International Journal of Safety and Security Engineering
Vol. 16, No. 6, June, 2026, pp. 1183-1196
Journal homepage: http://iieta.org/journals/ijsse

Decentralized and Fault-Tolerant UAV Swarm Navigation via Single-Agent Deep Reinforcement Learning for Civil Protection Applications

Volodymyr Iatsyshyn[1], Solomiia Liaskovska[1,2], Sviatoslav Shainoha[1], Oleh Berezsky[3], Bohdan Sydor[1], Jamil Abedalrahim Jamil Alsayaydeh[4*], Mohd Faizal Yusof[5], Yevgen Martyn[6]

[1] Department of Artificial Intelligence, Lviv Polytechnic National University, Lviv 79005, Ukraine
[2] Department of Mechanical Engineering, Faculty of Engineering, Computing and the Environment, Kingston University, London KT12EE, United Kingdom
[3] Department of Computer Engineering, West Ukrainian National University, Ternopil 46003, Ukraine
[4] Department of Engineering Technology, Fakulti Teknologi Dan Kejuruteraan Elektronik Dan Komputer (FTKEK), Universiti Teknikal Malaysia Melaka (UTeM), Melaka 76100, Malaysia
[5] Department-Research Section, Faculty of Resilience, Rabdan Academy, Abu Dhabi 22401, United Arab Emirates
[6] Department of Project Management, Information Technologies and Telecommunication, Lviv State University of Life Safety, Lviv 79007, Ukraine

Corresponding Author Email: jamil@utem.edu.my

https://doi.org/10.18280/ijsse.160602

Received: 15 January 2026
Revised: 29 March 2026
Accepted: 9 April 2026
Available online: 30 June 2026

Keywords:
machine learning, deep reinforcement learning, simulation environment, autonomous systems, multi-agent systems, AirSim, Gymnasium, Deep Q-Network

ABSTRACT

Path planning and collision avoidance for Unmanned Aerial Vehicle (UAV) swarms remain challenging in dynamic, unknown environments typical of civil emergencies and disaster response. This study investigates the use of Deep Reinforcement Learning (DRL) for decentralized swarm navigation, where coordinated behavior emerges from individually trained agents sharing only limited local state information. To evaluate this hypothesis, custom Gymnasium environments were developed, progressing from simplified 2D navigation to a 3D environment with dynamic obstacles. Four DRL algorithms—Deep Q-Network (DQN), Advantage Actor-Critic (A2C), Proximal Policy Optimization (PPO), and Soft Actor-Critic (SAC)—were implemented using Stable-Baselines3 and trained on single agents. The trained models were evaluated in a high-fidelity multi-agent AirSim simulation, where a three-UAV swarm navigated to prioritized targets while avoiding collisions. The system also supported fault tolerance through dynamic goal reassignment following agent failure. Experimental results revealed substantial differences in algorithm performance. DQN and A2C showed limited generalization, whereas PPO and SAC demonstrated robust decentralized coordination. PPO achieved a 95% success rate for the priority target, while SAC achieved 98% and successfully guided all three UAVs to their assigned targets in 52% of test episodes. The results demonstrate that actor-critic methods, particularly SAC, enable scalable and reliable decentralized swarm behavior, making them well suited for UAV coordination in civil safety and disaster response applications.

1. INTRODUCTION

Autonomous Unmanned Aerial Vehicles (UAVs) are becoming increasingly widespread in civil society. To expand their capabilities and make them effective tools for safeguarding human life, efficient control systems are needed [1]. One of the least researched topics is UAV swarms, in which multiple devices exchange information and perform tasks together [2]. The use of swarms requires an effective method of navigation and communication within the swarm. To solve this problem, machine learning methods are widely used, which adapt to dynamic environments and make decisions in real time.

This paper describes a proposed approach for planning the flight path of a UAV swarm designed for civil use. This approach considers environmental constraints, obstacles (such as collapsed buildings or smoke plumes), and the dynamics of other UAVs.

This publication represents a specific milestone within a broader, ongoing research project aimed at building a complete, autonomous control system for UAVs. A defining constraint of this overarching project is the reliance on low-compute hardware, specifically Single Board Computers (SBCs) typically found on lightweight civil drones. Consequently, the algorithms selected and developed must balance performance with computational efficiency suitable for edge deployment.

A core hypothesis of this research is that complex swarm coordination – specifically collision avoidance and dynamic task reallocation – can emerge from the interaction of autonomous agents trained individually. By training a single agent to navigate dynamic environments and subsequently

duplicating this policy across a swarm that shares local state information (position and velocity), we aim to suggest that explicit multi-agent training is not strictly necessary for effective civil protection teams. This approach significantly reduces training complexity while increasing scalability.

To validate existing and proposed approaches, an environment was developed to simulate the behavior of a swarm of UAVs in three-dimensional space, accounting for realistic flight physics and obstacles.

