance safety, they rely on handcrafted perception modules and external sensors, limiting generalizability. My work, by contrast, uses no explicit collision detectors and trains an end-to-end RL agent that learns purely from continuous state vectors in a PyBullet environment.

Sparse-reward RL research, like Zhao et al. (2021), tackles exploration challenges using count-based exploration bonuses in discretized occupancy grids. However, this approach doesn't translate well to continuous-action environments with high dimensional physics, such as the one I use. My project maintains continuous state and action spaces throughout training, avoiding discretization.

More recent works attempt to handle obstacle-rich environments with various techniques. Zhang (2022) employ TD3 with relative obstacle positions as features, but this requires the environment to feed explicit obstacle information, whereas my drone learns from raw state vectors only. Sheng (2024) propose dynamic reward shaping in dense, moving obstacle scenarios, but rely on extensive LiDAR data and dynamic obstacle models. Kalidas (2023) explore vision-based navigation with SAC, introducing complex perception pipelines that are not used in my work. Xi (2024) combine artificial potential fields with DRL to prevent oscillations but incorporate model-based heuristics that reduce end-to-end learning. Finally, Xu (2024) propose NavRL with a safety shield, but this again adds an external module to enforce safety, while my approach trains a single DRL agent from scratch.

In summary, my project is different because I am training a single autonomous Crazyflie drone in a continuous-action PyBullet environment with static obstacles without any explicit perception modules or handcrafted collision checks. This isolates the challenge of reward shaping and policy learning, highlighting the fundamental difficulty of sparse-reward obstacle avoidance in continuous spaces.

3 Method

My method is based on the following assumptions:

- The drone is modeled in `gym-pybullet-drones` (Panerati et al., 2021) with continuous state and action spaces.
- All observations are direct state vectors (position, velocity, orientation) and no additional perception modules such as vision are used.
- Rewards are carefully shaped but potential-based shaping terms preserve policy invariance.
- Each episode terminates when it reaches goal, crashes, or times out (with max 1000 steps).

To reach my goals, I formally modeled it as a continuous Markov Decision Process (MDP):

$$(s_t, a_t) \rightarrow s_{t+1}, r_t$$

where:

- $s_t \in \mathbb{R}^n$: The state vector from the simulator, including drone position, velocity, orientation, and goal information.
- $a_t \in \mathbb{R}^4$: The continuous motor commands (thrust, pitch, roll, yaw).
- r_t: The reward function (defined below).

Baseline: PPO I initially chose Proximal Policy Optimization (PPO) due to its widespread use in continuous control tasks and its relative robustness to hyperparameters. PPO uses a clipped surrogate objective to ensure stable policy updates:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[\min \left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right]$$

where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ and $\hat{A}_t$ is the advantage estimate. The parameters that I used were:

- Learning rate: 3×10^{-4}
- Batch size: 256
- Entropy coefficient: 0.01

Off-Policy Alternative: SAC Due to PPO's poor performance in sparse-reward, obstacle-rich settings (agents hovered or crashed immediately), I switched to Soft Actor-Critic (SAC). SAC optimizes a maximum entropy objective:

$$J(\pi) = \sum_t \mathbb{E}_{(s_t,a_t)} \left[r(s_t, a_t) + \alpha \mathcal{H}(\pi(\cdot|s_t)) \right]$$

where α balances exploration (entropy term) and exploitation. This allows SAC to explore more effectively than PPO, which is crucial in sparse-reward settings. The parameters I used were:

```
model = SAC("MlpPolicy", env, verbose=1, tensorboard_log=LOG_DIR,
            learning_rate=1e-4, buffer_size=1e6, batch_size=512,
            gamma=0.99, tau=0.005, ent_coef='auto')
model.learn(total_timesteps=500_000)
```

4 Experimental Setup

4.1 Environment Wrapper and Normalization

The environment uses PyBullet physics to simulate the Crazyflie 2.0× quadrotor (Figure 1). Three static cube obstacles are placed at $(1.5, 1.5, 1)$, $(2.5, 2.5, 1)$, and $(3.5, 3.5, 1)$. The state vector includes:

- Position (x, y, z)
- Linear velocity (v_x, v_y, v_z)
- Angular velocity $(\omega_x, \omega_y, \omega_z)$
- Distance to goal vector (dx, dy, dz)

Then overall, I wrapped the environment with:

```
env = DummyVecEnv([make_env])
env = VecNormalize(env, norm_obs=True, norm_reward=True, clip_obs=10.)
```

to stabilize learning with reward normalization.

