Comparative Analysis of DQN and PPO Algorithms in UAV Obstacle Avoidance 2D Simulation

Zoriana Rybchak, Mykhailo Kopylets

Lviv Polytechnic National University, S. Bandera Street, 12, Lviv, 79013, Ukraine

Abstract

This paper investigates the performance of Deep Q-Network (DQN) and Proximal Policy Optimization (PPO) algorithms in the context of two-dimensional simulated drone navigation, focusing on obstacle avoidance. It explores the usage of reinforcement learning to improve drones' navigational capabilities in environments populated with various obstacles. The study assesses how some activation functions, such as ReLU, Leaky ReLU, Tanh, and Sigmoid, impact the neural networks' performance and decision-making. Through comparative analysis, it highlights the unique aspects of DQN and PPO in navigation tasks and offers valuable insights for further investigations. The results demonstrate that the selection of activation function and algorithm significantly influences model efficiency in obstacle navigation, affecting overall strategy in simulations. This emphasizes the need for a comprehensive approach in choosing both architecture and activation functions for developing models with improved adaptability and strategic depth in complex scenarios.

Keywords

DQN, PPO, drone navigation, obstacle avoidance, reinforcement learning, activation function

1. Introduction

The advent of Unmanned Aerial Vehicles (UAVs) [1] commonly known as drones, has transformed numerous industries by providing new abilities for surveillance, delivery, military operations, and disaster management, among other uses. The ability of a UAV to autonomously navigate in complex environments is a fundamental prerequisite for these applications. This requires advanced obstacle avoidance systems that guarantee safety and operational efficiency. The importance of this study comes from creating and comparing these systems with the latest Deep Reinforcement Learning (DRL) algorithms [2].

This study focuses on a two-dimensional (2D) simulation environment that models the dynamics of UAV flight and interaction with various obstacles. The primary goal is to assess the performance of two prominent DRL algorithms, Deep Q-Network (DQN) [3] and Proximal Policy Optimization (PPO) [4] in guiding a UAV through cluttered environments. This involves an analysis of the UAV's ability to learn and adapt to its surroundings to avoid collisions and survive efficiently.

The specific objectives of this research are as follows:

- to develop a simplified 2D simulation environment to simulate UAV navigation challenges;
- to use a fine-tune DQN and PPO algorithms, ensuring their suitability for the task of UAV navigation and obstacle avoidance;
- to investigate the impact of different activation functions on the learning efficiency and performance of the DRL models;
- to evaluate and compare the performance of DQN and PPO algorithms in various simulated scenarios, providing insights into their strengths and limitations;

COLINS-2024: 8th International Conference on Computational Linguistics and Intelligent Systems, April 12–13, 2024, Lviv, Ukraine

zoriana.l.rybchak@lpnu.ua (Z. Rybchak); mykhailo.m.kopylets@lpnu.ua (M. Kopylets)

0000-0002-5986-4618 (Z. Rybchak); 0009-0004-5823-9871 (M. Kopylets)

- to draw practical conclusions that can aid in the design of more robust and reliable UAV navigation systems in the future.

This research aims to contribute to the understanding of autonomous UAV navigation in a controlled 2D simulation setting, emphasizing the exploration of DRL algorithms' efficacy in obstacle avoidance. While the findings offer insights into algorithmic behaviors and potential improvements, they should be considered preliminary and largely relevant to similar simulated contexts. Future studies might expand these results, exploring their relevance in more complex or real-world scenarios.

2. Related Works

The comprehensive study [5] conducted by Reuf Kozlica and colleagues explores the comparative efficacy of DQN and PPO within the framework of a material sorting task simulation. This investigation shows that PPO outperforms DQN in terms of learning speed and task completion. It also points out the importance of how rewards are set up for these algorithms, showing that changes in reward structures can affect how well they learn. This study emphasizes the benefits of using advanced reinforcement learning techniques like PPO and suggests looking into more complex methods to make these models work better in industrial settings. The insights from this research are important. It provides key information that is very useful for other areas of study, including drone navigation and obstacle avoidance, even though the main focus was on a different task. This connection highlights the wide-ranging usefulness and importance of comparing DQN and PPO in different types of challenges.

Another research [6] conducted by Amudhini P. Kalidas and colleagues presents a detailed evaluation of DQN, PPO, and Soft Actor-Critic (SAC) [7] for UAV obstacle avoidance, focusing on both stationary and moving obstacles. This study, conducted in a simulated 3D environment, underscores the significance of selecting the appropriate reinforcement learning strategy for specific UAV tasks, especially in complex and dynamic scenarios. SAC demonstrates a superior performance highlighting the benefits of off-policy algorithms in environments requiring nuanced decision-making and flexibility. DQN's success, despite its discrete action space limitation, emphasizes the importance of sample efficiency and the strategic use of replay buffers. Conversely, PPO's challenges in this context reveal the limitations of on-policy algorithms in handling the complexity of 3D environments with dynamic obstacles.

This thorough evaluation of DQN, PPO and SAC algorithms contributes significantly to the conversation around effective UAV navigation and paves the way for further studies. The findings suggest exploring hybrid approaches or further algorithmic enhancements to address the identified limitations, particularly for on-policy methods like PPO.

3. Methods and Materials

3.1. Reinforcement learning

Reinforcement learning is a branch of machine learning where an agent learns to make decisions by performing actions in an environment to achieve some objectives. The learning process is based on the agent's interactions with the environment, where it learns from the consequences of its actions through a process of trial and error, receiving feedback in the form of rewards, which it seeks to maximize over time.

