Multi-Agent Collision Avoidance Using PPO In Decentralized Reinforcement Learning For Drone Simulated Environment

VOLODYMYR LYSENKO, University of Twente, The Netherlands

The rapid increase of unmanned aerial vehicles (UAVs) in delivery, inspection and emergency-response tasks is pushing air traffic density beyond human supervision capabilities. Ensuring that swarms of autonomous drones remain safe, especially in urban airspace, using collision-avoidance strategies that are decentralized, data-efficient and robust to partial observability.

This research investigates how well decentralized Deep Reinforcement Learning (DRL) policies enable multiple drones to reach assigned goals without collisions when each drone perceives only local, sensor-like information. A custom Gymnasium environment was built to simulate 2-D shared airspace with adjustable traffic density and obstacles number. Proximal Policy Optimization (PPO) served as the baseline algorithm for training the policies which will be responsible for the drone's movement towards the goal and collision avoidance. Trained policies were evaluated on (i) collision-free episode rate, (ii) goal-completion rate, (iii) training sample efficiency, (iv) average steps for episode completion. An optional experimental branch will extend basic state vectors with simulated Light Detection and Ranging (LiDAR)-style observations to quantify the impact of richer perception on learning speed and final performance.

CCS Concepts: • **Computing methodologies → Multi-agent reinforcement learning**.

Additional Key Words and Phrases: Drone Collision Avoidance, Decentralized Reinforcement Learning, Proximal Policy Optimization, Gymnasium, UAV's Collision Avoidance using RL

1 INTRODUCTION

In recent decades, unmanned aerial vehicles (UAVs) have risen in popularity across diverse spheres, including agriculture, logistics, surveillance, military intelligence, and disaster emergency help [4–6, 8, 11–13]. Implementing a stable and reliable collision avoidance system may optimize various domains but requires critical accuracy in urban and populated areas as the number of drones in the same air space increases [4–6, 8, 9, 13]. Despite potential financial losses in drone delivery collision situations, having an efficient, safe, and immutable collision avoidance system can be a life-changing factor during rescue operations within inaccessible locations, such as hillsides with dense forests or unstable buildings after the earthquake. The most problematic part is collision avoidance with static and dynamic obstacles, especially in dynamic multi-agent scenarios where drones operate independently and have limited sensing capabilities [2, 4, 5, 9, 10].

Standard solutions, which involve centralized algorithms that rely on a comprehensive view of the environment, can face limitations such as potential delays in decision-making or high computational requirements, especially in large-scale multi-UAV systems[7, 10]. Additionally, such collision avoidance may be infeasible or used only in limited space [2] since these methods require much environmental data. Therefore, the disadvantages of these methods urged researches in decentralized approaches, where each UAV takes action based on its decision-making system that considers local information about its own state and of its immediate neighbors [6, 7, 10, 12]. However, developing robust decentralized policies is challenging, especially under realistic conditions where drones operate with imperfect sensing and partial observability[3, 10, 12].

Reinforcement learning (RL) and Deep Reinforcement Learning (DRL) have been presented as a potentially best solution that teaches autonomous agents to act appropriately in programmatically complex tasks, such as navigation and collision avoidance[1–5, 8, 11, 12, 14].

This research aims to investigate the effectiveness of decentralized reinforcement learning policies for multi-drone collision avoidance within a simulated environment developed using the `Gymnasium` RL framework. Each drone will operate as an independent agent, perceiving its surroundings via a limited observation space. Additionally, the paper will estimate how the usage of sensor-like retrieval mechanisms impact on the policy performance using the `LIDAR`-style distance measurements and object classifications.

2 PROBLEM STATEMENT

In the last few years, large language models (LLMs) AI agents have significantly changed the educational process, information exchange, and human-computer interaction. Similarly, the deployment of UAVs across a wide variety of fields, such as package delivery, surveillance, environmental monitoring, and fire prevention systems, potentially creates another revolutionary technological shift[3, 6, 10].

The next revolutionary step could be connected to autonomous aerial navigation, which will lead to drone usage for commercial or distribution purposes. Increasing density in the airspace, particularly as operations move into urban and populated areas, creates a need for a safe and reliable system that will circumvent other obstacles[5, 6, 10]. Traditional approaches to collision avoidance can be computationally complex and may face scalability issues in large multi-drone systems[10].

These limitations motivate research decentralized approaches, where each drone makes decisions using local information[3, 6]. However, full-scale massive functioning in real-world environments means drones often experience partial observability[3, 6], which makes designing effective decentralized collision avoidance policies challenging[12]. Using RL and DRL approaches, autonomous agents can be trained to deal with dynamic and uncertain environments[8, 9, 11, 14]. Due to financial and safety measures, developing and testing such autonomous systems frequently requires simulated environments[1, 10, 13].

Author's address: Volodymyr Lysenko, University of Twente, Enschede, The Netherlands.

TScIT 43, July 4, 2025, Enschede, The Netherlands

Although the following topics have already been discussed in some research papers, these papers do not focus on the efficiency of PPO in decentralized approaches. The existing papers do not reveal promising or validated results. Factors like efficiency can be crucial when talking about mass production deployment or designing a system for a variety of environments.

2.1 Research Question

Main Research Question: How effective are decentralized RL policies trained by PPO using for multi-drone collision avoidance under partial observability conditions?

Extension Research Question: How does the use of simulated sensor perception, for instance LIDAR-based observations, affect the effectiveness of decentralized multi-agent reinforcement learning policies for drone collision avoidance in simulated environments?

3 RELATED STUDIES

3.1 Observation Models and Partial Observability

Environment observation is one of the core parts of autonomous drone navigation. Various studies outline the importance of local and partial observations to enable decentralized operations. Drones are often equipped with LiDAR or other sensors to collect positions about other drones within their sensing range[6, 12].
Standard sensor types and feature extraction include:

Range Sensors: It is often crucial to explore the distance to nearby objects to overcome them safely. Commonly, these aims are fulfilled with LiDAR and ultrasonic sensors, which allow drones to detect obstacles and the closest distance to them precisely.

Visual Inputs: RGB-D cameras are often used to capture the colour and depth images simultaneously[8–10, 14]. This helps in the detection of surrounding objects as well as the distances to them.

Scalar Data: Angle to goal, elevation angle, geo fence (virtual barrier), and preferred velocity features are often combined with visual or range data to achieve the state representation for the learning agent[12, 14]. The concept of a "mixed state" combining image and scalar data is also explored to enhance learning efficiency[14].

Agent-Level States: Many decentralized approaches that provide stable collision avoidance often extract observable states of other agents before performing a collision avoidance decision, for instance, velocities positions and other metrics of other drones rather than raw sensor data[7, 12].