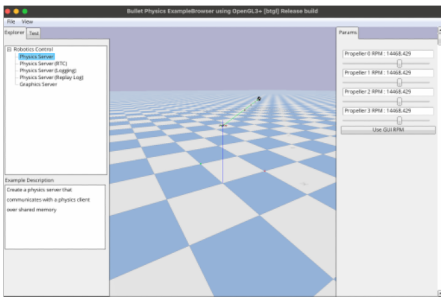


Figure 1: Simulated Environment using gym-pybullet-drones

4.2 Baseline and Reward Shaping Variants

I began with a sparse reward baseline:

- $+100$ on reaching the goal ($d < 0.3\,\text{m}$)
- -100 on crashing (collision with a cube or falling below $z < 0.1$)
- 0 at all other steps

4.3 Reward Shaping Experiments

To address exploration and reward sparsity, I systematically experimented with 5 reward shaping variants (Table 1), motivated by my earlier observations that dense feedback is essential in obstacle-rich environments:

Table 1: Systematic comparison of reward shaping variants with evaluation metrics

Reward Shaping Type	**Description**
Sparse Only	$+100$ (goal); -100 (crash)
+ Alive Bonus	$+0.1$ per step; $+2\Delta d$; $+10$ (goal); -10 (crash)
Potential Based	-0.01 per step; $+5\Delta d$; $+20$ (goal); -10 (crash)
Dist. + Potential-shaping + Velocity alignment	$-d$; $+5\Delta d$; $+2$ (vel.dir); $+100$ (goal); -50 (crash)
Time Penalty + Potential-shaping + Velocity Align	$-$TIME_PENALTY per step; $+3\max(\Delta d, 0)$; $+1$ (vel.dir); $+200$ (goal); -500 (crash)

4.4 Evaluation Metrics

I evaluated all agents on 100 rollouts (N=100) and outputted:

- ep_len_mean: Average episode length, indicating stability and exploration.
- ep_rew_mean: Average episodic reward, assessing learning efficiency.
- Success Rate: Percentage of episodes where the drone reaches the goal.
- Qualitative Behavior: Visual inspection of trajectories to determine whether the drone actually navigates around obstacles or immediately crashes.

This setup ensured a rigorous comparison of the effect of each reward shaping strategy on the learning dynamics of each algorithm.

5 Results

5.1 Experiment 1: Can a Standard On-Policy Method (PPO) Learn This Task?

I initially selected PPO as my baseline algorithm because of its reputation as a stable, on-policy method in continuous-control tasks, widely used in robotics literature (Hwangbo et al., 2017; Saleh

et al., 2018). I implemented PPO with a sparse reward function: +100 on goal reached ($d < 0.3$ m), -100 on crashing (colliding with a cube or descending below $z < 0.1$), and 0 otherwise.

As shown in Figure 2, PPO struggled to make progress. The `ep_len_mean` hovered around 500-550 steps, indicating that the agent never got any positive feedback unless it actually touched the goal (which almost never happened), it learned a "hover-in-place" or "slow-descent" policy that delayed crashing for as long as possible. Furthermore, in my setup, gravity eventually pulls the drone down if no forward motion occurs. Rather than aggressively exploring, PPO found that minimal throttle/tilt lets it stay aloft longer than flying at all. I played around with the reward function by adding an alive bonus to see if I could give additional rewards for moving closer to the end goal. In practice, because of small fluctuations (hovering slightly off-center increases distance a bit), the cumulative per-step "–d" term averaged closer to –18 points/step at times, so total episode reward ended up around –10000 (instead of just –2750). Thus the logged `ep_rew_mean` stayed near –10 000. The high variance and lack of an upward trend revealed that PPO was unable to solve the task: it learned to survive for a while but failed to reach the goal, highlighting the limitations of sparse rewards in continuous, obstacle-rich environments.

