Visualizing Hyperparameters in 2D Drone Navigation

Maanas Punuru
Panther Creek High School
Cary, North Carolina, USA
maanas.punuru@gmail.com

Jan Ole Ernst
University of Oxford
Oxford, England, United Kingdom
jan.ernst@physics.ox.ac.uk

Abstract

Reinforcement Learning (RL) is a subfield of Machine Learning that involves agents learning what actions to take given the current state and an eventual goal. RL has become prevalent in the modern world. RL-powered self-driving drones are used in modern society for various tasks, such as delivering packages. Thus, it would be practical and interesting to design an RL algorithm that attempts to teach a drone how to reach a target. However, small-scale learning is difficult due to the problem of learning to take sequential beneficial steps, especially with stochastic wind. In this paper, we address this issue by testing various hyperparameters in an attempt to mitigate these problems. These small-scale tests can be crucial to large-scale drone navigation problems, helping improve contemporary algorithms. In this paper, we test hyperparameters such as unique reward shaping formulae and discount factors. Multiple experiments conclude that some hyperparameters have a large effect on drone performance, while some do not.

Keywords

Reinforcement Learning (RL), Unmanned Aerial Vehicle (UAV), Hyperparameters, Reward shaping, Discount factor, Learning rate, Actor-Critic, Stochasticity, Proximal Policy Optimization (PPO)

ACM Reference Format:
Maanas Punuru and Jan Ole Ernst. 2026. Visualizing Hyperparameters in 2D Drone Navigation. In *Proceedings of International Journal of Secondary Computing and Applications Research (IJSCAR VOL. 3, ISSUE 2)*. ACM, New York, NY, USA, 6 pages. https://doi.org/10.67149/yhjs2024.5/bx8r3n6w

1 Introduction

Reinforcement Learning is a subfield of Machine Learning that involves agents learning actions to take given a current state. Reinforcement Learning algorithms learn from training episodes, where the task is simulated or even performed in a real-world environment, so that the agent learns the optimal actions to take.

Reinforcement Learning techniques are being widely used in modern commercial applications. Applications from Large Language Models (LLMs) to self-driving cars and drones use RL techniques and algorithms like Actor-Critic, Proximal Policy Optimization (PPO), and Q-learning.

Among many possible algorithms for a self-steering drone, an Actor-Critic architecture[1] is one way to implement a learning algorithm in an RL environment. The Actor-Critic system, as the name suggests, has two components: the actor and the critic, which are separate neural networks with their own parameters.

The actor takes the best actions sampled from the current policy, which incorporates what it has learned so far. After each episode, it learns based on its mistakes and what it did well. To determine how well the actor performed, a critic is necessary. The critic does not directly evaluate the actor's actions but instead evaluates the expected reward of the current state via a learned value function. Then, the actor's rewards are subtracted from this baseline. The difference is called the "advantage" and is used in backpropagation.

In this paper, we propose numerous reward shaping formulae and evaluate their performance on an Actor-Critic learning algorithm. We also test another hyperparameter, the discount factor, and test how constant and stochastic wind affect drone performance.

We also test the vanilla Actor-Critic algorithm's performance against PPO, which is a type of Actor-Critic that deals with large environments well.

2 Related Work

The applications of Reinforcement Learning to Unmanned Aerial Vehicle (UAV) navigation have been a significant area of research in recent years, particularly in developing autonomous systems for tasks like path planning and obstacle avoidance in complex environments [2]. Contemporary research frequently utilizes advanced Actor-Critic algorithms like Proximal Policy Optimization (PPO) for robust training in 3D flight control. PPO is widely used due to its performance in continuous control problems, often applied to complex obstacle avoidance in simulated and real environments [3]. The efficacy of an RL agent is highly dependent on its reward function, leading many studies to focus on reward shaping techniques. Similar to the distance-based shaping proposed later, researchers have explored methods to combat the issue of sparse rewards, where the agent only receives a substantial reward upon reaching the final goal, by providing intermediate rewards for making progress [4]. Furthermore, a major challenge in this field involves managing environmental uncertainty, such as the wind noise described in this paper. More complex models and recurrent neural network architectures have been explored to maintain performance in environments with unobservable or rapidly changing dynamics [5].