3.2 Reinforcement Learning Algorithms

Deep Reinforcement Learning (DRL) has gained promising results for multi-UAV collision avoidance due to its ability to learn complex behaviours in dynamic, unfamiliar environments through trial and error, eliminating manual data labelling and hard-coded rules[4, 5, 11]. Key RL algorithms in the literature can be grouped into two categories:

1. Policy Gradient Methods

Proximal Policy Optimization (PPO): A widely used policy gradient method, often used for deep RL when the policy network is very large. Algorithm is widely known for its stability,and suitability for continuous state and action spaces, which successfully fits the research objectives. The main advantage of PPO is prevention of huge policy changes which positively affects to training stability.

Deep Deterministic Policy Gradient (DDPG): An off-policy actor-critic algorithm designed for continuous control problems.

Soft Actor-Critic (SAC): Another off-policy actor-critic algorithm that has demonstrated strong performance and sample efficiency, particularly in 3D environments with dynamic obstacles.

Trust-Region Policy Optimization (TRPO) and ACKTR: Other popular policy gradient methods used for continuous control.

2. Q-Learning

Another standard algorithm for DRL drone training is Q-Learning, and its variations. Q-learning is commonly used to create stable and efficient path-finding/route-planning systems. The well-known variations are:

- Q-Learning
- Deep Q-Network (DQN)
- Double DQN (DDQN) and Dueling DQN

3.3 PPO

In general, PPO operates on an actor-critic architecture, utilizing two Convolutional Neural Network (CNN) models,specifically the `Actor`, which determines the policy, and the `Critic`, which evaluates the state-value function[4]. This research uses PPO with multilayer perceptron (MLP) policies based on fully connected networks, which are more appropriate for structured vector-based observations. As an on-policy learning approach, PPO updates its decision-making policy based on a small batch of experience collected directly from interactions with the environment[4]. After the policy has been upgraded, these observation are dumped and batches are refilled with new observation data for new updates[4]. A comparative study assessing PPO against Deep Q-Networks (DQN) and Soft Actor-Critic (SAC) for obstacle detection and avoidance in UAVs found that PPO performed poorly, particularly in large 3D environments with dynamic actors[4]. This research also aims to explore and validate this negative characteristics of PPO.

4 METHODOLOGY AND APPROACH

This section outlines the methodological approach adopted to develop, train, and evaluate DRL policies, specifically focusing on the PPO algorithm and LiDAR-style observation for multi-agent drone collision avoidance in a simulated environment. The methodology encapsulates environment design, agent observation modeling, training pipeline configuration, reward shaping, hyperparameters tuning, and evaluation procedures.

Initially, the main approach was focused on building a custom 2-D environment with varying number of agents and obstacles, where agents can be trained together by PPO algorithm. Further, it was planned that the best policies will be evaluated based on **1000** random episodes, providing the data for the chosen figures of merit.

Throughout the experimental training process, it was observed that policies trained without rich data perception, such as partial observation with mainly proximity-only features, struggled to converge to robust and generalizable behavior. Drones often failed to reach their goals or demonstrated poor obstacle avoidance. Thus, the simulated LiDAR-based observations were implemented to achieve more stable and efficient results. Combining proximity and LiDAR-based observations illustrated better results in safely overtaking nearby obstacles and agents, enabling more informed decisions in a partially observable setting.

The decentralized control approach was chosen to mimic realistic deployment scenarios where inter-agent communication is limited or costly. The decentralized system will allow different companies to deploy their drones without configuring the shared traffic protocols, which saves funds and effort and causes companies to rely on their stability. Each drone acts independently based on its local sensory input.

In order to test the performance and stability of policies trained by PPO, the custom 2D Gymnasium-compliant environment was built. The environment simulates a square arena with variable numbers of drones and static obstacles. The simulation environment supports multiple variations of observation spaces, configurable traffic density, variable spawn area, and physical constraints that simulate real-world UAV dynamics.

Because `Stable-Baselines3`'s PPO was created for one agent in the environment, the initial Gymnasium environment was wrapped in custom `PettingZoo ParallelEnv`. `ParallelEnv` separated each agent's observations, rewards and other information in several directories, by which solved the compatibility issue. Additionally, the final training pipeline contains several `SuperSuit` wrappers that interchange `ParallelEnv`'s format for compatibility and efficiency.

Lastly, the vectorized parallel environments with agents are trained across a staged curriculum using the `Stable-Baselines3`'s PPO implementation. The reward function was carefully shaped to incentivize goal-directed movement, penalize collisions and inefficient paths, and encourage smooth trajectories.

4.1 Simulated Environment

The environment was implemented using the `Gymnasium` framework and extended with `PettingZoo` and `Supersuit` to support `Stable-Baseline3` multi-agent vectorized training. A continuous 2D layout was selected to balance realism and computational efficiency, providing scalable experimenting with real-world-like physical interaction within the given time limit.

4.1.1 ***World Model.*** The simulated world is 100m × 100m, recalling a realistic operational scale for small drone tasks in urban or industrial scenarios, such as package delivery or rescue operations. Time is discretized at 10 Hz to provide smooth motion dynamics.

All unit calculations and distances are expressed in the metric system. Physics updates follow a fixed time-step integration scheme. Specifically, every action an agent takes at a specific time step is multiplied by a constant factor of 0.1, which helps to simulate real-world physics.

For instance, if the agent was moving with velocity (5,0) and at a specific step, the agent takes action (-2,0) to perform an urgent brake, then the agent will end up having a velocity of $-2.0 \times 0.1 = -0.2 + 5.0$ resulting in (4.8, 0) after that step. Such step application efficiently simulates real-world physics momentum and urges the model to learn long-term steady braking.

4.1.2 ***Agent Dynamics.*** Agents are modeled as circular discs with a static radius of 0.5 m. Integration is performed using a basic forward Euler method, which offers sufficient stability given the 10 Hz update rate and controlled dynamics.

4.1.3 ***Obstacles and Goals.*** Static obstacles are included to simulate environmental imperfection that can be met in the real world daily. Each episode initializes with a configurable number of circular obstacles with randomized positions and radiuses sampled between 1.0m and 3.5m. This variability forces agents to learn robust and dynamic avoidance strategies. Each agent is assigned a unique goal location, either symmetrically mirrored from its initial spawn point or randomly generated with a minimum distance of 15 meters from the agent's spawn point.