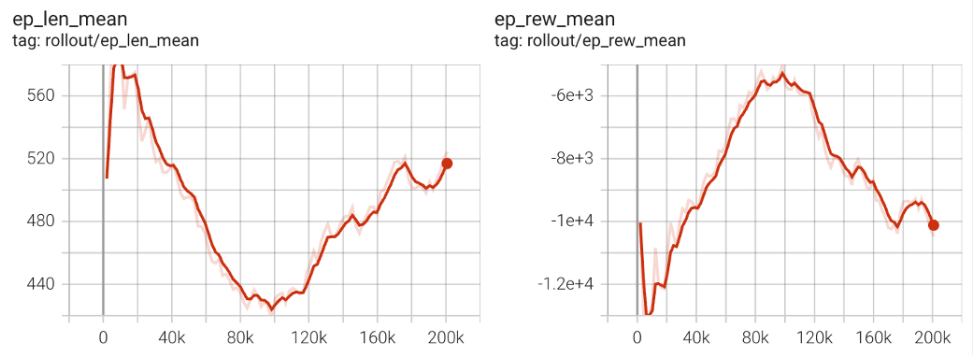


Figure 2: Training Metrics `ep_rew_mean` and `ep_len_mean` using PPO Baseline

5.2 Experiment 2: Can an Off-Policy Method (SAC) Do Better with the Same Sparse Reward?

Since PPO uses on-policy updates and a conservative clipping mechanism, it tends to limit the policy's ability to explore new trajectories once it finds a local minimum (like standing still or slow drifting). Therefore, I decided to use SAC as it uses entropy regularization to encourage exploration by maximizing both the expected reward and the policy's entropy This means the agent tries more diverse actions rather than prematurely exploiting sub-optimal behaviors (like hovering).

I then implemented Soft Actor-Critic (SAC) with default hyperparameters. Like PPO, the reward function was sparse: +100 for reaching the goal, -100 for crashing, and 0 otherwise.

Figure 3 illustrates SAC's performance. Initially, `ep_len_mean` climbed to approximately 600 steps by 40k steps of training, showing more exploratory behavior than PPO but then plateaued. The `ep_rew_mean` dipped as low as -1200 around 100k steps before slowly climbing back to around -200 at 500k steps. Despite this improvement in survival time, SAC still failed to consistently reach the goal. Visual observation confirmed that the agent hovered, sometimes exploring, but ultimately crashed before reaching the target. This aligns with SAC's entropy-driven exploration: initially beneficial for diversity, but insufficient without denser feedback. Even though SAC initially outperformed PPO with pure sparse signals (because its off-policy buffer and entropy allowed a few random successes), it still failed to learn a robust obstacle-avoiding flight. To me, this meant that the drone hovers until crash around step 600 and occasionally gets lucky when it crashes close to the end goal.

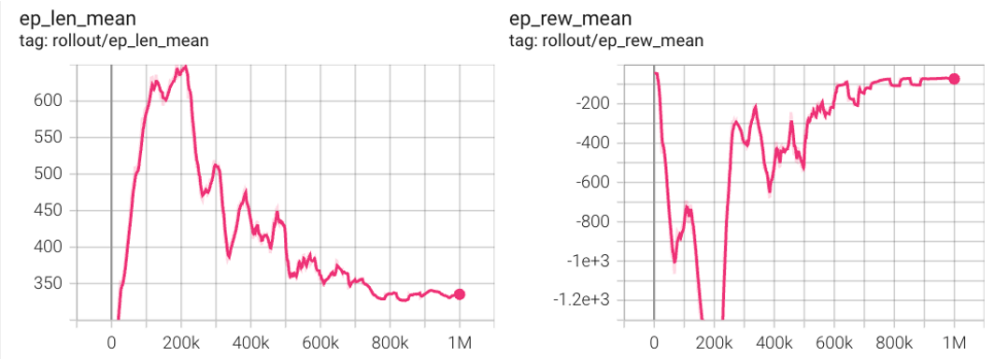


Figure 3: Training Metrics `ep_rew_mean` and `ep_len_mean` using SAC

5.3 Experiment 3: Reward Shaping and Hyperparameter Experiments

In this stage, I systematically experimented with different reward shaping strategies to encourage the drone to learn obstacle-avoiding navigation with Soft Actor-Critic (SAC). For each variant, I trained the agent for 1 million steps and evaluated its performance over 100 episodes.