3 System

A custom environment was created in Gymnasium[6][1] to design the problem, and an Actor-Critic model was implemented to solve it.

IJSCAR VOL. 3, ISSUE 2

[1]Environment code available at: https://github.com/maanasp8/2D_Drone_Navigation_With_Wind_Functionality

3.1 Drone Environment

The Drone Delivery Environment simulates a randomly initialized drone that needs to reach a randomly initialized target on a grid of the user's preference. The drone and the target are initialized randomly before each episode. The user can decide the size of the grid, but it must be a square. If the drone reaches the boundary of the grid, the drone will not be permitted to take any action that would cause it to leave the grid. The boundary acts as a hard wall, ensuring that the drone stays in the grid.

At each timestep, the drone can move a set number of grid units in each direction as specified by the step size variable. For example, if the step size was set to 2, it could move up to 2 units in the x-axis and 2 units in the y-axis.

This environment is a multi-step environment in which the drone will have to take multiple beneficial steps to reach the target. To ensure that episodes do not take too long, there is a maximum steps setting that the user can set to ensure that the drone does not take prolonged periods of time to reach the target.

Additionally, the environment-specified rewards are +100 for reaching the target and -1 at every timestep. From these, the drone will learn to take as few steps as possible and reach the target. Since the penalties of -1 are accumulated if the target is not reached, the drone learns to minimize the steps it takes.

$$r_i = \begin{cases} r_{i-1} + 100, & \text{if the drone reaches the target at step } i, \\ r_{i-1} - 1, & \text{if the drone does not reach the target at step } i. \end{cases} \tag{1}$$

where:

$$r_i : \text{total reward at timestep } i$$

$$r_{i-1} : \text{total reward at timestep } i-1$$

The environment also contains optional functionality for wind in both the x and y axes that affects the drone's movement. The wind pushes the drone a number of grid units depending on the wind magnitude. Additionally, the user can turn on wind noise, which stochastically updates the wind at every timestep, similar to a random walk. The user must also set a maximum wind magnitude so that the wind does not become too large and unrealistic.

3.2 Reinforcement Learning Algorithm

The Reinforcement Learning algorithm will take in the current state, which has 6 components: The drone's location in the x-axis, the drone's location in the y-axis, the target's x-position, the target's y-position, the current wind in the x-axis, and the current wind in the y-axis.

The policy network will output an action based on the current state. The state attributes will pass through the layers of the neural network until an action is returned. The number of actions it can choose depends on how the user sets the step size variable. For example, if the step size was set to 1, the drone could move a maximum of 1 unit in each direction or stand still, creating 9 possible actions. Standing still is an option for the drone because the only way to reach the target at a timestep may be to use the wind instead of moving.

Similarly, the value network will take in the attributes of the state and output a value corresponding to the expected reward. This

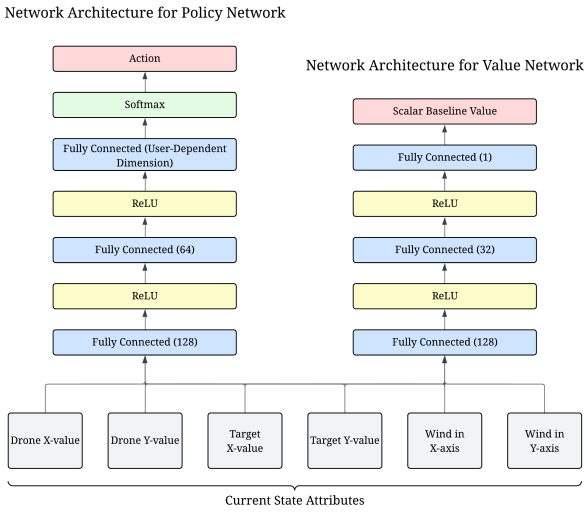

Figure 1: The architecture of the policy and value networks.

reward is used in back propagation to improve the model. After an episode is complete, the network learns how it could have taken better actions and updates the weights and biases accordingly.