4.1.4 ***Observation Space.*** As it is depicted in the table, Each agent operates under partial observability and receives a limited observation vector to mimic decentralized, sensor-driven control. The base observation can be seperated in 3 sections:

- **Kinematics:** agent's velocity (2 floats), vector and angle to the goal (3 floats)
- **Environment awareness:** vectors and distances to the nearest obstacle, agent, and arena border (11 floats)
- **Optional perception enhancement:** a LiDAR-style input of 24 rays evenly spaced 360° around the agent, each returning obstacle distance up to 20m (24 floats)

Earlier iterations without LiDAR observations failed to produce reliable behaviors in complex scenarios. Including LiDAR dramatically improved training stability and final policy performance, confirming its necessity in complex environments where spatial reasoning is critical.

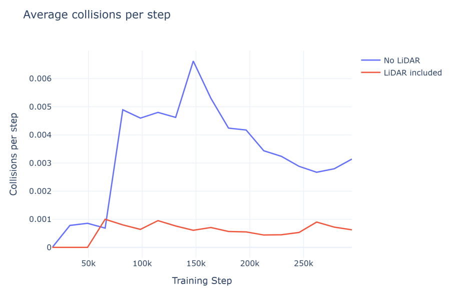


Fig. 1. Average collisions per step during the training with LiDAR and without. The X-axis represents training time steps, whereas Y-axis describes the number of collisions per step based on the rollout avarage. Number of collisions per rollout can be rougly calculated by mutiplying graph's value by $N\_Envs \times N\_Steps$.

Figure 1 depicts two curves of identical training setups where the orange line represents the training setup with LiDAR perception and blue without. The metrics showed on graph are the average collisions per step, which are calculated by $collision\_steps \div total\_rollout\_steps$. This way the graph's average metrics can be roughly converted back too explore the difference in number collision steps between these two setups. The blue curve which corresponds to No-LiDAR observation stays on values from `4e-3` to `5.5e-3`. These values can be multiplied by the number of steps per rollout, specifically 512 steps in 16 environments, which results in $5 \times 10^{-}3 \times 512(steps) \times 16(envs) = 40.96$ collisions happened during that rollout. Alternatively, the Lidar observation scales down the number of collisions to 16.3(calculated by same operations), which makes usage of LiDAR strongly positive. The LiDAR observation stably outperformed other training setups without it, especially in high-density scenarios. Logically, the lowest difference in comparing LiDAR vs No-LiDAR setups happened during training 1 agent in environment. Overall, as density of the airspace becomes larger, the LiDAR presented more advantages, therefore it was chosen to stick with evaluating LiDAR-including models.

4.1.5 ***Observation Normalization***. Additionally, after several training round iterations it was found that training could be optimized and achieve better results while using normalized observation. Therefore, observation features are normalized and clipped to [-1.0,+1.0] or [0.0,+1.0]. For instance, distances are normalized to values [0.0,+1.0] by division of distance by arena size, whilst velocity is, logically, clipped to [-1.0,+1.0] by division on max velocity allowed.

4.1.6 ***Environment Specifications***. Table 7 in Appendix B represents a detailed overview of the built simulator's specifications.

4.2 Implementation

During this research, the project with the simulated environment was built using Python and Gymnasium API for training multi-agent drone collision avoidance and extended with multi-agent support via PettingZoo and vectorized training utilities from Supersuit. This section outlines the core components of the implementation.

4.2.1 ***Gymnasium Custom Environment***. The Gymnasium API provides a platform for creating simulation environments compatible with most RL libraries. The base class `DroneWorldEnv` of the custom environment follows the standard `gym.Env` interface and defines the drone dynamics, obstacle generation, observation model, reward structure, episode termination logic, rendering features, etc. Every custom environment built using Gymnasium must subclass `gym.Env` and implement the core methods: `__init__()`, `reset()`, and `step(action)`. Optionally, methods such as `render()`, `close()`, and `seed()` can be implemented to support visualization, cleanup, and reproducibility. The implemented solution uses the Gymnasium v1.0+, which supports both human and off-screen rendering modes using Pygame. Whilst the human rendering mode implementation is primarily used when debugging or discovering the agent's behaviour, the non-human `"rgb_array"` rendering mode is convenient for automatically generating videos of specific episodes The environment implements all core methods required by the `Gymnasium API`, specifically the methods and their purposes are:

`__init__()`: Sets up the environment configuration, such as the number of drones, obstacle count, maximum steps, action/observation space definitions, rendering options, and dynamics parameters such as velocity, acceleration and spawn area limits. The action space is defined as a `Box(2,)` representing 2-D accelerations, while the observation space is a flat vector of 16 to 40 floats, depending on whether LiDAR is enabled.

`reset()`: Initializes all episode components, including drone and obstacle positions, drone radiuses, velocities, and per-agent goals. For convenience, the method allows resetting the environment with a specific seed. Goals are spawned after drones and are mirrored in relation to the drone's spawn point or randomly sampled to sustain a minimum 15-meter distance from the agent's spawn point. The obstacles are randomly positioned with some margin around the drone's spawn point, goal, arena walls and other obstacles.

`step(actions)`: Applies the 2-D acceleration vectors to the velocity vector via the internal method `_apply_physics()`, while following proper speed limits, detecting collisions with walls, obstacles or other drones, and calculates individual rewards through `_compute_reward()`. Each agent receives an updated observation, scalar reward, done flag, and additional info in line with the Gymnasium API format.

`render(mode)`: Provides real-time visualization via Pygame, showing drones, goals, obstacles, velocities, rewards, and optional LiDAR beams. It supports both interactive ("human") and frame-buffered ("rgb_array") rendering.

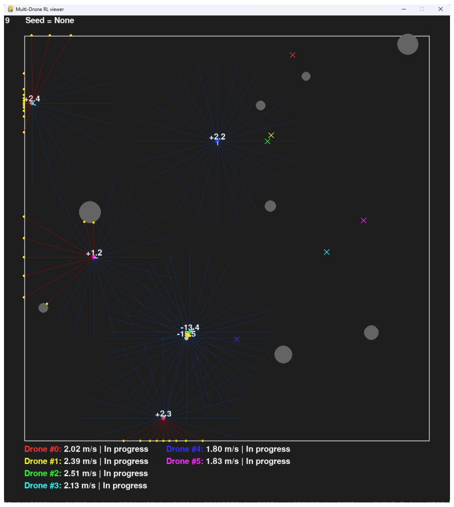
`close()`: Shuts down the Pygame display.