Initially, I implemented a per-step alive bonus of $+10$ in addition to the sparse terminal rewards (± 100). My intention was to prevent immediate crashes by rewarding the agent simply for staying airborne. However, this alive bonus dominated the learning signal, overshadowing the goal and crash rewards. As a result, the agent learned to hover in place or even terminate the episode prematurely to secure the maximum cumulative reward without actually navigating toward the goal.

To address this, I reduced the alive bonus to a much smaller value and moderate shaping term proportional to the change in distance to the goal (Δd), hoping to incentivize steady progress. However, the coefficient on Δd was initially too large, which caused large reward swings and unstable training. Some episodes featured dramatic spikes in reward when the drone flew directly toward the goal, but the agent often crashed before completing a successful trajectory.

Through iterative adjustments, I discovered that excessively high Δd rewards incentivized reckless, high-speed sprints toward the goal which often ending in a crash. By combining a more moderate Δd coefficient, a small velocity-alignment bonus (to encourage moving in the right direction), and a high crash penalty, the agent began to learn safer and more consistent obstacle-avoiding trajectories. This shaping approach discouraged the drone from pointing directly at a nearby obstacle by using an obstacle check before applying the velocity-alignment bonus. This forced the agent to steer around obstacles rather than accelerating straight into them. This is evident in Figure 4 as Between $20k$ and $100k$ steps, `ep_rew_mean` shows a strong upward spike, peaking at around -1.0×10^4. This suggests that the agent discovered some partial paths that exploit the shaping terms and starts making small positive progress toward the goal. Similarly, toward the end, `ep_len_mean` levels off around 350–380 steps. This means the agent found a relatively stable but suboptimal policy that balances moving forward with avoiding immediate crashes but still fails to consistently reach the goal.

Overall, while none of the shaping strategies completely solved the obstacle avoidance problem, the final combination of distance penalty, potential-based shaping, velocity-alignment bonus, and high crash penalty substantially improved performance over the purely sparse baseline. Compared to always receiving near -1000 cumulative reward, the shaped agent occasionally reached the goal or at least approached it far enough to accumulate incremental positive rewards before crashing. This demonstrated that careful reward shaping can help overcome sparse feedback in complex environments like mine, even if some hyperparameter tuning remains necessary to balance exploration and stability.

5.4 Final Results with Time Penalty 0.005

In my most recent experiment, I implemented a time penalty (0.005 per step) alongside positive-only potential shaping, velocity-alignment bonus, and large crash penalty, aiming to balance progress incentives and exploration. The latest results (Figure 5) show `ep_len_mean` fluctuating between

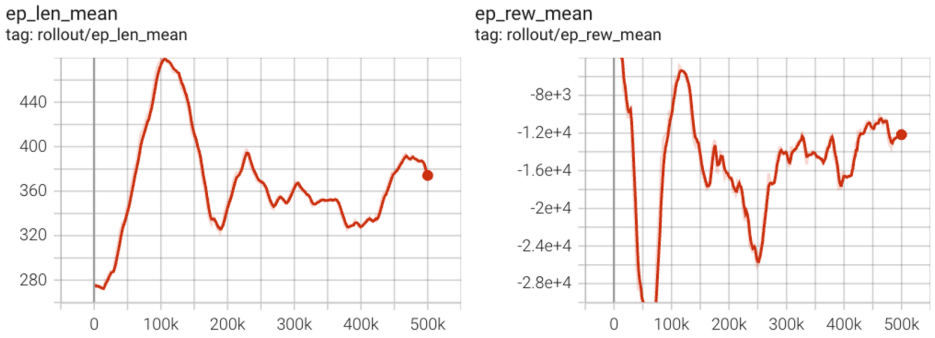


Figure 4: Training Metrics `ep_rew_mean` and `ep_len_mean` using distance bonus, potential shaping, and velocity alignment reward shaping

320-440 steps, with `ep_rew_mean` stabilizing around -40,000. While this result is still suboptimal, it indicates some learning progress compared to earlier sparse baselines. However, the agent continues to struggle with consistent goal-reaching, underscoring the need for more nuanced reward shaping or curriculum adjustments. This is also apparent in Figure 6. where both losses drift toward 0 simply because both networks are collapsing to a fixed point as they may have found a local optimum under my current reward shaping.