The contributions of this work can be summarized as:

- Investigation of a decentralized training paradigm in which swarm coordination emerges from independently trained agents, eliminating the need for explicit multi-agent reinforcement learning.
- Development of a decentralized, fault-tolerant UAV swarm control system capable of autonomous navigation and dynamic goal reassignment in the event of agent loss, ensuring mission continuity.
- Design and implementation of a progressive multi-stage training pipeline using custom Gymnasium environments, gradually increasing task complexity from simple 2D navigation to realistic 3D multi-agent coordination.
- Integration of Deep Reinforcement Learning (DRL) algorithms (Deep Q-Network (DQN), Advantage Actor-Critic (A2C), Proximal Policy Optimization (PPO), and Soft Actor-Critic (SAC)) into UAV swarm control, with systematic benchmarking of their performance in complex dynamic environments.
- Creation of a realistic simulation testbed based on AirSim and Stable-Baselines3 for validating swarm behavior under realistic physical constraints and communication delays.

It is important to emphasize that this work does not propose a novel reinforcement learning algorithm. Instead, the contribution lies in a simplified training paradigm for decentralized UAV swarm coordination, where policies are trained in a single-agent environment and subsequently deployed in a multi-agent setting. The objective is to evaluate whether complex swarm behaviors can emerge without explicit multi-agent reinforcement learning, thereby reducing training complexity and enabling deployment on resource-constrained platforms.

2. MATERIALS AND METHODS

An UAV is an aircraft capable of flying without a pilot on board and can operate autonomously or receive commands from a remote operator. While initially developed for military purposes, UAVs are now pivotal in civilian applications. These include environmental monitoring, agricultural assessment, and most critically, civil protection tasks such as:

- Natural Disaster Response: Rapid mapping of flooded areas, assessing structural damage after earthquakes, and monitoring forest fires to guide ground crews.
- Technological Disaster Management: Inspecting hazardous zones in nuclear or chemical accidents where human presence is too risky, and monitoring gas leaks or structural integrity of industrial sites.
- Search and Rescue (SAR): Locating missing persons in remote or difficult terrain during emergencies.

UAVs can be categorized by weight, design, control method, altitude, and flight duration [3]. By control type, UAVs are divided into uncontrolled, autonomous, and remotely piloted. By weight, they are divided into four categories: micro (up to 2 kilograms), mini (from 2 kilograms to 20 kilograms), small (from 20 kilograms to 150 kilograms), and large (from 150 kilograms). By design, UAVs are divided into four types: fixed-wing, multicopters, single-rotor helicopters, and hybrid Vertical Take-Off & Landing (VTOL) with fixed wings. Recent research has demonstrated the efficiency of decentralized and asynchronous coordination strategies for UAV swarms. In particular, the study [4] proposed an asynchronous method for parallel processing of navigation data streams at the agent level, enabling real-time decision-making without relying on a centralized control unit. This approach minimizes coordination latency, eliminates computational bottlenecks, and improves the scalability of UAV swarm systems. The study [5] used the PPO algorithm. This is an on-policy algorithm, i.e., it updates the strategy based on data collected in the previous step. Like A2C, it is based on actor-critic architecture and uses an identical preference function. In addition to the components described in A2C, the PPO algorithm includes clipping and the number of policy epochs. The advantages of PPO include stable policy development, thanks to the clipping function, which reduces policy fluctuations, and fast convergence, thanks to more controlled updates.

Multicopters were used for this study. A multicopter is a type of UAV that uses several horizontally arranged propellers that rotate diagonally in opposite directions. They are used to control the speed, direction, and altitude of flight. Multicopters are classified according to the number of propellers, namely bicopters (2 propellers), tricopters (3 propellers), quadcopters (4 propellers), hexacopters (6 propellers), and octocopters (8 propellers). Quadcopters are the most common because they provide the best balance between lift, controllability, maneuverability, and cost. The main advantages of multicopters are their ability to take off and land vertically. They can also hover in the air and move in any direction at the same speed, and they are highly maneuverable. They are easy to use, more economical, and cheaper than fixed-wing aircraft, and can carry significantly heavier loads. Their ability to fly at low altitudes also enables them to collect data with high spatial resolution, making them particularly suitable tools for mapping [6].

The disadvantages of multicopters are their shorter flight range and speed compared to fixed-wing UAVs. They also require a lot of energy to maintain altitude. On average, they are capable of maintaining flight for 30 minutes. If the UAV's payload capacity is greater, this time is proportionally shorter. Multicopters require precise control over propeller speed, so these UAVs typically operate on electric motors. The use of gasoline engines under such conditions is practically impossible. Multicopters also require complex and regular maintenance due to their complex design.