In an RL framework, the agent interacts with the environment in discrete time steps. At each time step, the agent observes the current state of the environment, decides on an action based on its policy - a strategy that specifies the action to be taken in each state - executes the action, and then receives a reward and observes a new state from the environment. The ultimate goal of the agent is to learn a policy that maximizes the cumulative reward it receives over time.

The core components of an RL are:

- *Environment*. The domain or space in which the agent operates and makes decisions.

- *Agent*. The learner or decision-maker that interacts with the environment.
- *State*. A representation of the current situation or condition of the environment.
- *Action*. The set of all possible moves or decisions that the agent can make.
- *Reward*. Feedback from the environment used to evaluate the effectiveness of an action.

The process of reinforcement learning can be visualized in a Figure 1 that illustrates the cyclic interaction between the agent and its environment:

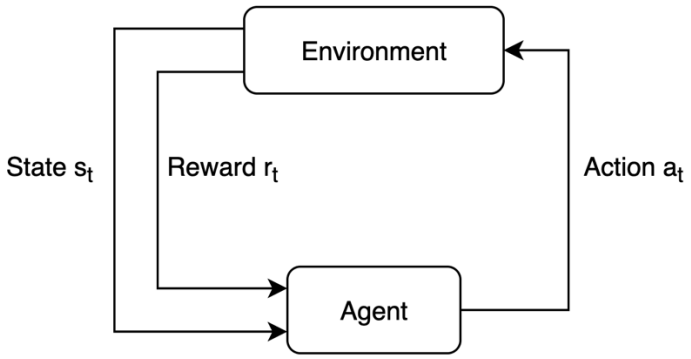

Figure 1: The reinforcement learning process

Reinforcement learning encompasses a wide range of algorithms and strategies. While foundational methods such as Q-learning [8] and SARSA [9] lay the groundwork for understanding RL dynamics, this study focuses exclusively on DQN and PPO. These advanced methodologies incorporate deep learning to effectively handle environments characterized by high-dimensional observation spaces, offering sophisticated solutions for complex decision-making tasks.

3.2. Deep Q-Network

The DQN is a model-free, off-policy reinforcement learning algorithm. This algorithm is particularly distinguished by its use of a neural network to approximate the Q-function, which is pivotal in estimating the expected rewards for action-state pairs.

In the $stable\_baselines3$ implementation [10], the DQN algorithm utilizes the *experience replay* mechanism, which stabilizes training by storing the agent's experiences and utilizing them for multiple learning iterations. By storing the agent's experiences at each timestep, denoted as $e_t = (s_t, a_t, r_t, s_{t+1})$, in a dataset known as the replay buffer, DQN decouples the correlation between consecutive experience samples. Here, each experience consists of the state s_t, the action $a_t$the reward r_t, and the subsequent state s_t.

During the training process, rather than learning from consecutive experiences as they occur sequentially within the environment, the DQN algorithm randomly samples a mini-batch of experiences from the replay buffer. This stochastic sampling method serves to break the correlation between consecutive learning steps, reducing update variance and leading to more stable and efficient learning.

Given a mini-batch of experiences (s, a, r, s') sampled from the replay buffer, the loss function for updating the Q-network is defined by:

$$L(\theta) = E_{(s,a,r,s')\sim U(D)}\left[\left(r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta)\right)^2\right] \quad (1)$$

where:

- $U(D)$ represents the uniform distribution over the dataset D (the replay buffer),
- γ is the discount factor,
- θ denotes the parameters of the Q-network being trained,
- θ^- are the parameters of the target network, updated periodically to stabilize the learning process by providing a fixed target for Q-value updates.

This approach allows the DQN to more effectively leverage its past experiences, leading to improved sample efficiency and faster convergence.

3.3. Proximal Policy Optimization

The PPO algorithm represents a significant advancement in on-policy, policy gradient methods for reinforcement learning, striking a balance between sample efficiency, simplicity, and ease of tuning.

PPO's implementation [4] within the *stable_baselines*3 library harnesses an *actor-critic* approach, to concurrently predict both action values and value functions. This method aims to improve the stability and efficiency of policy updates, making it one of the most widely used algorithms in complex environments. PPO operates within the actor-critic framework, where the "actor" updates the policy based on the gradient of expected rewards, and the "critic'' estimates the value function, serving as a baseline to reduce variance in the policy gradient estimation. The actor and critic are represented by neural networks with parameters θ for the actor and ϕ for the critic.

The core of PPO's innovation lies in its objective function, which seeks to take the greatest advantage of the policy update step without causing excessive changes to the policy. The objective function, $L^{CLIP}(\theta)$, is designed to minimize the cost of deviation from the old policy while encouraging improvement. It is defined as:

$$L^{CLIP}(\theta) = E_t\left[\min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\varepsilon, 1+\varepsilon)\hat{A}_t)\right] \quad (2)$$

where $r_t(\theta)$ is the probability ratio $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$, indicating how the new policy π_θ diverges from the old policy $\pi_{\theta_{old}}$. $\hat{A}_t$ is an estimator of the advantage function at time t, which measures the benefit of taking action a in state s over the expected value. $clip(r_t(\theta), 1-\varepsilon, 1+\varepsilon)$ constrains $r_t(\theta)$ to be within the range $[1-\varepsilon, 1+\varepsilon]$, with ε being a hyperparameter, typically set to a small value like 0.2.

The advantage function, $\hat{A}_t$, is critical for calculating the policy gradient. It is usually estimated using the critic's value function $V_\phi(s_t)$, leading to the expression:

$$\hat{A}_t = r_t + \gamma V_\phi(s_{t+1}) - V_\phi(s_t) \quad (3)$$

This estimator helps in determining the relative value of each action compared to the baseline value of the state, as provided by the critic.