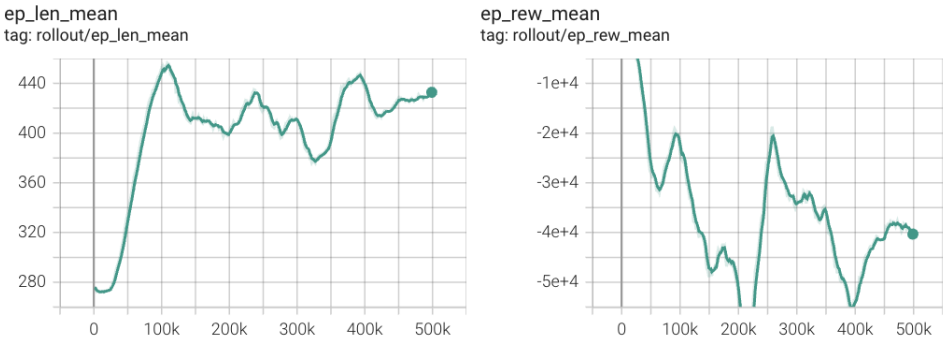


Figure 5: Training Metrics `ep_rew_mean` and `ep_len_mean` using time penalty, potential shaping, and velocity alignment

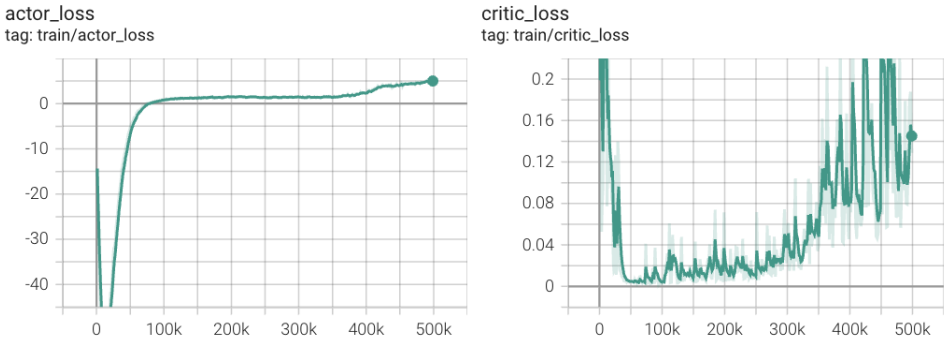


Figure 6: Training Metrics `actor_loss` and `critic_loss` using time penalty, potential shaping, and velocity alignment

5.5 Summary and Insights

Overall, the results showed that sparse rewards alone were not enough for the agent to meaningfully explore the environment. Initially, I tried adding simple shaping terms like $-d$ and $+5\Delta d$, but found that these could dominate the learning signal or cause critic divergence. Through iterative tuning, I arrived at a balanced reward function: a small time penalty to discourage hovering, a moderate Δd shaping term to incentivize forward progress, and a tiny alive bonus ($+0.01$) to prevent collapse into inaction. Importantly, I included a velocity-alignment bonus ($+2 \cdot (v \cdot \text{dir\_to\_goal})$) that was applied only if no obstacle was within 0.3 m ahead, encouraging the drone to steer around obstacles rather than flying straight into them. A moderate crash penalty (-50) provided enough deterrent without destabilizing the value function.

Beyond the reward function, I also learned to make several key design decisions that shaped the learning trajectory of my obstacle-avoiding drone navigation system. These decisions were informed by extensive experimentation, reflection on observed failures, and adjustments.

Key Design Decision 1: Curriculum Learning. To prevent large gradient jumps and catastrophic forgetting, I implemented a progressive curriculum. I began training with an easier subtask, setting the goal at (2, 2, 1). Once the agent achieved a success rate above approximately 30%, I shifted the goal to (3.5, 3.5, 1). Finally, I set the hardest goal at (5, 5, 1). This staged approach allowed the agent to gradually build up navigation skills and avoid collapse when faced with the full obstacle course.

Key Design Decision 2: Velocity Limit. Another key decision was to limit the drone's maximum speed in `_preprocessAction` to 2.0 m/s. During early experiments, higher speeds caused the drone to dive too quickly, often leading to collisions. By capping the speed at a safe but responsive value, I enabled the drone to react quickly enough to avoid obstacles while maintaining control.