2.1 Swarm of UAV in civil protection

A swarm of UAVs is a group of autonomous drones that work together to accomplish tasks. The way a swarm is organized is similar to that of its biological counterparts, such as birds or ants, where individuals can coordinate and interact with each other and their environment to achieve a common goal [7]. Each UAV operates autonomously, exchanging information with others and adapting to changes in the environment. Figure 1 demonstrates an example of swarm of UAVs.

The main idea behind swarm is the absence of a single

decision-making system. Instead, since each device acts independently, decentralized self-organization of the swarm is ensured. For example, some systems employ a leadership approach, where one UAV acts as the leader, and the rest of the UAVs follow its trajectory [8]. However, most approaches reject leadership and are based on local interaction, where each UAV chooses its actions based on the behavior of its neighbors.

UAV swarms have several key advantages over single-unit applications. The first advantage is that using a group of specialized devices is more cost-effective than using a single, complex, multi-purpose device. Additionally, this system is scalable, as devices can be added to the swarm without compromising coordination, allowing for larger areas to be covered during missions and enabling the load to be distributed [8]. Furthermore, the swarm increases the system's fault tolerance, as if one UAV fails, the others can compensate for it. Additionally, the parallel execution of smaller subtasks by each device significantly speeds up work on complex missions.

Figure 1. Swarm of UAVs

2.1.1 Approaches to route planning for UAV swarms

For route planning and collision avoidance for a swarm of UAVs, the most fundamental distinction is between centralized and decentralized methods.

Centralized algorithms represent the swarm as a single multidimensional system with a combined degree of freedom (DOF) of all agents and apply a standard planning algorithm for the entire swarm. The advantage of this approach is guaranteed completeness of the solution, i.e., if a solution exists, it will be found. The main disadvantage of such algorithms is that as the number of devices increases, the complexity of finding a solution increases linearly [9].

Decentralized algorithms are based on the fact that each UAV plans its path individually. The study [9] provides an example of an algorithm in which each UAV first finds a path without obstacles, after which speeds are adjusted to avoid collisions with other devices. The decentralized approach is scalable and less computationally demanding. However, in such an algorithm, finding a solution is not guaranteed, so additional information exchange between vehicles is necessary for success. This approach is more reliable because it eliminates a single point of failure, unlike the centralized approach. Considering all the advantages of this approach, it will be used to solve the task at hand.

2.1.2 Autopiloting with PX4

PX4 Autopilot is open-source software for automated UAV piloting, widely used to control various autonomous devices [10]. PX4 supports a wide range of platforms, including multirotors, tiltrotors, fixed-wing aircraft, and ground and marine robotic systems. It provides a reliable environment for flight control, ensuring stabilization, navigation, and mission support in challenging conditions.

The autopilot is based on a core that processes information from sensors located on the device, including GPS, inertial measurement units (IMU), barometers, lidars, and cameras. The collected data is used to represent the environment and the current state of the UAV and is further used to perform controlled and autonomous flights. PX4 also supports advanced trajectory planning capabilities and stable flight even in the event of signal loss and difficult weather conditions.

PX4 also supports the MAVLink protocol, a lightweight protocol for exchanging messages between a flying device and a ground station or other UAVs. MAVLink allows you to exchange telemetry, send commands, and receive data about the status of the UAV in real time.

2.2 The task of route planning and collision avoidance

Path planning is a key element in ensuring the autonomy of UAVs in performing various tasks, as it involves finding the optimal path from the starting point to the specified destination [11]. It enables UAVs to perform missions efficiently, reducing energy and time costs and minimizing risks. However, in addition to determining the best trajectory, it is necessary to solve the problem of avoiding and circumventing obstacles that may appear on the path during maneuvers, since the main task is to reach the target without colliding with the environment [12]. In a civil protection context, "obstacles" may include unpredictable elements like falling structures, smoke, or other emergency vehicles.

2.2.1 Classification of path planning algorithms

Path planning algorithms are commonly divided into two broad categories: traditional algorithms and intelligent algorithms [13]. Traditional methods usually require a prior map of the target environment before trajectory planning begins. Based on this map, a graph representation is first constructed, and then search algorithms are applied to identify feasible trajectories for the UAV group. Common traditional path planning algorithms include Dijkstra's algorithm, A*, Rapidly-Exploring Random Tree (RRT), and Fast Marching Square, all of which have been widely used in UAV path planning tasks [14-16].

However, in disasters, pre-existing maps are often obsolete due to destruction. Intelligent algorithms, in turn, mimic the behavior of biological organisms to search for the best solution jointly. Such algorithms are capable of adapting to unknown and dynamic environmental conditions by generalizing experience after training. Given the numerous advantages of using intelligent algorithms – namely, high adaptability to unpredictable changes, the ability to work in unknown environments, and noise resistance – this study will focus specifically on intelligent algorithms for performing civil protection tasks.