The Figure 2 demonstrates the rendered environment with 6 drones and 8 obstacles. The drones are represented as colorful discs, whilst the obstacles are gray circles of different sizes. In the beginning, each agent is assigned a color which is used to visually differentiate drones and their goals. Each agent is equipped with tiny white line which scales depending on the speed and illustrates where the agent is currently heading. As depicted, on top of every agent there is a number which represents the reward for the specific agent in current step. The blue and red lines around the agent show LiDAR observations in real time. These blue beams become red when the object or other drone is detected in that beam. For debugging and observing purposes the simulator was equipped with status bar representing the state and velocity of each drone. Additionally, the number in the top left corner prints the current time stamp and displays seed of the episode.

4.2.2 ***PettingZoo Adapter***. To enable multi-agent reinforcement learning (MARL) using the Stable-Baselines3-compatible training stack, the base environment was wrapped using the `PettingZoo` library in `ParallelEnv` mode. The adapter class, named _DronePZ, acts as a lightweight interface between the vectorized Gymnasium environment and the PettingZoo ecosystem, enabling seamless multi-agent interactions and batched agent updates.

The adapter conforms to the `ParallelEnv` interface, which allows simultaneous agent decisions and avoids the step-by-step alternation used in PettingZoo's `AECEnv`. This parallel mode is particularly suitable for environments where all agents operate synchronously, as in centralized training with decentralized execution (CTDE).

Design and Structure: The DronePZ class holds the custom `DroneWorldEnv` that was built on base of `Gymnasium gym.Env` as its internal world model and interprets the existing `Gymnasium` drone environment data in `PettingZoo`-compliant style. The detailed description of methods can be found in Appendix A.



```
Drone #0: 2.02 m/s | In progress    Drone #4: 1.80 m/s | In progress
Drone #1: 2.39 m/s | In progress    Drone #5: 1.83 m/s | In progress
Drone #2: 2.51 m/s | In progress
Drone #3: 2.13 m/s | In progress
```

Fig. 2. Rendered screenshot of simulated environment with LiDAR observations.

Agent Management and Spaces: The adapter manages a static list of possible agents via `self.possible_agents`, with individual entries like `"drone_0"` through `"drone_n"`. Active agents are tracked using `self.agents`, which is updated on every `step()` call to remove drones that have finished their episodes. Observation and action spaces are gained from the inner `DroneWorldEnv` world class.

Reward and Episode Metadata: At each step, if the environment reaches a terminal state (e.g., all drones are done or a time limit is exceeded), the adapter propagates cumulative metrics such as `collision_count`, `goals_reached`, and `wall_hits` to every surviving agent's `info` dictionary. This allows evaluation and training scripts to measure performance of certain episodes based on the important metrics.

Auxiliary Controls: In simple training scenarios, for instance 2-3 drones without obstacles, the whole arena space of 100m may be too big. Trajectories of the moving agents may not intersect, which urges to learn only go to the goal strategy throughout this episodes. To solve this issue and optimize the training, an additional method `change_spawn_area()` was implemented. This method allows dynamic modification of the environment's spawn area during training. This feature enables adjusting the density of the aerial traffic when training small number of agents.

4.2.3 ***Supersuit Training Wrappers***.

In order to use `Stable-Baseline3` vectorized RL algorithms on the developed environments in stable and efficient training settings, a series of wrappers from the `Supersuit` library were applied. These wrappers apply necessary environment transformations, which help improve training throughput and ensure compatibility with `Stable-Baselines3's` VecEnv-based training interface.

In short, wrappers are responsible for providing stable input for training, separation on `N` environments for better throughput, displaying the statistics and normalization of the raw data. Table 1 represents the specific order of wrappers that are applied to `DronePZ` environment before training. The detailed description of the wrappers' functionalities and purposes can be found in Appendix A.

Table 1. List of environment wrappers

#	Wrapper Name
1	`black_death_v3()`
2	`pettingzoo_env_to_vec_env_v1()`
3	`concat_vec_envs_v1()`
4	`VecMonitor()`
5	`VecNormalize()`

This wrapper pipeline ensures:

- Compatibility between PettingZoo environments and Stable-Baselines3
- Correct handling of terminated agents without breaking vector shapes
- Parallel training over multiple environment copies to increase sample efficiency
- Stable learning through standardized observation inputs

4.3 Training Curriculum

Initial experiments demonstrated that direct training of agents in high-density scenarios with many drones and obstacles often led to poor policy results or required more training time steps. Separate PPO models were trained per phase, each with a fixed number of agents. Every phase is dedicated to training the "N" number of drones. However, all the stages of the specific phase are trained in one run, such that more complex training stages inherit the policy behavior from simple ones. Rather than giving the PPO complex scenarios with high drone-obstacle density, the staged curriculum allows the policy to learn fundamental skills such as directional control and goal alignment by starting with simpler episodes[7, 10, 12]. These abilities were gradually refined as new challenges were introduced in later stages.

Table 8 in Appendix B represents approximated stage timestep values under which the training has achieved performant results within time-efficient training. Depending on the needs, some stages were shortened or widened to achieve better training and evaluation results.

4.4 Hyperparameters Tuning

Training decentralized drone navigation policies with PPO requires carefully selected hyperparameters to ensure convergence, stability, and generalization. The tuning process took a long time via iterative experiments during early testing stages, where learning dynamics were more stable and observable. Each final hyperparameter was evaluated based on its impact on training metrics and overall performance.

Table 2 shows the rough estimate of the best-fitting PPO hyperparameters found during this research. The depicted hyper parameters illustrate the most performant configuration for training 4 drones. Detailed description of the reasoning beyond key hyperparameters can be found in Appendix A.

Table 2. Best-fitting PPO hyperparameters

Hyperparameter	**Value**
Batch size	2048
# Environment steps per update	512
# Parallel environments	16
Discount factor γ	0.99
Policy net architecture	[512, 256, 128]
Value net architecture	[512, 256, 128]
Learning rate	3×10^{-4}
Optimization epochs	6
GAE λ	0.95
Clip range	0.25
Entropy coefficient	0.008
Value loss coefficient	0.5
Max gradient norm	0.5

The Figures 3, 4 and 5 illustrate the comparison between two different hyperparameters configuration for two drones training. For illustration and comparison purposes the values are smoothed by `0.8`, which allows us to track trends and notice huge differences. The blue line illustrates tuned params, whereas orange represents the initially chosen parameters. As depicted on the graphs, the final parameters choice provides more stability in reaching the goals and all drones reach their goals more frequently. In ideal training, we should expect 32 reaches per rollout, as we are training 2 drones using 16 different environment which gives $16 \times 2 = 32$ drones aiming to reach their goals per PPO iteration. Despite more stable collision avoidance presented in Figure 5, the initially chosen parameters illustrate the wrong behavior tendency in two other figures. Specifically, because initial batch size was small the model did not experience the required variability to explore that reaching destinations results in greater profit rather than just surviving. The final parameters result in more stable goal-reaching with values varying from 29 to 32 compared to 15 to 25 with initial params. The episodes end faster, namely length vary from 240 to 315 which illustrates a healthy behavior compared to 400-500 outputted by initial configuration.