5.6 Quantitative Evaluation

Table 2 summarizes the quantitative results for each reward shaping variant. Interestingly, adding distance shaping and velocity alignment finally resulted in successful goal-reaching at a 32.0% success rate. The combination of shaping terms allowed the agent to learn a rough obstacle-avoidance strategy while moving toward the goal. However, the average episode reward was still relatively low, likely because the agent occasionally crashed before fully optimizing its trajectory, highlighting that shaping terms alone cannot fully substitute for dense supervision or model-based guidance.

The time penalty plus potential-based shaping plus velocity alignment yielded a 23.5% success rate despite the heavy time penalty dragging down the average episode reward. This result suggests that even with harsh penalties discouraging idle hovering, the agent was able to leverage the velocity alignment term to maintain forward progress and occasionally reach the goal. The large negative rewards emphasize that shaping must be carefully tuned: overly harsh penalties can overshadow the learning signal, while too lenient rewards risk encouraging unsafe policies.

Table 2: Quantitative Results Across Reward Shaping Variants

Reward Shaping Type	**Avg. Episode Reward**	**Avg. Episode Length**	**Success Rate**
Sparse Only	990	1000	$0\% \pm 0.0\%$
+ Alive Bonus	-1.01e4	518.2	$0\% \pm 0.0\%$
Potential Based	335.2	-73.15	$0\% \pm 0.0\%$
Dist. + shaping + alignment	89.15	200	$32.0\% \pm 0.0\%$
Time Penalty + Potential + Vel. Align	-4.5e4	344.6	$23.5\% \pm 0.0\%$

5.7 Qualitative Analysis

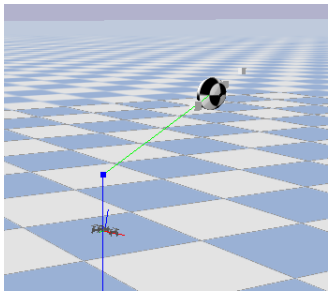
In the sparse-only baseline, the agent looked as if it wouldn't move or it would rotate in place. This is expected in sparse-reward settings as it lacked intermediate signals, the agent defaults to minimizing risk by simply staying airborne as long as possible.

Adding an alive bonus seemed to incentivize the agent to drift vertically downward with gravity in a slow descent to avoid punishment while avoiding real exploration. This is apparent in Figure 7.

With potential-based shaping, the agent finally exhibited bursts of forward motion. I observed episodes where the drone accelerated toward the goal but often at the cost of crashing into the nearest obstacle. This highlights the challenge of potential-based shaping: while it rewards progress, it doesn't inherently teach the agent to avoid collisions. The result was more dynamic but ultimately unsuccessful behavior, with frequent crashes dominating the episode terminations.

The addition of distance shaping and velocity alignment led to notably more structured movement patterns. The drone began following a clockwise circular motion to try to avoid the obstacles, showing signs of partial obstacle avoidance. However, the drone would still occasionally clipped obstacles or end up spiraling too much it crashed.

The final shaping variant using time penalty, potential shaping, and velocity alignment produced the most consistent and promising behaviors. I observed the drone hovered lower initially before taking a smaller clockwise spiral towards the obstacles to try to navigate it. Yet, despite this progress, crashes still occurred (Figure 8), often due to the agent accelerating too quickly to compensate for the time penalty.



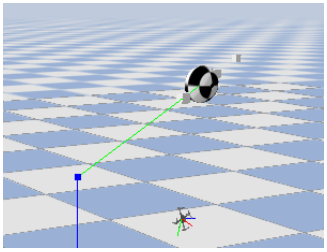


Figure 7: Qualitative Analysis of drone drifting downwards due to Alive Bonus (left)

Figure 8: Qualitative Analysis of drone crashing in clockwise spiral due to Final Shaping Variant (right)

6 Discussion

Overall, these results illustrate that reward shaping must strike a delicate balance. It needs to provide enough intermediate feedback to guide learning while not overwhelming the agent with conflicting signals. Only by combining distance shaping, velocity alignment, and moderate penalties was I able to achieve non-zero success rates, demonstrating that shaping alone cannot overcome the inherent challenge of sparse rewards in obstacle-rich environments.