2.2.2 Intelligent algorithms: Reinforcement learning

For planning UAV routes, most studies used machine learning, specifically various reinforcement learning algorithms. These algorithms aim to train an agent to make optimal decisions in a learning environment. The agent gradually gains experience and receives rewards for its

actions, based on which a strategy is developed to maximize the total reward in the future [17, 18].

Some studies use the Deep Q-Learning algorithm [19, 20] to solve the path planning problem. This algorithm combines the Q-learning algorithm with neural networks. The traditional Q-learning algorithm is based on a Q-table, where for each state of the environment, a set of actions is available, and the action with the highest Q-value is selected. The primary issue with traditional Q-learning is the increase in memory required as the number of states and actions increases. Deep Q-learning solves this problem by using a neural network to approximate the Q-function. However, DQN also has certain limitations. First, the algorithm can only work with a discrete action space. Second, the computational cost increases rapidly for complex tasks. The algorithm also demonstrates slow convergence for large state spaces.

The study [21, 22] compared deep Q-learning with another algorithm, actor-critic with a preference function, for the task of performing an autonomous mission. This is an on-policy learning algorithm, i.e., it updates the strategy based on data collected in the previous step. The main advantages of actor-critic are reduced variance when updating the policy and, unlike deep Q-learning, the ability to choose actions in both discrete and continuous spaces. One of the disadvantages is that, like any on-policy algorithm, A2C uses data inefficiently, as old data becomes irrelevant when policy parameters change.

In the study [2], a SAC algorithm was used to train Autonomous UAV in a two-dimensional environment with obstacles. Like the previous two algorithms, it is based on the actor-critic architecture. However, SAC works with a stochastic policy, preserves action diversity, and promotes better exploration of the environment. Unlike previous algorithms, SAC uses entropy regularization for the objective function [23]. The algorithm utilizes two Q-functions to mitigate the variability of estimates, similar to the double Q-learning algorithm. Its target value network is updated using exponential smoothing.

Additionally, unlike A2C and PPO, SAC is an off-policy algorithm that utilizes an experience buffer. Another feature of this algorithm is that it can only work with a continuous action space. However, SAC has greater computational complexity compared to the previously described algorithms.

2.3 Simulation environment

AirSim is an open simulation environment developed by Microsoft that supports various types of both UAVs and ground vehicles. The main advantages of this environment include realistic visualization using the Unreal Engine 5 engine, support for various types of sensors, such as cameras, lidars, GPS, and others, ongoing support, and regular updates from the community. There is also an API for development using Python, which simplifies the development and testing of various scenarios. The disadvantages include high resource requirements due to the realism of the graphics and the complexity of the settings, which is a result of their large number.

2.4 Problem statement

Within the framework of this study, we simulate a civil protection mission where a swarm of UAVs must navigate a hazardous area. We assume that each UAV in the swarm has local information about its environment, its position [24] in the swarm, and distances to obstacles. Each UAV acts individually based on local observations, and the control system is decentralized [25, 26].

In this research, we specifically focus on model-based path planning and collision avoidance, assuming that the larger control architecture is supported by fully functional, modular subsystems. Given the project's focus on low-compute hardware (SBC), the path planning module relies on the inputs from pre-existing, efficient subsystems, including:

- Friendly UAV Relative Position Estimation: Reliable localization of swarm members relative to the ego-agent.
- Visual Odometry: Accurate self-localization and ego-motion estimation without heavy reliance on external GPS.
- Fast Image Depth Estimation: Real-time obstacle detection and ranging derived from frontal camera feeds.

By abstracting these perception and localization tasks, this study isolates the decision-making component, evaluating the capacity of learning-based models to navigate complex environments using these high-level inputs.

We hypothesize that by training agents individually to respect dynamic constraints and avoid moving obstacles, the collective application of this policy in a communicating group will result in emergent swarm intelligence. This emergent behavior enables coordinated navigation, collision avoidance, and dynamic task distribution. Thus, the system is scalable, as increasing the number of devices does not require changes to control algorithms, and is capable of operating even in the event of the loss of one or more UAVs – a common occurrence in hostile disaster environments.

Let us assume that the UAV swarm consists of three agents (see Figure 2), each with its own individually assigned target (e.g., a potential survivor location).