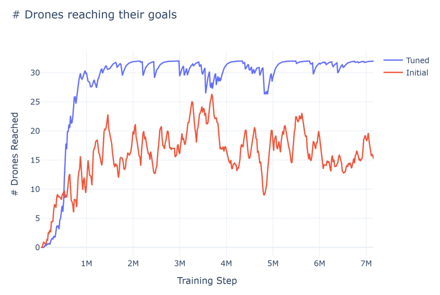


Fig. 3. Drones reaching goal statistics during the training with two different hyperparameters configurations. The X-axis represents training time steps, whereas Y-axis describes the number of drones that achieved their goals in that rollout. Training is done on 2 drones within 16 environments.

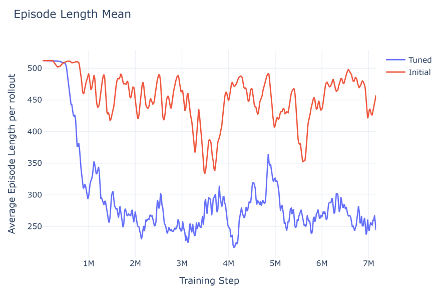


Fig. 4. Average episode length during the training with two different hyperparameters configurations. The X-axis represents training time steps, whereas Y-axis describes the average length of the episodes in that rollout. Training is done on 2 drones within 16 environments.

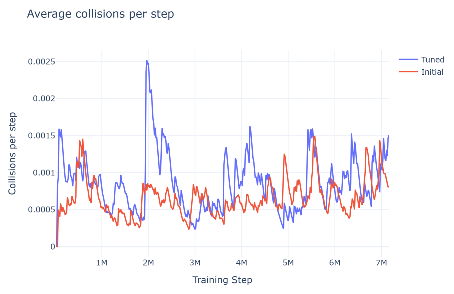


Fig. 5. Average collisions per step during the training with two different hyperparameters configurations. The X-axis represents training time steps, whereas Y-axis describes the number of collisions per step based on the rollout avarage. Number of collisions per rollout can be rougly calculated by mutiplying graph's value by $N\_Envs \times N\_Steps$. Training is done on 2 drones within 16 environments.

4.5 Reward Function Shaping

Throughout the experimental process reward system was remodeled and rescaled many times in order to achieve better results. The values of the coefficients and thresholds were altered, depending on the simulated traffic density and number of agents. The used reward coefficients can be categorized into three groups, namely, terminal rewards, distance thresholds and reward coefficients.

The first group is responsible for punishing or rewarding agents for the events that either end the episode or should provide a stable punishment. When one of these events happen, only terminal reward value is outputted to the system. For example, at the collision step the agent will only receive `-30` and no further coefficient logic affect on the reward. Table 3 demonstrates the values used for the rewarding terminal events.

The second category's duty is defining thresholds for distances where agents will be punished or motivated in order to achieve certain actions. The safe and dangerous radii are responsible for agent and obstacles repulsion, meaning that agents are punished within this areas. The repulsion punishment progresses linearly, thus the agents receive lower rewards as they get closer to another drone or obstacle. The final mile, brake and align radii are responsible for additional shaping terms which teach agents smoothness and real-world behavior, such as braking, steering to goal and keeping normal speed. Lastly, the minimum push speed was created to eliminate too low speed issues. Table 4 demonstrate the values for the area partition under which the PPO performed best.

The third type of rewards are used for continuous actions. Progress coefficient is used to award agent for coming closer to the goal, whereas step cost and action penalty are used for penalizing extra steps and sharp actions. The aligning, braking and repulsion coefficients are used for scaling the penalties applied for violating the safeness and smoothness constraints. Specifically, the difference between maximum allowed distance (defined by radii) and the actual distance to obstacle or goal is multiplied by these coefficients to achieve the wanted effect. Logically, zero speed penalty is applied when the agent either has extremely low velocity or stopped in place where it was not required. Lastly, the away coefficients are responsible for motivating agents to stay away from other agents and obstacles. Table 5 shows the best performing coefficients that are used in the current reward function model.

Table 3. Terminal Rewards and Penalties

Event	Reward / Penalty
Success (reach goal)	+150.0
Collision or wall-hit	−30.0
Episode truncation	−300.0

Table 4. Tunable Distance Thresholds

Threshold	Value
SAFE_DIST	15.0
DANGEROUS_DIST	8.0
FINAL_MILE_R	15.0
BRAKE_R	9.0
ALIGN_R	15.0
MIN_PUSH_SPEED	0.5

Table 5. Core Reward Coefficients

Coefficient	Value
PROG_COEF	6.5
STEP_COST	0.01
ACTION_PEN	0.005
ALIGN_COEF	0.5
BRAKE_COEF	2.0
REP_COEF	4.0
ZERO_SPEED_PEN	1.0
AWAY_DRONE_COEF	2.5
AWAY_OBS_COEF	1.0

The detailed description of how the reward coefficients and constants depend on the reward output can be found in Appendix A.

5 RESULTS & FINDINGS

5.1 Figures of Merit

In order to evaluate the performance of different models and introduce final results, the paper outlines four figures of merit.

- Collision-free episode rate
- Goal-completion rate
- Training sample efficiency
- Average time steps to complete the episode

These metrics were specifically chosen to outline the complexity of defining a policy which balances between successfully reaching the goal and avoiding collision. During the research it was discovered that keeping equilibrium between avoidance and goal completion is extremely challenging task, because if the penalties are too severe the agents are afraid to make risky actions, which frequently result in almost stall behavior. Alternatively, if the penalties are too soft, the agents are neglecting safety and follow strategy to strongly reach the goal even when it requires bumping in obstacle.

5.2 Results

Table 6. Training results for varying numbers of drones and obstacles

Dr.	Obs.	# Steps Req	Completion	Coll-free	Avg Ep. Length
2	0	1,200,000	100.00%	94.20%	257.53
2	4	2,200,000	99.30%	87.20%	289.84
2	8	2,200,000	97.40%	85.70%	322.27
2	10	2,500,000	94.60%	80.20%	379.35
4	0	2,000,000	99.50%	91.30%	315.31
4	4	2,500,000	95.60%	86.30%	349.76
4	8	3,000,000	91.80%	83.70%	380.13
4	10	3,500,000	89.70%	79.70%	396.43
6	0	3,500,000	98.70%	82.40%	344.00
6	4	4,000,000	93.90%	79.10%	396.21
6	8	4,000,000	89.60 %	74.20%	467.53
6	10	4,200,000	83.40 %	69.20%	485.32

Table 6 illustrates the best results achieved during all testing phases throughout this research. The table represents approximated number of steps under which PPO converges stable strategy. The further training with higher time steps values does not produce dramatically better results. The table also provides the completion and collision-free episode ratios.