Training in PPO alternates between sampling data through interaction with the environment using the current policy and optimizing the clipped objective function with respect to the actor's parameters. The critic's parameters are updated to minimize the value function error, typically using mean squared error loss:

$$L^{VF}(\phi) = (V_\phi(s_t) - V_t^{target})^2 \quad (4)$$

where V_t^{target} is the target value for state s_t, often computed using bootstrapped estimates of future rewards.

PPO's effectiveness comes from its balance between exploration and exploitation, achieved through the clipped objective function, which prevents overly aggressive updates that could lead to policy performance degradation.

3.4. Activation functions

Activation functions [11] play a pivotal role in neural networks by introducing non-linear properties that enable the network to learn complex patterns. In the context of DQN and PPO algorithms, the choice of activation function can significantly influence the performance and convergence of the models. The experiments in this study concentrate on four well-known activation functions: Leaky ReLU, ReLU, Tanh, and Sigmoid. The choice of these specific functions is based on the variation in their output value ranges. Leaky ReLU does not have a lower or upper limit, while ReLU is limited only below zero. Conversely, Tanh and Sigmoid are bounded on both

sides. Such variations influence the way gradients are sent back through the network during training, resulting in different learning behaviors and performance results.

The *Rectified Linear Unit (ReLU)* is perhaps the most popular activation function in deep learning architectures. This function passes positive values through unchanged while replacing negative values with zero. This characteristic helps to avoid the vanishing gradient problem during the training of deep neural networks, simplifying the optimization process and improving overall training speed. It is defined as:

$$f(x) = \max(0, x), \quad range(f) \in [0; +\infty) \tag{5}$$

The *Leaky Rectified Linear Unit (Leaky ReLU)* is a modification of the ReLU that permits a slight, non-zero gradient when the neuron is inactive. In contrast to ReLU, which may experience the problem of dead neurons throughout training, Leaky ReLU addresses this by allowing a small negative slope. This ensures that neurons stay active and keep adjusting throughout the learning phase, preventing the issue of neurons becoming non-responsive. It is defined as:

$$f(x) = \begin{cases} 0.01x, & if\ x < 0 \\ x, & if\ x \geq 0 \end{cases}, \quad range(f) \in (-\infty; +\infty) \tag{6}$$

The *Sigmoid* function, or logistic function, compresses its input values to fall within the range of 0 to 1. Historically, it was a favored option for binary classification tasks because its output could be interpreted as a probability. However, its application in deep learning has decreased because of the vanishing gradient issue, especially in layers that are distant from the output. This problem makes it challenging for the network to learn and adjust the weights of neurons in earlier layers effectively. It is defined as:

$$f(x) = \frac{1}{1+e^{-x}}, \quad range(f) \in (0; 1) \tag{7}$$

The *Hyperbolic Tangent (Tanh)* function produces outputs in the range of -1 to 1, effectively making it a scaled variation of the sigmoid function. This attribute means that the Tanh function centers the data, which can enhance convergence during the gradient descent process. This feature is especially advantageous in situations where normalizing the input data is beneficial, as it helps in stabilizing the learning process and can lead to faster convergence. It is defined as:

$$f(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}, \quad range(f) \in (-1; 1) \tag{8}$$

Each activation function contributes distinct characteristics to the learning process. ReLU and Leaky ReLU can accelerate the convergence of stochastic gradient descent due to their linearity and non-saturation of gradients. On the other hand, Tanh and Sigmoid, with their saturated nature, can regularize the learning and offer more refined control over the outputs, which can be advantageous in certain contexts within the reinforcement learning framework.

3.5. Technologies

The technology foundation of the simulation environment is mainly centered around Python, utilizing the dynamic and flexible features of this programming language. Python's ecosystem,

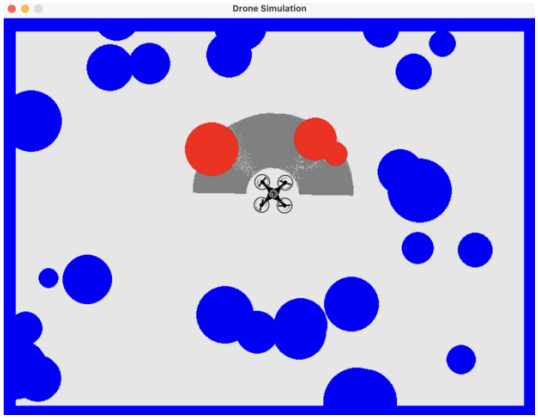

Figure 2: Simulation environment

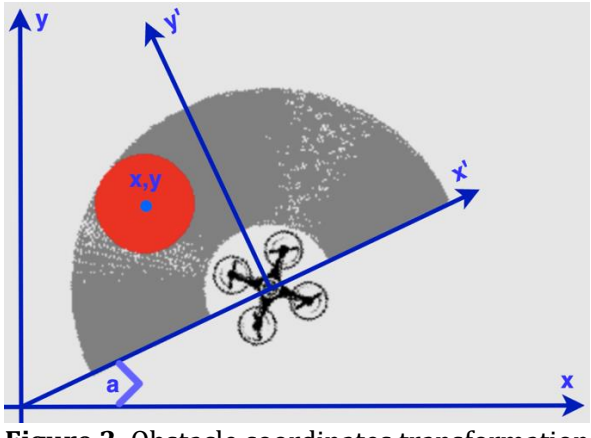

Figure 3: Obstacle coordinates transformation

known for its wide range of libraries and frameworks, supports quick development and prototyping, making it a perfect option for creating detailed simulations.

The graphical representation and interactive elements of the simulation are powered by Pygame [12], a cross-platform set of Python modules designed for writing video games. Its robust functionality and straightforward syntax provide a user-friendly interface for rendering the environment and processing user or AI-driven interactions.