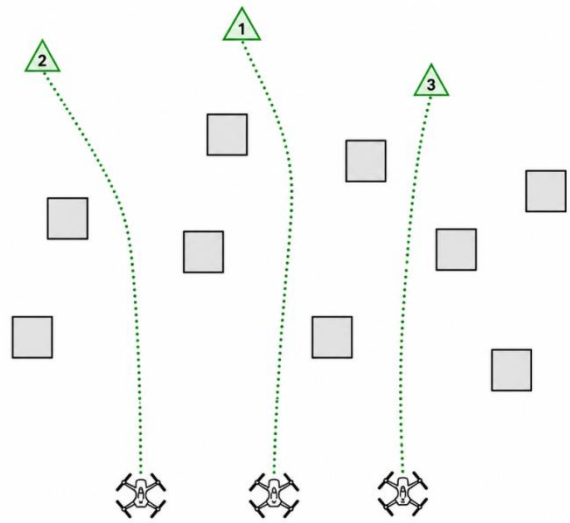

Figure 2. Visualization of possible trajectories for an UAV swarm of 3 drones

The targets are ranked by priority. The target for the first UAV (e.g., a critical medical supply drop) has a higher priority than the target for the second. If one of the UAVs fails (e.g., damaged by debris), the agent with the lowest priority changes its target to the higher priority one. Priority reassignment is modeled as a discrete event-triggered update occurring only upon agent failure, preventing oscillatory behavior in task allocation. For example, if the first agent encounters an obstacle and fails, the third agent changes its target to the more important one that belonged to the first device. At the same

time, the second UAV continues to move towards its original target (see Figure 3). This fault tolerance mechanism ensures that the most critical civil protection objectives are met even if the swarm sustains losses.

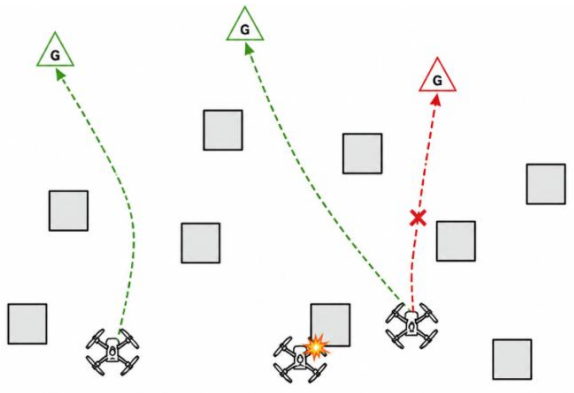

Figure 3. Visualization of possible trajectory changes for an UAV swarm of three drones in case of UAV failure

2.5 Training stages

To effectively address the complexity of the global task, a curriculum learning approach was adopted, conducting preliminary training in two simplified 2D environments to validate the algorithmic settings. The first stage focused on basic navigation with fixed discrete displacements and static obstacles, while the second introduced complex dynamics, including acceleration control, velocity constraints, and mobile targets, alongside a progressive milestone-based reward system. These foundational experiments confirmed the agent's ability to master navigation and velocity regulation in controlled settings, serving as a necessary prerequisite for the 3D environment.

2.5.1 3D environment with a single UAV

To train the agent in a three-dimensional environment, a single-agent setup is used with both static and dynamic obstacles. Static obstacles represent environmental constraints (e.g., buildings), while dynamic obstacles simulate moving entities such as other UAVs or falling debris. Dynamic obstacles follow bounded velocity profiles and are generated to create collision-prone interactions.

To ensure reproducibility, the environment is explicitly parameterized. The workspace is defined as a bounded horizontal plane (fixed altitude during training), with dimensions progressively increasing from approximately 40 × 20 m to 120 × 60 m as part of the curriculum learning schedule. Task complexity is scaled by increasing the number of static and dynamic obstacles at predefined training steps.

Static obstacles are uniformly distributed within map bounds, with a minimum inter-obstacle clearance of 5 m to maintain feasible navigation. Dynamic obstacles are sampled uniformly and assigned trajectories that intersect the agent's motion, with speeds bounded by the agent's kinematic limits.

Goal positions are sampled from rectangular regions that shift away from the origin across training stages, increasing trajectory length and planning horizon complexity. Within each stage, goals are uniformly distributed within the defined bounds.

Environment transitions occur at predefined training steps (e.g., 100k, 600k, 1.1M, 1.6M, and 2.3M), corresponding to increases in map size, obstacle count, and goal distance. These transitions are deterministic and independent of agent performance.

Training is performed on a single agent interacting with dynamic obstacles. This formulation avoids the computational complexity of multi-agent reinforcement learning while enabling the learned policy to generalize to multi-agent scenarios, where other agents are treated as interacting dynamic entities.

The agent is initialized at position (0, 0, 5) with velocity constrained to a maximum of 1 m/s along each axis. The discrete action space consists of 27 actions defined as combinations of acceleration increments along the x, y, and z axes. Horizontal accelerations are adjusted in steps of 0.1, while vertical adjustments use a smaller increment of 0.02. This forms a set of motion primitives covering principal and diagonal directions, including a zero-action. For algorithms requiring continuous control, actions are represented as bounded continuous vectors with the same magnitude limits.

Table 1 demonstrates the state space specification for this environment.