The policy and value networks will both have similar architectures, with both having 3 layers. The dimensions of the policy network will be 128x64x1, while the dimensions of the value network will be 128x32x1. This is because the number of nodes in the output layer of the policy network is large, since it will have one node for each possible action, each representing the favorability of taking any action. This will then go through a softmax layer to determine the optimal action. So, the second-to-last layer must be large to account for all of the factors that may affect each possible action. The value network, however, only outputs a single value that corresponds to the favorability of the state. Thus, a 32-node layer is still large in comparison to the output layer, ensuring the output layer has enough information to be accurate.

3.3 Reward Functions

To help the algorithm learn better, reward shaping[7] can be implemented. Reward shaping involves adjusting the rewards of taking certain actions so that the algorithm learns to take actions that human practitioners believe are beneficial. Reward shaping is often helpful during the algorithm's exploration phase, where it is still new to the environment. The agent will usually benefit from signals about what actions to take, especially near the beginning of the training run. However, reward shaping does not always work in complex environments, since humans might not know the most optimal path to the final goal themselves.

In this environment, it is evident that getting closer to the target is optimal, even if it is impossible to reach the target on the current time step. So, by rewarding the algorithm for getting closer to the target, the algorithm will learn to take better steps.

The reward functions that were tested involved subtracting the drone's distance from the target and the drone's previous distance

from the target to determine if the drone moved closer to the target or not. Then, that value is multiplied by a variable scale factor to adjust the weighting that the algorithm places on the reward shaping. This formula ensures that the algorithm values taking consecutive beneficial steps.

When the drone is far from the target, it is difficult to attain the +100 reward for teaching it because the drone must take many correct sequential steps in order to get close. This is facilitated by the distance-based reward shaping formula that rewards the drone for getting closer to the target and helps with exploration in the initial steps. With this distance-based reward, the drone learns to take the largest steps possible in order to get closer to the target and attain a bigger distance-based reward.

$$r_i = \begin{cases} r_{i-1} + 100, & \text{if the drone reaches the target at step } i, \\ r_{i-1} + W \times (D_{i-1} - D_i), & \text{if the drone does not reach the target at step } i. \end{cases} \tag{2}$$

where:

W : reward shaping weight

D_i : distance to the target at timestep i

D_{i-1} : distance to the target timestep $i-1$

This formula represents this reward function, where W is a positive constant weight that influences the reward function. D_{i-1} represents the old distance from the target, and D_i represents the new distance from the target. This difference determines whether the agent gets closer to the target or not, and rewards it accordingly.

Although the distance-based reward shaping helps the drone initially learn to take steps to get close to the target, when the drone is actually close to the target, it becomes difficult for it to actually reach the target since it has learned that larger steps are always better than small steps. So, it tends to take steps that go beyond the target instead of directly reaching it.

To combat this, the reward shaping function can be altered so that the reward shaping has less of an impact on the algorithm when it is close to the target. There are two ways that this can be implemented: an abrupt removal of reward shaping if the drone's position lies within a certain radius of the target, and a gradual diminishing of reward shaping based on the drone's distance from the target.

$$r_i = \begin{cases} r_{i-1} + (W \times (D_{i-1} - D_i) \times (\frac{D_{i-1}}{\sqrt{2} \times E})), & \text{if the drone does not reach the target at step } i, \\ r_{i-1} + 100, & \text{if the drone reaches the target at step } i. \end{cases} \tag{3}$$

where:

W : reward shaping weight

D_i : distance to the target at timestep i

D_{i-1} : distance to the target timestep $i-1$

E : environment grid width

This formula represents the gradually diminishing reward shaping formula. For this formula, the drone receives a reward of 100 upon reaching the target. However, when the drone does not reach the target, the reward it would normally get based on the regular reward shaping formula is multiplied by a factor that increases the weight of shaping if the drone is far from the target and decreases the weight of shaping if the drone is close to the target. In the formula, E represents the width of the environment's grid.

$$r_i = \begin{cases} r_{i-1} + W \times (D_{i-1} - D_i), & \text{if the drone is more than one step away from the target at step } i-1, \\ r_{i-1} - 1, & \text{if the drone is within one step of the target at step } i-1 \text{ and does not reach the target at step } i, \\ r_{i-1} + 100, & \text{if the drone is within one step of the target at step } i-1 \text{ and reaches the target at step } i. \end{cases} \tag{4}$$

where:

W : reward shaping weight

D_i : distance to the target at timestep i

D_{i-1} : distance to the target timestep $i-1$

This formula represents the abruptly diminishing reward shaping formula, where regular reward shaping is implemented when the drone is more than 1 step away from the target. If the drone is within one step of the target, however, the reward shaping turns off, and the environment's rewards revert to the original ones. If the drone moves from within one step of the target in a way that it is no longer within one step of the target, the reward shaping is turned back on.

4 Evaluation

To achieve optimal drone performance, many different hyperparameters were tested. Some of these include learning rate, discount, and reward shaping. Key hyperparameters whose impact is visible have been displayed in graphics, while other hyperparameters have been described. Tests were performed on a seeded environment to mitigate stochasticity that may alter results.

Often, when solving Reinforcement Learning problems, the rewards are plotted to analyze agent performance. However, to solve the drone environment, many different hyperparameters have been tested, such as reward shaping and the discount factor, which artificially alter the reward to improve learning. Thus, it is best to plot the drone step count to reach the target and analyze patterns within the step count for episodes, since lower step counts signify better performance across reward functions and always imply a higher reward within an experimental setup (Eq. 1).

For all tests, the grid size was set to 10, and the step size was set to 2. This means that the drone could move up to 2 units in both the X and Y directions at every timestep in a 10 by 10 grid. So, the expected time (with random drone and target initialization) for an optimally moving drone to reach the target is about 2.6 timesteps.

This was also plotted on all graphs to compare learning to the maximimum level of performance possible.

For each hyperparameter possibility, 10 seeds were tested to reduce variance. Statistical testing, like Welch's ANOVA[8], was conducted to determine whether results were caused by random chance or not, using the last 400 steps to evaluate performance.

The two main hyperparameters that were tested were the discount factor and the reward shaping formula. Other factors that were also tested but whose performance is not displayed include learning rate and shaping weight. These were not displayed because their impact is very predictable and known.

Also, the drone's performance when faced with stochastically updating wind of various maximum wind magnitudes was tested. Constant wind was not tested because the reason for including wind in the environment was to make the environment more realistic, and stochastically updating wind is more realistic than constant wind.

Also, to analyze the Actor-Critic algorithm's performance itself, its performance was tested against a PPO algorithm. Compared to the Actor–Critic implementation, the PPO algorithm introduces several additional hyperparameters that regulate how policy updates are performed. The PPO implementation uses a clipping parameter of 0.2, which constrains the policy probability ratio during optimization to prevent excessively large policy updates and improve training stability. It also performs 4 optimization passes over the same batch of experience, allowing the algorithm to extract more learning signals from each collected trajectory. During training, experience is accumulated over 10 episodes before performing a policy update, and the resulting data are divided into mini-batches of 256 timesteps for stochastic optimization. This batching and repeated reuse of data theoretically improves learning capacity compared to the Actor–Critic approach, which updates the policy directly from a single episode of experience.

Both implementations share several core experimental settings to ensure a fair comparison. In particular, they use the same Actor and Critic network dimensions, a discount factor of 0.9, and the same gradual reward shaping function (Eq. 3). Using identical discounting and shaping mechanisms isolates the algorithmic differences between Actor–Critic and PPO, allowing any performance differences to be attributed to the differences in the policy optimization method for PPO and Actor-Critic.

Because of PPO's policy optimization method, it generally performs better on environments that are more stochastic. So, its performance was also tested with the stochastic wind to compare performance with Actor-Critic.

All tests were run with 10 random seeds for each parameter value. These same 10 seeds were used for all tests to ensure consistency. To test statistical significance, a Welch's ANOVA test[8] was performed. This test determines the chance that the difference in performance of one parameter value from the others was a result of the value itself or of random chance.