The AI part of the simulation is built on the *stable_baselines*3 [13, 14] library, a set of reliable reinforcement learning algorithms for Python. It provides a variety of ready-to-use neural network policies, such as DQN and PPO, crucial for developing drone navigation models. This library simplifies the use of complex learning algorithms and promotes uniformity and repeatability in the AI training process.

3.6. Simulation environment

The simulation environment, shown in Figure 2, crafted with Python and the Pygame library, offers an interactive 2D space visualized through an 800x600 pixel window. This window is a dynamic arena where the drone interacts with various obstacles. Borders, arranged as squares with each side measuring 20 pixels, frame the simulation space, setting clear boundaries and presenting navigational challenges for the drone. The placement and dimensions of these borders are integral as they influence the drone's route planning and maneuvering strategies. Within these confines, the environment is populated with obstacles, manifested as circles with radii ranging from 30 to 100 pixels. These obstacles are placed to emulate real-world challenges, creating a complex landscape for the drone to navigate. The exact positioning and size of these obstacles significantly affect the simulation's complexity, providing a realistic approximation of navigating through cluttered spaces. The drone, central to this environment, is defined by its 30-pixel radius and equipped with a sensory device. This device, positioned at the drone's core, grants a 120-pixel radius field of view, covering a sweeping 180-degree arc in front of the drone.

The data perceived through this sensory apparatus is pivotal as it informs the drone's decision-making process, enabling it to react and adapt to its immediate surroundings. The sensory interpretation of the environment is defined by translating absolute obstacle coordinates into its drone's frame of reference. This transformation [15] involves (Figure 3):

- translating the coordinate system to align with the drone's current position:

$$\begin{cases} x' = x - x_{drone,} \\ y' = y - y_{drone,} \end{cases} \quad (9)$$

where (x', y') are the obstacle's coordinates after moving the coordinate system, (x, y) are the coordinates before moving, and (x_{drone}, y_{drone}) are the absolute drone's coordinates.

- rotating it to match the drone's heading direction:

$$\begin{cases} x_{\text{obstacle}} = x' \cos(a) - y' \sin(a), \\ y_{\text{obstacle}} = x' \cos(a) + y' \sin(a), \end{cases} \quad (10)$$

where a is the angle by which the coordinate system needs to be rotated, and $(x_{\text{obstacle}}, y_{\text{obstacle}})$ are the obstacle coordinates in the drone's frame of reference.

The next step in this sensory processing is normalization. The drone normalizes distances to obstacles against its field of vision radius while taking into account its own size to accurately measure how close it is to obstacles. The directional vectors, indicating obstacles' positions relative to the drone, are normalized to ensure consistent representation. This normalization ensures that x-coordinates range between -1 and 1, and y-coordinates between 0 and 1, providing a standard format for further data processing. To ensure efficiency and concentrate on relevant obstacles, the drone is designed to process only the first 10 obstacles within its field of view. This approach of selective focus helps the drone prioritize the most immediate obstacles, thereby improving its path planning and decision-making capabilities.

The simulation environment, through its setup, obstacle behavior, and sensory data processing, creates a framework for studying drone navigation methods. The interaction among

the drone's sensory functions, the environment's characteristics, and the neural network models lays the groundwork for this simulation.

4. Experiment

4.1. Simulation environment configuration

Before initiating the simulations, a pre-generated file containing the coordinates and radii of various obstacles. This standardization ensures that all models are trained under consistent conditions, facing the same set of obstacles. The simulation environment is represented as a window, at the center of which the drone is initially placed. To foster a controlled and safe training start, all obstacles are generated while maintaining a safe zone of 50 pixels radius around the drone's initial position.

Each movement of the drone within the environment is referred to as a "step". An "episode" is defined as the sequence of steps from the start of the drone's movement until it collides with an obstacle. Upon collision, the drone is reset to its initial central position, and a new set of obstacles is fetched from the pre-generated file and rendered into the environment. This process constitutes the core of the training loop, allowing the drone to learn from diverse scenarios.

For the experimental setup, the models are configured to run for 1 million steps, which approximates around 10 hours of continuous simulation. Two primary models are under examination: DQN and PPO. Each of these models is run four times, altering the activation function with each iteration. This approach is aimed at exploring the influence of different activation functions on the learning behavior and overall performance of the models.

4.2. Hyperparameters and reward function

The action space for the neural network models is discrete and defined within the range of 0 to 99. Each action value can be decomposed into two components: the turning angle and the distance of movement. The turning angle is calculated as an integer part of the action divided by 10, determining the drone's rotation. If the calculated angle is less than or equal to 5, the drone rotates left by the angle in degrees. For angles greater than 5, the drone turns right by the difference between 10 and the angle degrees.

The turning angle is defined as:

$$angle = \begin{cases} action \div 10, & if\ action \div 10 \leq 5 \\ 10 - action \div 10, & if\ action \div 10 > 5 \end{cases}, \quad action \in [0; 99] \tag{11}$$

The distance the drone moves forward in pixels is determined by the remainder of action divided by 10, allowing the drone to move a distance ranging from 0 to 9 pixels in each step.

The distance is defined as:

$$distance = action\ mod\ 10, \quad action \in [0; 99] \tag{12}$$

The observation space is defined by an array of triplets, each consisting of x, y, and l values. Here, x and y represent the normalized directional vector pointing from the drone to an obstacle, and l indicates the normalized distance to that obstacle. This setup gives the neural network a comprehensive view of the environment, detailing the relative positions and distances of up to 10 obstacles near the drone. This approach ensures that the neural network has a clear understanding of the immediate surroundings, facilitating informed decision-making for navigation and obstacle avoidance.