Table 1. State space specification

Component	Feature	Dimension	Normalized Range	Description
Target Info	Relative position (x, y, z)	3	[-1, 1]	Normalized direction vector from agent to target
	Distance to target	1	[0, 1]	Distance to target scaled by maximum sensing range
Ego State	Linear velocity (vx, vy, vz)	3	[-1, 1]	Velocity normalized by maximum allowed speed
Dynamic Obstacles	Neighbor directions	9	[-1, 1]	Normalized vectors to 3 nearest moving obstacles
	Neighbor distances	3	[0, 1]	Scalar distances to the 3 nearest moving obstacles(normalized)
Static Obstacles	Obstacles directions	6	[-1, 1]	Normalized vectors to the 3 nearest obstacles
	Obstacles distances	3	[0, 1]	Scalar distances to the 3 nearest obstacles(normalized)
Total	Observation vector	37	-	Stacked vector input to the policy

All observation features are normalized before being passed to the policy. Direction vectors are naturally bounded in [-1, 1]. Distances are clipped to a maximum sensing range and scaled to [0, 1]. Velocity components are normalized by the maximum allowed velocity, resulting in values in [-1, 1].

The reward function is designed to guide the agent toward the target while ensuring safety and flight stability. The total reward at time step t, denoted as r_t, is defined as a sum of four components:

$$r_t = r_{milestone} + r_{terminal} + r_{collision} + r_{boundary}$$

(1) Progressive Milestone Reward ($r_{milestone}$). To encourage sustained progress, a threshold-based reward system was implemented where the reward magnitude scales with the distance covered. When the agent crosses a threshold k ∈ {10, 20, …, 90, 95} towards the target, it receives a reward equal to that threshold value:

$$r_{milestone} = \begin{cases} +k \text{ if progress exceeds threshold } k \\ -k \text{ if progress retreats below threshold } k \\ 0 \text{ otherwise} \end{cases}$$

(2) Terminal Reward ($r_{terminal}$). Upon successfully reaching the target zone (dtarget < 5 m), the episode terminates, and the agent receives a final sparse reward. This reward is penalized by the time taken to encourage time-optimal trajectories:

$$r_{terminal} = 300 - \frac{n}{5}$$

where, *n* represents the number of time steps in the current episode.

(3) Collision Penalty ($r_{collision}$). Safety is enforced through penalties for proximity to static obstacles. A penalty of -2 is applied if the agent enters a safety buffer zone (<3 m from an obstacle), and a collision penalty of -10 is applied upon contact.

(4) Boundary Constraint ($r_{boundary}$): To ensure the agent operates within the valid workspace, a penalty is applied if the flight altitude z violates the defined vertical corridor z ∈ [2, 8] meters.

The episode terminates under the following conditions: the agent successfully reaches the target (entering a 5-meter radius), collides with an obstacle, or violates critical altitude constraints (dropping below 0.3 meters or exceeding 10 meters).

During training, randomized dynamic moving obstacles are spawned periodically. Most obstacles have trajectories that intersect the current projected UAV trajectory. Obstacles are moved for several iterations until they are far enough from the UAV, and then they are removed from the training environment. The total and simultaneous number of moving obstacles, as well as the period of their creation, increases with the level of map complexity.

Thus, the agent can learn to avoid moving obstacles that directly pose a collision threat by trying to “knock down” the agent. Since other UAVs do not aim to collide with each other, this approach not only accelerates learning through the use of single-agent learning, but also enables solving more complex problems, as the threat of collisions in this environment is constant. Another advantage is the ability to scale to a large number of agents without requiring retraining.

2.5.2 3D environment with UAV swarm

The final step is to add several agents for testing in a three-dimensional environment. The testing environment features three agents and enables multi-agent interaction. This stage does not involve further training. Instead, it serves as the validation ground for our hypothesis. We deploy the policy trained in the single-agent environment onto three distinct UAVs, each controlled by an independent instance of the model.

This setup mimics a coordinated rescue team entering a hazard zone. When the environment is initialized, the trained single-agent model is dynamically loaded for each UAV. At each step, each agent processes its own observation – replacing the "moving obstacles" from training with the real-time position/velocity data of the other swarm agents.

When the environment is initialized, the trained model is dynamically loaded. At each step, each of the three agents has its own observation, which corresponds to the state space during training of the software environment, replacing observations of moving obstacles with information about other agents. The model, in turn, based on observations, outputs the optimal action for a specific agent, after which the simulation environment interprets this action. The episode continues until each agent reaches the goal or collides with an obstacle or another agent.

Training and evaluation environments are strictly separated. Training is performed exclusively in custom Gymnasium environments with procedurally generated maps, while evaluation is conducted in AirSim using independently defined scenarios. No environment instances, obstacle configurations, or map generators are shared between training and testing.

2.6 Control system design

The evolution of control architectures for autonomous mobile robots, particularly UAVs, has gone through several key stages, leading to the formation of modern hybrid approaches.