5 Results

Many different discount factors were tested, these being 0.99, 0.98, 0.95, 0.90, 0.85, and 0.80. It was found that the discount factor had a seemingly large effect on drone performance, with average steps needed varying by 10 based on the discount factor selected (Fig. 2). However, after performing a Welch's ANOVA test[8], it was found that the difference between the discount factors was not statistically significant ($p = 0.33$).

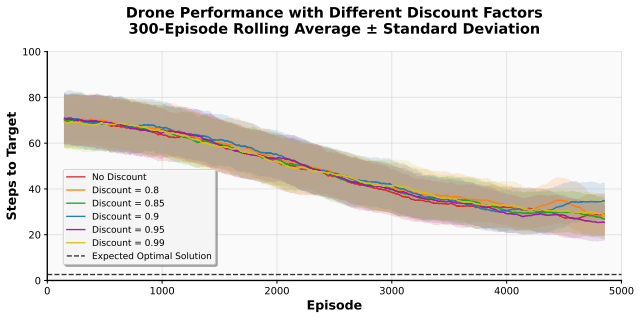

Figure 2: The effects of various discount factors on drone performance.

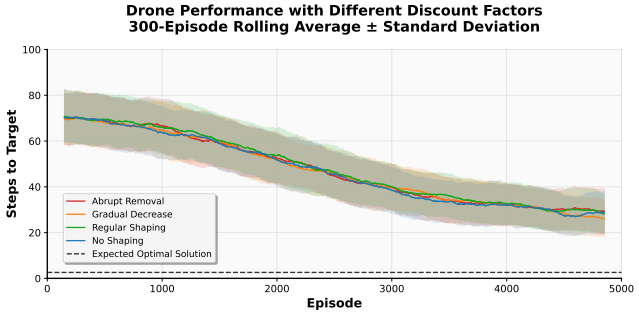

Figure 3: How different reward shaping algorithms affect drone performance.

This is likely because, moving optimally, the drone should be able to reach the target in, on average, 3-4 steps. So, the discount factor does not have a large compounding impact on rewards, as each episode ends quickly.

Table 1: Performance Across Discount Factors

Discount Factor	Mean Steps to Target	Std Dev
$\gamma = 0.80$	28.21	3.10
$\gamma = 0.85$	26.87	2.02
$\gamma = 0.90$	34.75	21.82
$\gamma = 0.95$	25.39	3.97
$\gamma = 0.99$	28.60	5.00

Thus, varying the discount greatly varied drone step count (25.39 to 34.75), although Welch's ANOVA indicated that the differences were not statistically significant ($p = 0.33$).

When both of the custom reward functions were tested against the regular reward shaping function and the environment-defined reward, it was found that the gradual diminishing reward and the constant reward shaping performed the best (Fig 3). However, Welch's ANOVA[8] indicated that the differences between reward

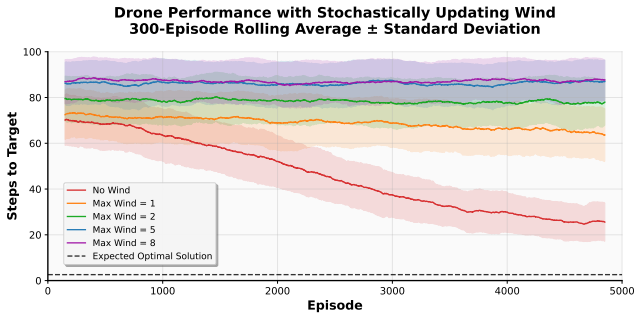

Figure 4: Drone performance when stochastically updating wind is implemented.

shaping methods were not statistically significant ($p = 0.60$). This suggests that the observed performance differences may be attributable to random chance across runs rather than the formulas themselves.

This is likely because the environment's signal of -1 and +100 was enough for the agent to generally get a good sense of needing to take steps in the direction of the target.

Table 2: Performance Across Reward Shaping Methods

Reward Shaping Method	Mean Steps to Target	Std Dev
Abrupt	29.22	5.37
Gradual	26.27	5.23
None	28.95	5.26
Regular	28.15	5.71

Thus, gradual reward shaping achieved the lowest mean step count (26.27 ± 5.23), although Welch's ANOVA indicated that the differences between methods were not statistically significant ($p = 0.62$).