The observation space is defined as:

$$space = x_1, y_1, l_1, \dots x_i, y_i, l_i, \dots x_n, y_n, l_n, \quad x_i \in [-1; 1], y_i \in [0; 1], l_i \in [0; 1], n = 10 \tag{13}$$

The reward function is crucial in steering the learning journey of the neural networks. It was initially designed to give a penalty of -10 for a collision and a small positive reward of +0.01 times l for each step taken without crashing, with l standing for the distance the drone moved in pixels. However, experimental tests uncovered two problems: the drone sometimes remained stationary, not moving at all, and exhibited "trembling" behavior, which involves making quick,

back-and-forth turns when it was navigating near obstacles. These issues indicated a need to refine the reward structure to encourage more desirable behaviors, such as smooth navigation and avoidance of unnecessary movements or hesitation near obstacles.

To resolve the issues of stationary behavior and excessive turning, the reward system was modified by introducing minor penalties. A penalty of -0.01 is now given if the drone does not move forward, and an additional -0.01 is applied for every turn it makes. This adjustment encourages the drone to keep moving continuously with as few turns as possible, changing its path only when it's essential to avoid collisions. The revised reward function fosters a balance between moving forward, conserving energy by minimizing unnecessary actions, and maintaining safety by steering clear of obstacles. This approach aims to guide the development of optimal navigation strategies that prioritize smooth and efficient movement.

The reward function is defined as:

$$reward = \begin{cases} -10, & if\ \exists i: l_i \leq 0\ (collision) \\ -0.01, & if\ distance = 0\ or\ angle \neq 0 \\ 0.01 * distance, & otherwise. \end{cases} \quad (14)$$

This section has detailed the core components of the simulation, including the action and observation spaces, and the carefully crafted reward function intended to direct the learning algorithms towards effective navigational behaviors. Together, these elements constitute the foundation that supports the neural network models as they operate, learn, and adjust in the simulated drone environment.

4.3. Evaluation metrics

Throughout the simulations, a comprehensive set of metrics will be collected to evaluate the performance and behavioral patterns of the neural network models. These metrics offer insights into the models' efficacy and the drone's interaction dynamics within the simulation environment. Let's consider these metrics.

Average steps per episode reflects the mean number of steps taken per episode, highlighting the drone's efficiency in navigating through the environment. This metric is crucial for understanding how well the drone utilizes its actions to survive in the environment.

Percentage of full stops measures the proportion of actions within each episode where the drone remains stationary, calculated as the ratio of steps without movement to the total number of steps. This metric addresses the stationary behavior issue, with a lower percentage indicating more continuous and dynamic navigation.

Percentage of turning actions quantifies the frequency of turning actions, highlighting the drone's maneuvering behavior. It's calculated as the ratio of steps involving turns (either left or right) to the total steps. This metric is insightful for assessing the drone's adaptability and its tendency to change directions in response to obstacles, aiming for a balance that avoids excessive "trembling" or unnecessary rotations.

Average distance covered per episode represents the cumulative distance covered by the drone in pixels per episode. This metric directly correlates to the drone's ability to progress through the environment, serving as an indicator of effective obstacle avoidance and efficient navigation.

Average speed of movement per step is calculated as the average distance moved per action step, this metric sheds light on the drone's navigational efficiency. It reflects the balance between speed and cautiousness in approaching obstacles, with higher values indicating quicker advancement towards survival objectives.

Average reward per episode calculates the mean reward accumulated per episode, providing a comprehensive measure of the drone's overall performance, including its ability to avoid obstacles, maintain movement, and minimize unnecessary actions.

Continuous monitoring of the simulation process is essential to qualitatively assess each model's behavioral pattern. Observing the drone's movement, its reaction to obstacles, and its overall navigation strategy offers an in-depth understanding of the model's performance and its decision-making rationale.

By systematically analyzing these metrics, we aim to construct a detailed evaluation framework that not only quantifies the models' performance but also provides qualitative insights into the behavioral strategies adopted by the drones under different neural network configurations.

5. Results

Graphical representations of the training process for the DQN model, as shown in Figure 4, reveal distinct patterns in the average reward values across different activation functions. Notably, the Sigmoid function demonstrated a remarkable performance from 300,000 to 600,000 steps, where a sharp increase in the average reward value was observed. This surge, followed by a steep decline, suggests a temporary optimization that did not significantly affect the overall performance, as evidenced by the slight increase in the average reward for Sigmoid -5.666 compared to Tanh -6.982. Despite this, Sigmoid led in terms of the lowest "Percentage of Full Stops" 0.626% and a relatively low "Percentage of Turning Actions" 59.983%, coupled with the highest average speed 4.984, indicating a more efficient navigation strategy compared to other activation functions within the DQN framework.

In contrast, the PPO model's performance significantly, as shown in Figure 5, favored the Tanh activation function, with an average reward of 88.527, nearly double that of the next best function, ReLU. This was mirrored in the "Average Steps per Episode" and "Average Distance per Episode," both approximately twice as high as those recorded for other functions. Interestingly, ReLU and Leaky ReLU scored the lowest in "Percentage of Full Stops," highlighting their propensity for continuous movement. However, the Sigmoid function exhibited the weakest overall performance except in "Average Steps per Episode."

The values of all evaluation metrics for DQN and PPO are detailed in Tables 1 and 2, respectively.

When comparing DQN and PPO models, PPO unequivocally outperformed DQN across all metrics, with even PPO's weakest results surpassing the best of DQN's, indicating a superior navigation and obstacle avoidance capability.