Deliberative (planning) architectures – this approach, which historically emerged first, is based on the Sense-Plan-Act (SPA) paradigm. It is based on the assumption that intelligent behavior is the result of a rational planning process based on a complete and accurate symbolic model of the world. The system sequentially performs cycles: it collects sensory data to update the global map of the world, uses planning algorithms (e.g., A* or its variations) to find the optimal sequence of actions leading to the goal, and only then executes this sequence. The advantages of this approach include the ability to generate globally optimal or near-optimal plans for complex tasks, as well as high predictability of behavior. Among the disadvantages are high computational complexity, which leads to significant delays in response. It is also critically vulnerable to inaccuracies in the model of the world and dynamic changes in the environment (the so-called “frame problem”). For fast UAVs operating in unstructured space, this approach is too slow and unstable. In response to the shortcomings of deliberative systems, Rodney Brooks proposed a radically different approach – Subsumption Architecture. This paradigm rejects the idea of centralized planning and a symbolic model of the world. Instead, the system is built “bottom-up” from a set of parallel behavioral modules. Each module represents a close “perception-action” connection and is responsible for a specific level of competence (e.g., “avoid obstacles,” “move forward”). Higher levels can “absorb” (suppress) the outputs of lower levels, thus implementing more complex behavior. Among the advantages are an extremely fast response to environmental changes, low computational requirements, and high reliability and fault tolerance due to parallelism. Among the disadvantages are the inability to plan for the long term and engage in goal-oriented behavior, as well as the complexity of designing and configuring interactions between modules to achieve the desired emergent behavior. Note that the device cannot make plans that go beyond its immediate perception.

Recognition of the limitations of both extreme approaches

has led to the development of hybrid architectures that seek to combine their strengths. Such systems employ a hierarchical structure, where different levels are responsible for various aspects of control. This research is being conducted for application in a three-level planning and control system. Figure 4 demonstrates a 3-level control system diagram. First level is strategic high-level mission planning, second level is tactical mid-level for near-term planning and collision avoidance, third level reactive level for immediate control and position hold in case of system failures.

Within this hierarchical framework, the proposed DRL-based controller operates at the tactical layer, responsible for local path planning and collision avoidance. The agent is intentionally decoupled from low-level UAV dynamics and stabilization, which are handled by the reactive control layer. Instead, it receives high-level state observations and outputs motion commands constrained by kinematic limits. The primary objective of training is to learn robust navigation policies in dynamic environments, enabling safe and efficient trajectory generation while remaining agnostic to platform-specific dynamics.

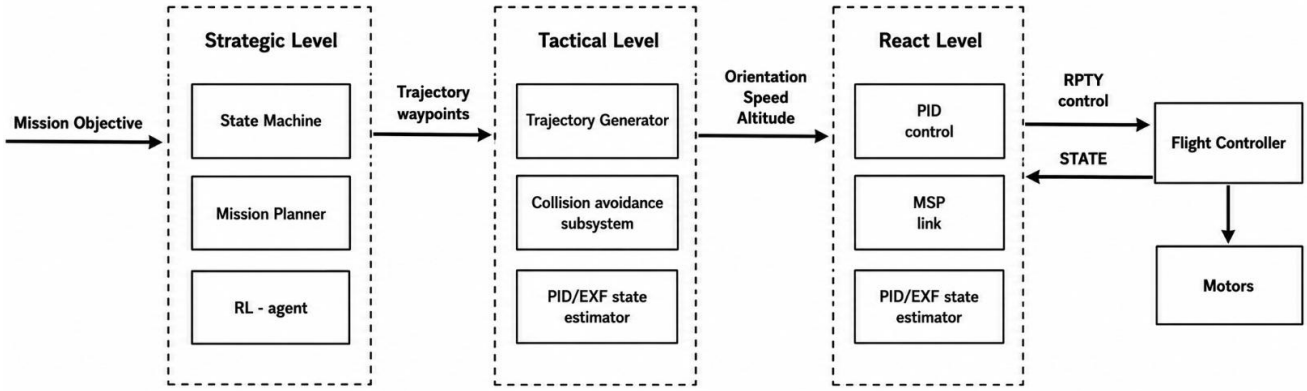

Figure 4. 3-level control system diagram

3. RESULTS

We have conducted a comparison of training efficiency and operational correctness of models trained in a custom Gymnasium environment with the following RL algorithms: DQN, A2C, PPO, and SAC. All evaluation results are obtained in previously unseen AirSim scenarios without further training or fine-tuning.

All experiments were conducted on a workstation with an Intel i9-10850K CPU and an NVIDIA RTX 3090 GPU (24 GB VRAM), using PyTorch-based Stable-Baselines3.