5.3 Reflection on Results

The results in Table 6 reveal several trends in PPO operation under increasing multi-agent complexity. Across all drone number setups, adding obstacles steadily decreases both goal-completion and collision-free rates. For instance, with two drones the success rate drops from 100% in obstacle-free space to 94.6% when 10 obstacles are present, and the collision-free rate falls from 94.2% to 80.2%. This outlines that the learned policy struggles more with obstacle avoidance as the workspace becomes crowded. Even in empty environments, larger teams require more training steps to reach high performance 2 drones converge to near-perfect success in 1.2 M steps, 4 drones in 2.0 M, and 6 drones in 3.5 M. This indicates that the state-action space grows rapidly with team size, slowing PPO's sample efficiency.

As obstacles or agents increase, the average episode length tends to increase, for instance from 257.5 to 379.4 time steps for two drones as obstacles go from 0 to 10. In longer episodes, agents tend to coordinate more carefully to avoid collisions—but they do not fully compensate, since collision-free rates still decline.

Despite the initially high expectancy, PPO illustrated promising but not perfect results in resolving continuous tasks in complex multi-agent environment. As indicated in the related papers, PPO's main problem is that its strict reliance on an on-policy rollouts results in very poor sample efficiency and slow convergence in high-dimensional, multi-agent collision-avoidance tasks, making it impractical for training in high-density environments [4].

Overall, PPO demonstrates robust baseline capability for multi-agent collision avoidance, but its efficiency and final safety performance decreases as both drone and obstacle density grow. While PPO is a popular on-policy method for single-agent tasks, several disadvantages make it a less valuable choice for solving complex multi-agent collision-avoidance scenarios

5.4 Limitations

In this research project PPO treats the multi-agent system as a single large Markov Decision Process (MDP). PPO does not explicitly decompose rewards or actions per agent, thus it does not distinguish separate drone's contributions from the final outcome. Consequently, the PPO treats the system as one single decision-making problem, where positive actions of one drone can be neglected by strongly negative actions of another.

The neural policy network in PPO must encapsulates both: perception and coordination logic for all drones simultaneously[1, 4, 13]. When the traffic density increases, the network either has to grow or becomes a bottleneck, leading to negative impact on success and collision-free rates.

Lastly, the lack of experience in RL, ML and UAV's spheres, together with the University's time constrains, cause a significant impact on the research outcome. The author of this research has gained all required knowledge throughout the experimental and research processes. The research would have produced more accurate and better results if the author had a ready-to-test Stable-Baseline-3 compatible simulator, or had prior knowledge about the drones functioning and Reinforcement Learning.

5.5 Future Improvements

This research could be improved upon iterative reward function improvement and further hyper parameters tuning. The current setup allows building complex training curriculum with a lot of configurable variables.

The built simulator is fully compatible to most algorithms available in Stable-Baseline-3. Therefore, the following simulator can be used as baseline tool for testing performance of other RL algorithms in order to achieve stable collision avoidance. Additionally, the simulator can be reworked to simulate and train models for complex scenarios in 3D environment, which will result in producing more relevant and realistic results.

6 CONCLUSION

This research aimed to answer two primary research questions regarding the use of decentralized PPO policies for multi-drone collision avoidance under partial observability, and the impact of enhanced sensor perception - LiDAR on policy effectiveness. The experiments conducted in this study show that decentralized PPO can, indeed, learn collision-avoidance behaviors in multi-drone settings, but its effectiveness rapidly drops in high-density environments, which illustrate that PPO is not recommended to be used in massive drone swarms.

Effectiveness of decentralized PPO under partial observability. In order to answer the posed **RQ1**, the experiments included training various PPO policies using a variety of hyper-parameters and reward function coefficients. The results demonstrate that PPO can learn decentralized collision-avoidance behaviors, achieving high goal-completion and collision-free rates in simple environments, for instance 100% completion and 94.2% collision-free with 2 drones and 0 obstacles. However, performance reduces as complexity increases. Across all team sizes, adding obstacles steadily reduced both completion and safety, Table 6. Moreover, training steps required grew when simulating scenarios with high number of drones, reflecting PPO's on-policy sample inefficiency in high-dimensional joint state–action spaces.

Impact of LiDAR-style observations. Series of testing trainings were designated to investigate **RQ2**. Namely, multiple training configurations were run to measure the efficiency of LiDAR. Moreover, the research included measuring different reward setups. Introducing LiDAR-style perception markedly improved learning stability and final performance. Policies trained with both state vectors and LiDAR converged faster and provided better results even in mid-density scenarios and resulted higher collision-free and completion rates compared to proximity-only baselines. This confirms that richer data perception mechanisms in partial-observability challenges enable safer obstacle avoidance and agent spacing.

Outcomes. While PPO provides a solid baseline for decentralized drone collision avoidance, its sample inefficiency and unified rewarding process limit scalability. Incorporating richer sensor data LiDAR improves the results, but further gains likely require different algorithms, for instance off-policy or factorized multi-agent algorithms. Future work should explore these alternatives and extend evaluation to 3D environments to better approximate real-world UAV operations.

USE OF AI

During the research `ChatGPT 4o` and `Grammarly` were used to validate grammar, typographical accuracy, and preserve the required academic style. `ChatGPT 4o` was also used for additional scrapping in the `Gymnasium, PettingZoo` and `SuperSuit` documentation. The author takes full responsibility for the content of this work. Lastly, `ChatGPT 4o` was used to improve efficiency and debug the rendering of the simulator.

REFERENCES

[1] José Amendola, Linga Reddy Cenkeramaddi, and Ajit Jha. Drone landing and reinforcement learning: State-of-art, challenges and opportunities. *IEEE Open Journal of Intelligent Transportation Systems*, 5:520–539, 2024.

[2] Gopi Gugan and Anwar Haque. Path planning for autonomous drones: Challenges and future directions. *Drones*, 7(3), 2023.

[3] Yu-Hsin Hsu and Rung-Hung Gau. Reinforcement learning-based collision avoidance and optimal trajectory planning in uav communication networks. *IEEE Transactions on Mobile Computing*, 21(1):306–320, 2022.