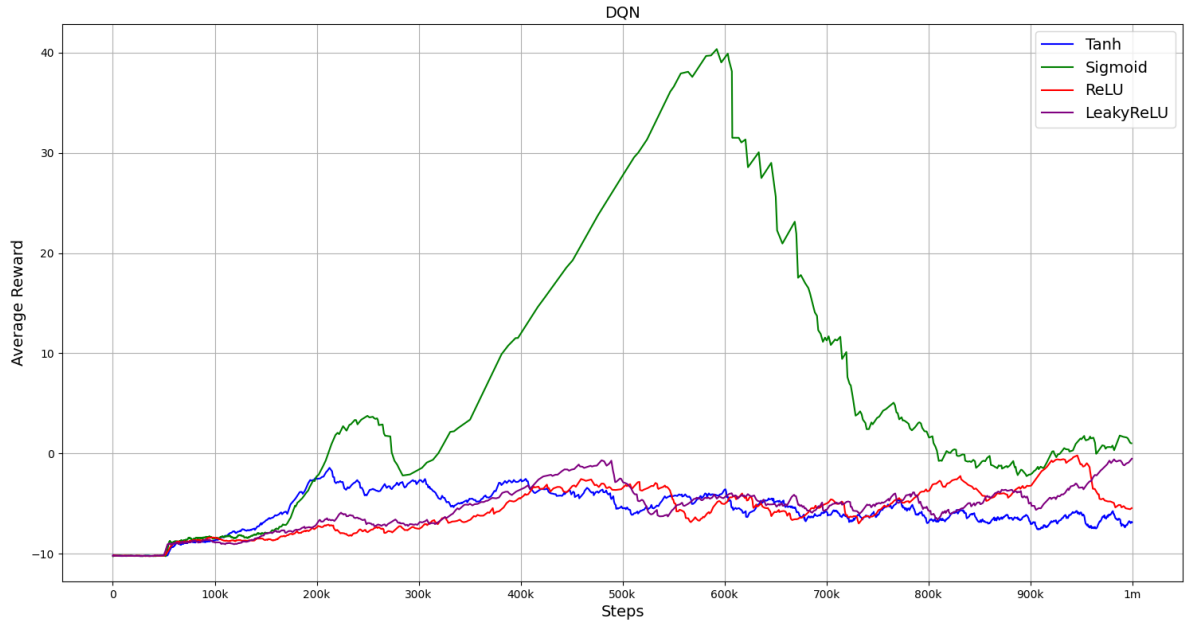

Figure 4: Average reward for DQN

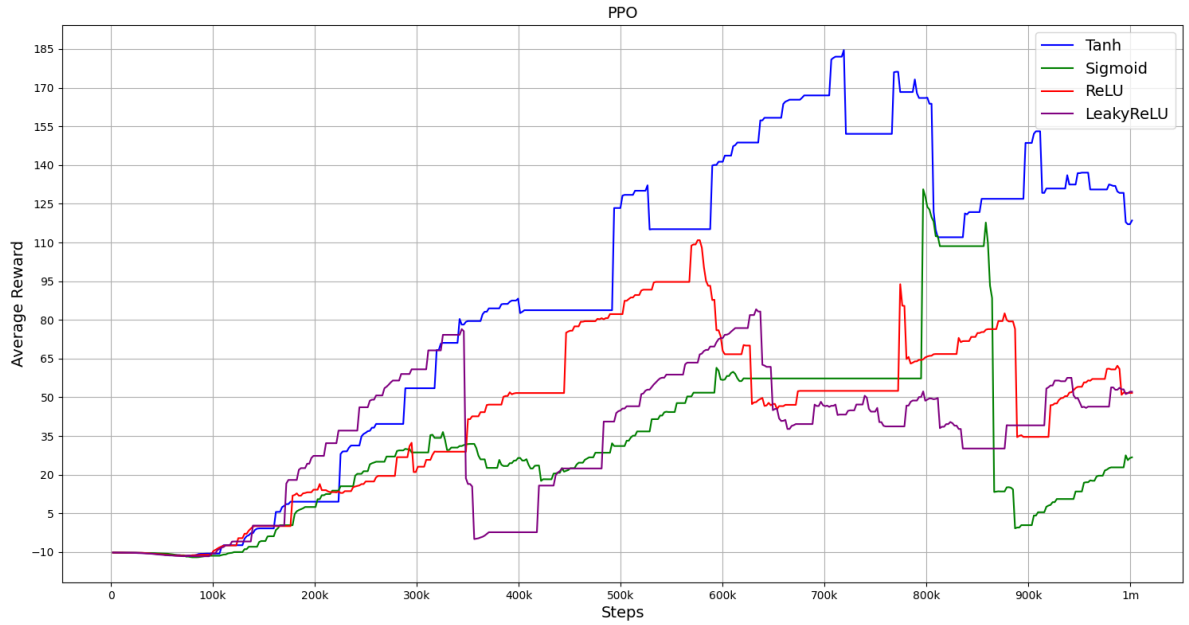

Figure 5: Average reward for PPO

The comparative analysis suggests that bounded activation functions (Sigmoid and Tanh) generally yielded better results than their unbounded counterparts (ReLU and Leaky ReLU), though Sigmoid underperformed in the PPO model compared to ReLU and Leaky ReLU. This discrepancy underscores the complexity of applying a one-size-fits-all approach to activation function selection for specific tasks, indicating that the optimal choice may vary depending on the learning algorithm and the nature of the task.

Further investigation into the models' navigational strategies, particularly in obstacle-rich environments, highlighted significant differences in approach. The DQN model consistently sought to navigate through corridors between obstacles. In contrast, the PPO model preferred to avoid obstacles altogether, moving towards areas with more open space.