To guide the drone toward reaching the target, reward shaping algorithms of various weights were also tested. These weights would be multiplied by the change in distance from the previous state and added to the final reward. It was found that the weight that had the best results was a weight of three, with larger weights causing the algorithm to value reaching the target less, and smaller weights not having much of an impact on the algorithm's convergence.

When a stochastically updating wind is implemented, it appears as if the drone almost stops learning completely. This can be seen in Fig. 4. When a Welch's ANOVA test was conducted, it was found that the impact of wind on performance was very statistically significant ($p = 0.00$). It appears that an extremely small amount of learning occurs when stochastically updating wind noise is implemented. The rapid stochastic updates introduce large amounts of variance in the drone's training data, causing difficulty in the drone's ability to learn to take beneficial actions, especially in a limited training window.

So, varying the wind greatly affected drone step count (25.39 to 87.89), which was proved when Welch's ANOVA indicated that the differences were statistically significant ($p = 0.00$).

Table 3: Performance Across Wind Conditions

Wind Condition	Mean Steps to Target	Std Dev
0 Max Wind	25.56	3.15
1 Max Wind	63.64	2.46
2 Max Wind	77.98	0.93
5 Max Wind	87.00	1.30
8 Max Wind	87.69	1.29

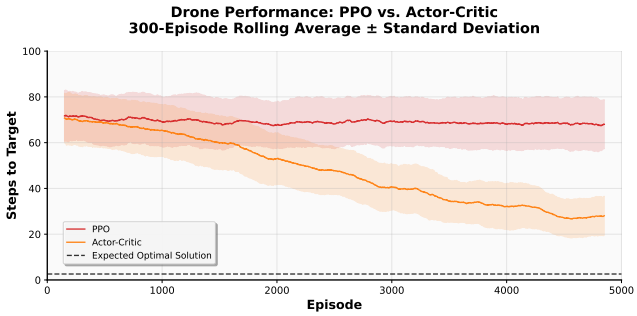

Figure 5: Comparison of Actor-Critic vs. PPO performance.

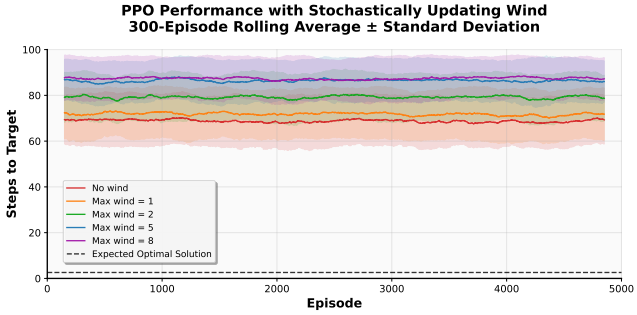

Figure 6: Visualization of PPO performance when faced with stochastically updating wind.

To evaluate the performance of the Actor-Critic algorithm, it was compared with PPO. It is evident that Actor-Critic performed much better than PPO, as can be seen in Fig. 5. This is likely because PPO is a complicated algorithm that requires long training epochs and many training epochs for the agent to become familiar with the environment. Actor-Critic, on the other hand, is a simpler algorithm that can gain insight quickly, as it updates after each episode. However, it struggles to gain deep insight into highly optimal courses of action. Since these tests were run in a limited training window, this is not a problem.

Since PPO fares well with stochasticity, its performance was assessed with stochastically updating wind (Fig. 6). It is evident that wind has a negative effect on PPO performance. Again, the aforementioned problem of PPO needing longer to learn an effective policy applies here, as it appears that PPO is not learning much or even at all.

6 Conclusions

This study demonstrates the effect of hyperparameter tuning with an Actor-Critic architecture in 2D drone navigation. Hyperparameter tuning causes a large change in performance by impacting the drone's learning through human input. Constant wind and stochastically updating wind were also tested.

The learned policy of the drone is fit to the structure of the simulated environment, so the learned policy will likely vary drastically from that of a real UAV. However, the method of taking sequential steps to move closer to the target remains the same.