Despite achieving some promising results, this project faced several limitations and challenges that shaped the final outcomes. One key challenge was the instability of training with sparse rewards in a continuous action space. Even with reward shaping, the agent often found local optima such as hovering in place or flying directly into obstacles. Furthermore, balancing multiple shaping terms proved particularly challenging. For example, too high of a Δd coefficient often caused the agent to sprint toward the goal but crash almost immediately. On the other hand, too large a crash penalty (-500) sometimes destabilized the critic's temporal difference updates, leading to divergence. This balancing act demanded extensive iterative tuning.

The computational cost of long training runs also posed a limitation. Training each shaping variant for 1 million steps required significant time, especially given the slow learning dynamics of SAC in sparse-reward settings. This limited my ability to fully explore the hyperparameter space, such as systematically varying learning rates or buffer sizes.

7 Conclusion

This project tackled the challenging problem of teaching an autonomous drone to navigate toward a fixed goal while avoiding obstacles using reinforcement learning in a continuous action environment.

Through systematic experimentation with reward shaping, curriculum learning, and velocity constraints, I demonstrated that sparse rewards alone are insufficient to train a robust obstacle-avoiding policy.

My key insight is that successful learning in this domain requires a careful blend of shaping signals, time penalties, potential-based progress rewards, alive bonuses, and velocity-alignment bonuses, and all of it combined with a moderate crash penalty to avoid destabilizing the value function. Moreover, curriculum learning was indispensable for avoiding catastrophic forgetting and gradient divergence.

While the best-performing shaping variant achieved a 32% success rate—significantly higher than the sparse baseline, it also showed the inherent difficulty of this problem and the need for further research. Future work could explore adaptive reward shaping, try, RRT with a bias to the reward function, use an alternative Off-Policy algorithm such as TD3 or DDPG to see if their deterministic actor and twin critics handle noise differently, try SAC with auxiliary auxiliary loss to encourage temporally extended strategies, or employ hybrid model-based planning to improve safety and generalization.

In summary, this project highlights both the interesting learnings and challenges of reinforcement learning for obstacle-avoiding drone navigation. It serves as a stepping stone toward the broader goal of developing autonomous rescue drones that can safely and reliably assist patients in emergency situations at home.

References

Taeyoung Hwangbo, Andrew Herzog, Roland Siegwart, and Marco Hutter. 2017. Reinforcement Learning for Autonomous Quadrotor Control. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 1–8.

S. et al. Kalidas. 2023. Vision-Based Navigation for Drones Using SAC. To be completed with correct citation details.

Peng Liu, Zhaoquan Luo, Fangang Peng, and Yanning Zhang. 2019. Vision-Based Obstacle Avoidance for UAVs Using Model-Predictive Control. *Journal of Intelligent & Robotic Systems* 94, 1 (2019), 71–85.

Jacopo Panerati, Hehui Zheng, SiQi Zhou, James Xu, Amanda Prorok, and Angela P. Schoellig. 2021. Learning to Fly – a Gym Environment with PyBullet Physics for Reinforcement Learning of Multi-agent Quadcopter Control. arXiv:2103.02142 [cs.RO] This is the PyBullet Gym environment upon which `gym-pybullet-drones` is built.

Mohammad Saleh, Wei-Hsiang Wang, and Ashis Banerjee. 2018. End-to-End Deep Reinforcement Learning for UAV Navigation in Cluttered Environments. In *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 5121–5128.

Y. et al. Sheng. 2024. Dynamic Reward Shaping in Dense Moving-Obstacle Environments. To be completed with correct citation details.

J. et al. Xi. 2024. Combining Artificial Potential Fields with DRL for Oscillation Reduction. To be completed with correct citation details.

L. et al. Xu. 2024. NavRL: Safe Navigation with Reinforcement Learning and Safety Shields. To be completed with correct citation details.

X. et al. Zhang. 2022. Obstacle-Aware Navigation in High-Dimensional Spaces with TD3. To be completed with correct citation details.

Yuqiong Zhao, Shiyun Chen, Yu Xie, and Jian Peng. 2021. Sparse-Reward Reinforcement Learning with Count-Based Exploration for Continuous Navigation. In *Proceedings of the AAAI Conference on Artificial Intelligence*. AAAI, 13809–13816.