This behavior was evident in environments both with and without obstacles, where DQN exhibited a tendency to maneuver closely between obstacles, increasing the risk of collision and consequently resulting in a lower average reward compared to PPO, which opted for safer, open areas, thus explaining the marked difference in performance metrics.

Navigation for behavior DQN and PPO are shown in Figures 6 and 7, respectively.

The results underscore the distinct navigational preferences and risk management strategies of DQN and PPO models in UAV obstacle avoidance, with PPO demonstrating superior performance and efficiency across various metrics. The choice of activation function plays a significant role in model behavior and performance, with no single family of functions emerging as universally superior.

Table 1
Evaluation metrics values for DQN

Activation function	Avg steps per episode	Percentage of full stops	Percentage of turning actions	Avg distance per episode	Avg speed	Avg reward
Tanh	159.434	1.497	62.558	649.456	4.318	-6.982
Sigmoid	**211.367**	**0.626**	**59.983**	**797.7136**	**4.984**	**-5.666**
ReLU	205.576	3.881	67.722	761.555	3.993	-7.616
Leaky ReLU	209.794	2.216	67.46	772.307	4.103	-7.406

Table 2
Evaluation metrics values for PPO

Activation function	Avg steps per episode	Percentage of full stops	Percentage of turning actions	Avg distance per episode	Avg speed	Avg reward
Tanh	**2727.577**	0.13	**50.839**	**12781.262**	4.2	**88.527**
Sigmoid	1538.434	0.149	60.723	5535.878	3.428	31.117
ReLU	1730.587	**0.074**	51.748	8023.028	4.12	43.598
Leaky ReLU	1444.961	0.088	51.174	6546.359	**4.269**	33.8

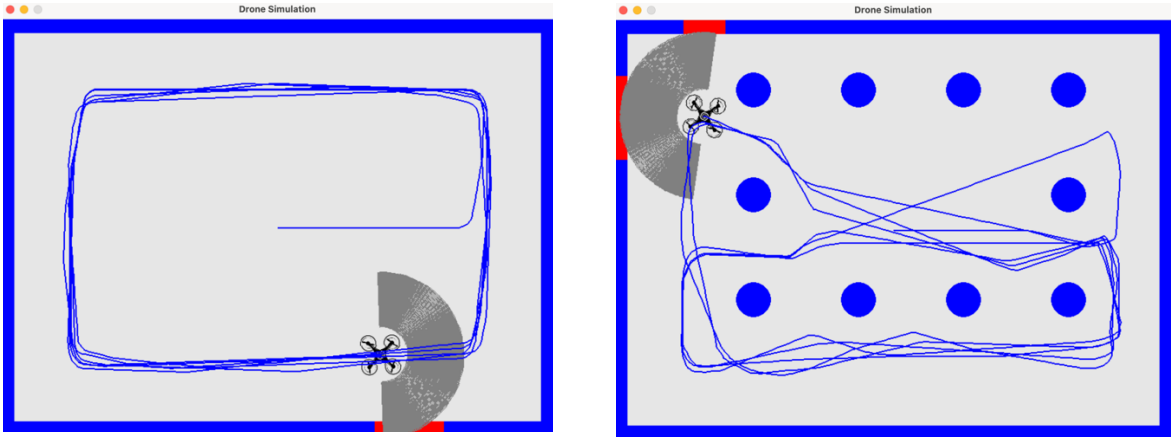

Figure 6: DQN navigational strategy

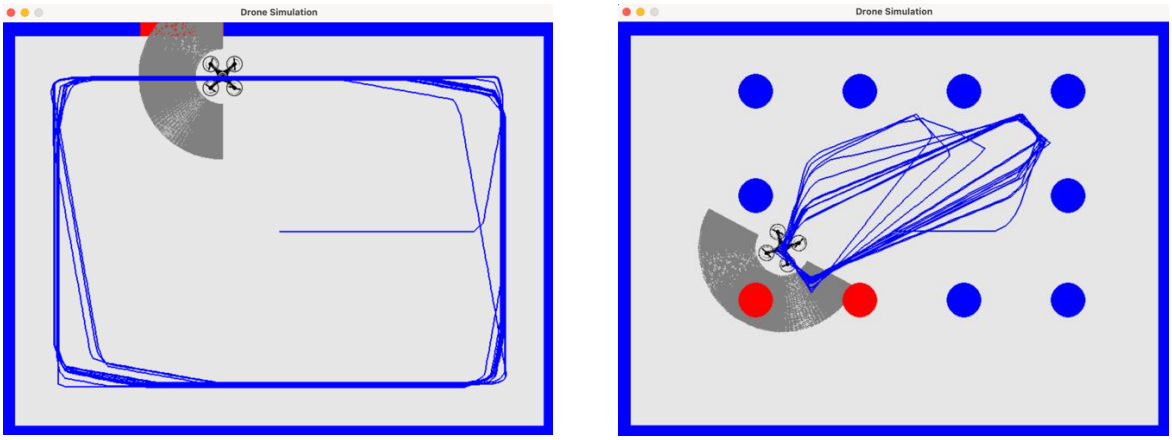

Figure 7: PPO navigational strategy

6. Discussions

Given the comparative analysis of DQN and PPO algorithms for UAV obstacle avoidance in 2D simulations, this discussion synthesizes findings, contextualizes them within broader research, and identifies future directions. The study demonstrates PPO's performance advantage over DQN in 2D environments for UAV obstacle avoidance. However, research [6] in 3D spaces shows DQN outperforming PPO, indicating the algorithm's efficacy may depend on the operational environment's dimensionality. This discrepancy underscores the complexity of autonomous UAV navigation system design and the importance of algorithm selection tailored to environmental characteristics. In 2D simulations, PPO's ability to balance exploration and exploitation contributes to its superiority, a benefit not as pronounced in 3D environments where DQN may offer better adaptability.