Four reward shaping methods were tested: No reward shaping, regular distance-based shaping, gradually diminishing shaping, and abruptly removing shaping. It was found that there was no shaping method that improved performance by a statistically significant amount. This is likely because the environment-defined rewards of -1 and +100 were still themselves enough to give the agent a signal of where to move.

It was found that the effect of the discount factor on performance was not statistically significant. The reason for this is likely the size of the environment. Since the environment was only a 10-unit grid, optimal training episodes should conclude in around 3-4 steps. Because of this, discount factors will not have a large effect on the reward values, as not many steps would have passed for the discount to compound significantly.

When stochastically updating wind was implemented, the drone received more variance in the data. So, it was harder for the drone to learn how wind affects movement. Thus, the drone performed worse when larger maximum wing magnitudes were given, since the stochasticity has more of an effect then.

When the Actor-Critic implementation was tested against PPO, it was found that PPO performed significantly worse, even when both were faced with stochastically updating wind. This is likely because of the limited training window, since PPO generally requires longer training sessions to learn an effective policy.

Future work should attempt to make these experiments more realistic, adding a 3rd dimension, simulating the drone's ascent and descent. Additionally, making the action space continuous instead of a fixed grid will represent more realistic problem setting. Obstacles, such as buildings or trees, that cannot be crossed or can only be crossed at a certain elevation, can also add to the verisimilitude of the environment.

All of these additions will make it difficult for the agent to learn specific actions to take. So, a working model will likely use more layers and have more nodes per layer to better interpret and represent the complexities of the environment. Also, the learning rate will likely be smaller to facilitate the difficult learning process. As a result, the algorithm will likely need to run for thousands of episodes for any results to become visible.

Although PPO performed worse here, when continuous states and action spaces are introduced, and the algorithm is given many more training runs to learn, PPO becomes a better option than what is used in this paper. PPO is extremely sensitive to hyperparameters but is also very stable, so it is best used in larger policy networks, which will likely be implemented if the environment becomes more complex.

References

[1] Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D., & Kavukcuoglu, K. Asynchronous methods for deep reinforcement learning. 10.48550/arXiv.1602.01783.

[2] Yang, T., Yang, F., & Li, D. A new autonomous method of drone path planning based on multiple strategies for avoiding obstacles with high speed and high density. Drones. 10.3390/drones8050205.

[3] Zhao, W., Chu, H., Miao, X., Guo, L., Shen, H., Zhu, C., Zhang, F, & Liang, D. Research on the multiagent joint proximal policy optimization algorithm controlling cooperative fixed-wing UAV obstacle avoidance. Sensors, 20(16):4546. 10.3390/s20164546.

[4] Ye, C., Zhu, W., Guo, S., & Bai, J. DQN-based shaped reward function mold for UAV emergency communication. Applied Sciences, 14(22):10496. 10.3390/app142210496.

[5] Duoxiu, H., Wenhan, D., Wujie, X., & Lie, H. Proximal policy optimization for multirotor UAV autonomous guidance, tracking and obstacle avoidance. International Journal of Aeronautical and Space Sciences, 23:339–353. 10.1007/s42405-021-00427-2.

[6] Towers, M., Kwiatkowski, A., Terry, J., Balis, J., De Cola, G., Deleu, T., Goulão, M., Kallinteris, A., Krimmel, M., KG, A., Perez-Vicente, R., Pierré, A., Schulhoff, S., Tai, J. J., Tan, H., & Younis, O. Gymnasium: A Standard Interface for Reinforcement Learning Environments. NeurIPS (2025), 10.48550/arXiv.2407.17032.

[7] Ibrahim, S., Mostafa, M., Jnadi, A., Salloum, H., & Osinenko, P. Comprehensive Overview of Reward Engineering and Shaping in Advancing Reinforcement Learning Applications. IEEE Access (2024), 10.1109/ACCESS.2024.3504735.

[8] Kao LS, Green CE. Analysis of variance: is there a difference in means and what does it mean? J Surg Res. 2008 Jan;144(1):158-70. doi: 10.1016/j.jss.2007.02.053. Epub 2007 Oct 22. PMID: 17936790; PMCID: PMC2405942.

Received 13 January 2026; Accepted 20 March 2026