[4] Amudhini P. Kalidas, Christy Jackson Joshua, Abdul Quadir Md, Shakila Basheer, Senthilkumar Mohan, and Sapiah Sakri. Deep reinforcement learning for vision-based navigation of uavs in avoiding stationary and mobile obstacles. *Drones*, 7(4), 2023.

[5] Kjell Kersandt, Guillem Muñoz, and Cristina Barrado. Self-training by reinforcement learning for full-autonomous drones of the future. In *2018 IEEE/AIAA 37th Digital Avionics Systems Conference (DASC)*, pages 1–10, 2018.

[6] Rajalakshmi Krishnamurthi, Anand Nayyar, and Aboul Ella Hassanien, editors. *Development and Future of Internet of Drones (IoD): Insights, Trends and Road Ahead*, volume 332 of *Studies in Systems, Decision and Control*. Springer, Cham, 2021.

[7] Pinxin Long, Tingxiang Fan, Xinyi Liao, Wenxi Liu, Hao Zhang, and Jia Pan. Towards optimally decentralized multi-robot collision avoidance via deep reinforcement learning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6252–6259, 2018.

[8] Abu Jafar Md Muzahid, Syafiq Fauzi Kamarulzaman, Md. Arafatur Rahman, and Ali H. Alenezi. Deep reinforcement learning-based driving strategy for avoidance of chain collisions and its safety efficiency analysis in autonomous vehicles. *IEEE Access*, 10:43303–43319, 2022.

[9] Sihem Ouahouah, Miloud Bagaa, Jonathan Prados-Garzon, and Tarik Taleb. Deep-reinforcement-learning-based collision avoidance in uav environment. *IEEE Internet of Things Journal*, 9(6):4015–4030, 2022.

[10] Mohammad Reza Rezaee, Nor Asilah Wati Abdul Hamid, Masnida Hussin, and Zuriati Ahmad Zukarnain. Comprehensive review of drones collision avoidance schemes: Challenges and open issues. *IEEE Transactions on Intelligent Transportation Systems*, 25(7):6397–6426, 2024.

[11] Sang-Yun Shin, Yong-Won Kang, and Yong-Guk Kim. Obstacle avoidance drone by deep reinforcement learning and its racing with human pilot. *Applied Sciences*, 9(24), 2019.

[12] Dawei Wang, Tingxiang Fan, Tao Han, and Jia Pan. A two-stage reinforcement learning approach for multi-uav collision avoidance under imperfect sensing. *IEEE Robotics and Automation Letters*, 5(2):3098–3105, 2020.

[13] Jawad N. Yasin, Sherif A. S. Mohamed, Mohammad-Hashem Haghbayan, Jukka Heikkonen, Hannu Tenhunen, and Juha Plosila. Unmanned aerial vehicles (uavs): Collision avoidance systems and approaches. *IEEE Access*, 8:105139–105155, 2020.

[14] Ender Çetin, Cristina Barrado, Guillem Muñoz, Miquel Macias, and Enric Pastor. Drone navigation and avoidance of obstacles through deep reinforcement learning. In *2019 IEEE/AIAA 38th Digital Avionics Systems Conference (DASC)*, pages 1–7, 2019.

Appendix A

Supersuit Wrappers.

1. Dead-Agent Handling with `black_death_v3`.
Since agents may reach their goals or collide before the episode ends, `black death v3` ensures that their observation and action spaces remain valid by freezing them and returning zeroed observations. terminates early.

```
black_death_v3
```

This wrapper allows to have a stable observation input for training model even with variable agents' lifespans. Without this wrapper, vectorized training would crash when an agent.

2. `PettingZoo` *to* `VecEnv` *Conversion.*
The base multi-agent environment, wrapped with `PettingZoo's ParallelEnv` interface, is converted to a single agent-compatible `VecEnv` using: wrapped with `PettingZoo's ParallelEnv` interface, is converted to a single agent-compatible `VecEnv` using:

```
pettingzoo_env_to_vec_env_v1()
```

This step formats the per-agent structure into a standard vectorized format, allowing interoperability with Stable-Baselines3 policies and replay buffers.

3. Vectorization and Parallelism.
In order to improve the training speed and provide episode diversity, a single environment is duplicated N times(in our case 16) via the function:

```
concat_vec_envs_v1()
```

This creates a batch of independent simulations running in parallel, providing 16 experiences per step to the PPO agent. This wrapper dramatically improves training speed and policy generalization by creating a more diverse situation for the agents to experience and allowing agents to try a variety of different tactics per each policy update.

4. Episode Logging and Statistics.
The wrapped environment is wrapped in `VecMonitor` wrapper to enable per episode metric logging, including reward and episode length tracking. This data allows further improvement in hyperparameters tunning and reward system adjustment.

5. Normalization with `VecNormalize`.
Finally, the vectorized environment is wrapped with `VecNormalize` to normalize observations(and optionally rewards). Even though the observations are manually normalized by variables division, for example, distance is divided by arena size which results in [~0.0-1.0] normalization, wrapper provides proper format and ensures normalization. Normalization helps stabilize learning by reducing sensitivity to input scale and variance.

Depending on the experiment reward normalization was tuned or even turned off. When rewards are left unnormalized, the training system fully relies on the configured reward shaping.

PPO HyperParameters. This section outlines the reasoning beyond key hyperparameters used in the final model and explains the rationale behind their values.

Discount Factor - `Gamma = 0.99`.
Throughout the experiment, it was discovered that a high discount fact of $\gamma = 0.99$ suits the best as it promotes the path-planning. Since reaching the goal can take 230–400 timesteps, lower discount factors (e.g. $\gamma = 0.95$) caused agents to prefer short-term motion over strategic progress, rs to ineffective trajectories and orbiting near obstacles.

Neural Network Architecture.
The policy and value networks share the same multilayer perceptron architecture:

```
[256, 128, 64]
```

This architecture provided a good balance between representational capacity and training stability. Shallower networks (e.g., `[128, 64]`) underfit complex observation spaces, especially with LiDAR, while deeper versions such as `[512, 256, 128]` were used in training phases with higher number of agents involved.

Learning Rate.
A constant learning rate of `3e-4` was chosen after comparing with

scheduled decays and smaller values. It allowed reasonably fast learning while avoiding gradient explosions. Learning rate scheduling (`get_linear_fn`) slightly improved the final results. However, the scheduled learning rate sometimes shortened the critical exploration while training by curriculum stages.