The study also highlights the impact of activation functions on learning dynamics, with bounded functions like Sigmoid and Tanh outperforming unbounded ones due to their gradient flow properties, facilitating stable learning. These insights suggest development of UAV navigation systems should consider both the algorithm's suitability for the environment's dimensionality and the choice of activation functions. The contrasting effectiveness of DQN and PPO across spatial dimensions underscores the need for a nuanced approach in algorithm selection.

Furthermore, this study revealed significant differences in how drones navigate when using DQN versus PPO algorithms. DQN drones were more agile, easily weaving through tight spaces and obstacles in enclosed areas. This agility suggests DQN's focus on immediate rewards equips drones with the ability to quickly adapt their paths for effective tight-space navigation. On the other hand, drones using PPO prefer safer, open areas, avoiding the riskier task of navigating through narrow spaces between obstacles. This cautious behavior stems from PPO's strategy of gradual policy updates and a careful balance between exploring new paths and exploiting known ones, aiming to avoid mistakes that could lead to collisions. This observation underscores the importance of choosing the right algorithm for a drone's mission, especially in environments where tight maneuvering is critical. Drones tasked with operating in complex, obstacle-dense areas might benefit from the adaptability of DQN, while those in less cluttered spaces could leverage PPO's emphasis on safety. Understanding these navigational tendencies is crucial for designing UAV systems that meet specific operational needs, providing insights into which reinforcement learning algorithms are best suited for various navigation challenges.

Future research could explore more complex 3D environments, a broader spectrum of reinforcement learning algorithms, and a wider range of activation functions.

7. Conclusions

This research compared DQN and PPO algorithms to understand their performance in UAV obstacle avoidance in a 2D simulation. The study found that PPO outperforms DQN in navigating complex environments, attributed to PPO's efficient policy updates and balance between exploration and exploitation. Additionally, bounded activation functions, such as Sigmoid and Tanh, were more effective than unbounded ones, highlighting the importance of selecting suitable activation functions for improved model performance.

Analysis revealed DQN drones navigate tighter spaces effectively, leveraging immediate reward strategies, while PPO drones opt for open areas, prioritizing safety through cautious policy updates. This distinction underscores the importance of aligning the algorithm with UAV operational needs, guiding system design for specific navigational challenges.

Given these insights, while PPO generally excels in complex environments, DQN's maneuverability in confined spaces presents a compelling case for scenario-specific algorithm selection. This nuanced understanding encourages a balanced approach to choosing algorithms and activation functions, tailored to the UAV's mission profile. Future research should explore broader algorithm applications, integrate diverse inputs, and test these findings in realistic settings to advance UAV navigation capabilities.

References

[1] S. A. H. Mohsan, N. Q. H. Othman, Y. Li, M. H. Alsharif, M. A. Khan, Unmanned aerial vehicles (uavs): practical aspects, applications, open challenges, security issues, and future trends, Intelligent Service Robotics 16 (2023) 109–137. doi:10.1007/s11370-022-00452-4.

[2] K. Arulkumaran, M. P. Deisenroth, M. Brundage, A. A. Bharath, Deep reinforcement learning: A brief survey, IEEE Signal Processing Magazine 34 (2017) 26–38. doi:10.1109/MSP.2017.2743240.

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, M. A. Riedmiller, Playing atari with deep reinforcement learning, CoRR abs/1312.5602 (2013). URL: http://arxiv.org/abs/1312.5602. arXiv:1312.5602.
[4] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, O. Klimov, Proximal policy optimization algorithms, CoRR abs/1707.06347 (2017). URL: http://arxiv.org/abs/1707.06347. arXiv:1707.06347.
[5] K. Reuf, W. Stefan, H. Simon, Deep q-learning versus proximal policy optimization: Performance comparison in a material sorting task (2023) 1–6. doi:10.1109/ISIE51358.2023.10228056.
[6] A. P. Kalidas, C. J. Joshua, A. Q. Md, S. Basheer, S. Mohan, S. Sakri, Deep reinforcement learning for vision-based navigation of uavs in avoiding stationary and mobile obstacles, Drones 7 (2023). doi:10.3390/drones7040245.
[7] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, S. Levine, Soft actor-critic algorithms and applications, 2019. arXiv:1812.05905.
[8] B. Jang, M. Kim, G. Harerimana, J. W. Kim, Q-learning algorithms: A comprehensive classification and applications, IEEE Access 7 (2019) 133653–133667. doi:10.1109/ACCESS.2019.2941229.
[9] D. Zhao, H. Wang, K. Shao, Y. Zhu, Deep reinforcement learning with experience replay based on sarsa (2016) 1–6. doi:10.1109/SSCI.2016.7849837.
[10] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, D. Hassabis, Human-level control through deep reinforcement learning, Nature 518 (2015) 529–533. doi:10.1038/nature14236.
[11] T. Szandała, Review and Comparison of Commonly Used Activation Functions for Deep Neural Networks, 2021, pp. 203–224. doi:10.1007/978-981-15-5495-7_11.
[12] Pygame, 2024. URL: https://www.pygame.org/.
[13] Stable-baselines3, 2024. URL: https://stable-baselines3.readthedocs.io/.
[14] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, N. Dormann, Stable-baselines3: Reliable reinforcement learning implementations, Journal of Machine Learning Research 22 (2021) 1–8. URL: http://jmlr.org/papers/v22/20-1364.html.
[15] Coordinates and transformations, 2024. URL: https://motion.cs.illinois.edu/RoboticSystems/CoordinateTransformations.html/.