Batch Size and Rollout Steps.
Training was performed with `n_steps = 512` and `batch_size = 2048` (assuming `N_ENVS = 16`). This batch size showed sufficient data diversity across parallel environments while staying computationally feasible. Specifically, each PPO training epoch is divided into four mini-batches with a rollout size of `n_steps = 512` and a batch size of `2048`. Foremost iterations of hyperparameters involved link with `n_steps = 300` and `batchSize = 480 or 120`. However, because some complex episodes with many agents required more than 300 steps to finish successfully and this setting does not provide promising results, it was decided to switch to `512, 2048` values.

GAE Lambda `= 0.95`.
The Generalized Advantage Estimation (GAE) parameter $\lambda = 0.95$ provided a good trade-off between bias and variance. This is a commonly used default in continuous control tasks and worked reliably in early curriculum phases.

Clipping Range `clip_range = 0.25`.
The clipping parameter was increased from the standard `0.2` to `0.25-0.34` to allow slightly more aggressive policy updates. This clipping range allowed model to make improvements faster without causing the instability due to the stabilizing effect of reward shaping and observation normalization.

Entropy Coefficient `ent_coef = 0.008`. Entropy regularization was applied to encourage exploration, particularly in obstacle-dense stages. A coefficient of `0.008` was empirically tuned to balance exploration without overwhelming the shaped reward signal. Strongly higher values led to unpredictable behaviour, while smaller values represented slow exploration.

Best Configuration. The code snippet 3 presents the final PPO model configuration, resulting in the best training performance and generalization across most curriculum stages.

PettingZoo Adapter Methods: **`reset(seed)`:** Resets the inner world and returns a dictionary of observations indexed by agent names instead of the raw observations per drone.

`step(action_dict)`: The following method inputs dictionary of per agent actions to be taken at this step. Simultaneously, this `DronePZ's` method takes action for all this drones by converting the input dictionary to raw `Numpy` array. In comparison to the DroneWorldEnv's `step()`, the `DronePZ PettingZoo` adapter returns per agent dictionaries of all observations, rewards, done's flags, truncation flags, and info for all currently active agents. Finished agents are dynamically removed from the environment.

`render()`: Delegates visualization to the internal `DroneWorldEnv`.

`close()`: Cleans up Pygame resources by calling the world's close method.

Reward Function

(1) *Terminal Cases.*
- **Success:** + REACH_BONUS
- **Collision or wall-hit:** − COLLISION_PEN
- **Truncation:** − TRUNC_PEN

(2) *Progress and Costs.*

$$r_{\text{prog}} = \text{PROG\_COEF} \times \Delta d \; - \; \text{STEP\_COST} \; - \; \text{ACTION\_PEN} \times \|\mathbf{a}\|$$

Where Δd is difference between distance to goal in previous step and in current step. Action penalty is applied to action $\|\mathbf{a}\|$ taken in this step.

(3) *Alignment.* Both action $\mathbf{a}$ and velocity $\mathbf{v}$ are projected onto the goal direction unit vector $\hat{\mathbf{g}}$:

$$r_{\text{align}} = \text{ALIGN\_COEF} \times \left(\hat{\mathbf{a}}\cdot\hat{\mathbf{g}} + \hat{\mathbf{v}}\cdot\hat{\mathbf{g}}\right).$$

(4) *Final-Mile & Braking.* Activated when $dist\_goal$ < FINAL_M_R:
- *Well pull:* $1 - (d/\text{FINAL\_M\_R})^2$
- *Braking bonus:* BRAKE_COEF $\max\{0, -\mathbf{a}\hat{\mathbf{g}}\}$ if d < BRAKE_R
- *Stall nudge / overspeed penalty:* small penalties or bonuses when speed falls below or exceeds thresholds

(5) *Tangential Penalty.* Discourages movement orthogonal to the goal:

$$r_{\tan} = -\,0.5\,\|\mathbf{v} - (\mathbf{v}\cdot\hat{\mathbf{g}})\,\hat{\mathbf{g}}\|.$$

This penalty was implemented because of the not optimal path curves and agent's orbital braking around the goal.

(6) *Repulsion & Collision Avoidance.* For the nearest neighbor distance m and obstacle distance o:
- *Static repulsion:* $-$ REP_COEF $(1 - m/\text{FINAL\_M\_R})^{1.2} \times$ $\max\{0,\ \text{SAFE\_DIST} - m\}/\text{SAFE\_DIST}$
- *Danger penalty:* similarly scaled by COLLISION_PEN inside DANGEROUS_DIST.
- *Active dodge:* bonus when velocity has negative radial component toward the nearest drone or obstacle, scaled by AWAY_DRONE_COEF or AWAY_OBS_COEF.

Appendix B

World	Continuous 2-D square arena, 100 m × 100 m. Positioning expressed in metres, time discretised at 10 Hz
Agents	Radius = 0.5 m; Max speed = 5 m s^{-1}; Max acceleration = 2 m s^{-2}. Dynamics integrated with a simple Euler step.
Static Obstacles	Tunable number of obstacles. Spawn at random location. Circular obstacles with random radius 1.0 – 3.5 m
Goal	Each agent has a unique goal location. Mirrored or random spawn point that is greater than 15 m away.
Observation Space (16-40 floats)	Own velocity – 2 floats Vector to goal – 2 floats Goal direction – 1 float Vector to closest wall – 2 floats Vector to nearest obstacle – 2 floats Distance to nearest obstacle – 1 float Vector to nearest drone – 2 floats Velocity of nearest drone – 2 floats Distance to nearest drone – 1 float Heading angle to goal – 1 float LiDAR observation: 24 rays of 20 m – 24 floats
Action Space	Two-dimensional acceleration vector applied to current velocity. Clipped to maximum acceleration.
Frameworks	Gymnasium for simulation structure. PettingZoo for multi-agent tagging. Stable-Baselines3 for PPO training. Supersuit for wrapping and vectorization. Pygame for visualization.

Table 7. Environment parameters and structure used in simulation.

Phase	Stage	Drones	Obstacles	Timesteps	Spawn Area (m)
1	1	1	0	500k	40.0
	2	1	3	750k	40.0
	3	1	6	1M	45.0
	4	1	10	1.25M	60.0
2	1	2	0	1M	40.0
	2	2	4	1.25M	40.0
	3	2	8	1.25M	50.0
	4	2	10	1.75M	70.0
3	1	4	0	1.25M	50.0
	2	4	4	1.75M	60.0
	3	4	8	2.25M	70.0
	4	4	10	2.5M	80.0
	5	4	15	2.75M	100.0
4	1	6	0	2.2M	70.0
	2	6	4	2.6M	80.0
	3	6	8	3M	100.0
	4	6	10	3.5M	100.0
	5	6	15	4M	100.0

Table 8. Curriculum training stages grouped into testing phases. Each phase is dedicated to n-number of drones.