The policy and value networks are parameterized using a Multi-Layer Perceptron (MLP) architecture comprising two hidden layers, each containing 256 units with ReLU activation functions. During the hyperparameter optimization phase, a series of experiments were conducted evaluating various architectural configurations to determine the optimal model capacity. These tests included deeper and wider structures, such as a three-layer network with [512, 256, 128] units. However, empirical results indicated that increasing the complexity of the network did not yield significant improvements in convergence speed or final policy stability for this specific task. In fact, the larger architectures often incurred higher computational costs without a commensurate gain in accumulated reward. Consequently, the [256, 256] configuration was selected as the final architecture, as it demonstrated the most favorable balance between computational efficiency and learning performance. For Actor-Critic methods such as A2C and PPO, this configuration is applied to both the actor and the critic networks independently [20, 21].

3.1 Deep Q-Network training results

The first algorithm evaluated was DQN, which was trained using a batch size of 64 and an update frequency of 2.

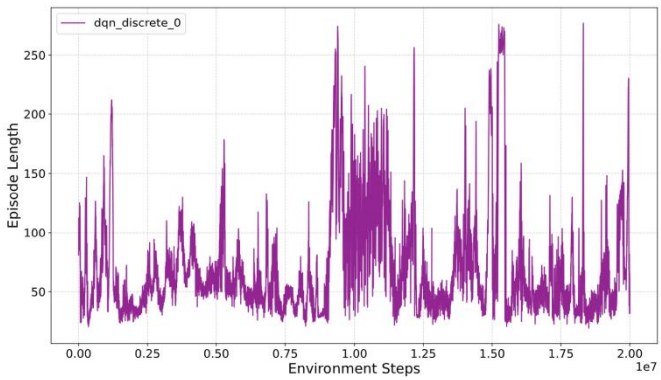

Figure 5. Average episodes length in final training of Deep Q-Network (DQN)

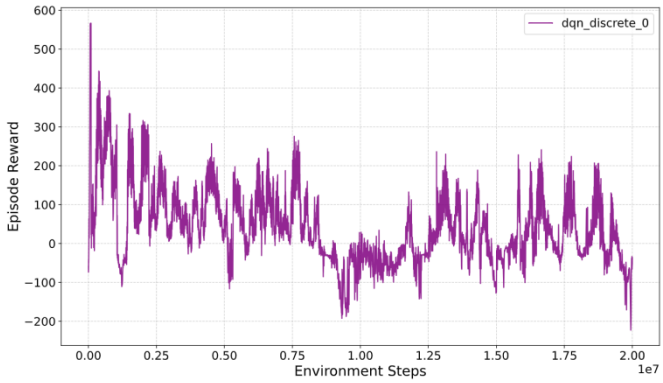

Figure 6. Average episodes reward in final training of Deep Q-Network (DQN)

Figure 5 shows average episode length in training of DQN.

As we can see, average episode length Min = 23.1 and Max = 262.1. Overall average episode length looks chaotic and hasn't improved over the course of the experiment.

Figure 6 shows average episode reward in final training of DQN and there is no visible improvement. Moreover, the mean reward gets worse in the course of experiment as task complexity increases over time.

The best indicator for the last map was an average reward of 256.6 with an average length of 46 steps. From all the experiments, it can be seen that learning is very unstable, as there are sharp jumps in reward and in the length of the learning episode that are not related to changes in map types.

3.2 Advantage Actor-Critic training results

Training using A2C was performed with the following parameters: a learning rate of 1e-3, a decay factor of 0.99, and a loss function coefficient for the critic of 0.5. The gradient was updated after every five steps. Figure 7 shows average episode length in final training of A2C. We can observe a slight overall increase of average episode length in the course of experiment, this can be explained with continued increase of task complexity as training progresses.

Figure 8 shows the average episode reward during A2C training. A rapid decrease in mean reward is observed in the first 3M steps due to increasing task complexity. Initially, UAVs perform trivial missions, while after 3M steps the reward stabilizes despite reaching the most complex maps, indicating gradual adaptation to higher difficulty.

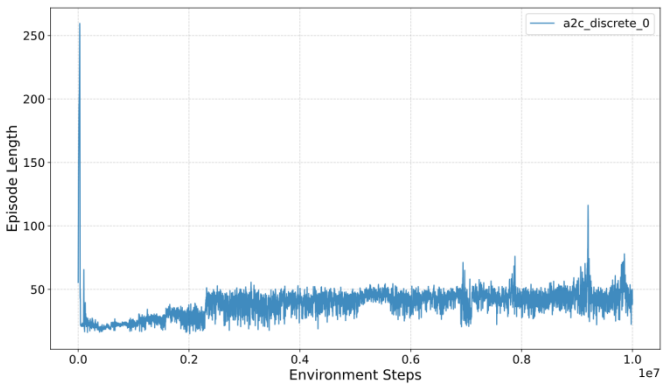

Figure 7. Average episodes reward in final training of Advantage Actor-Critic (A2C)

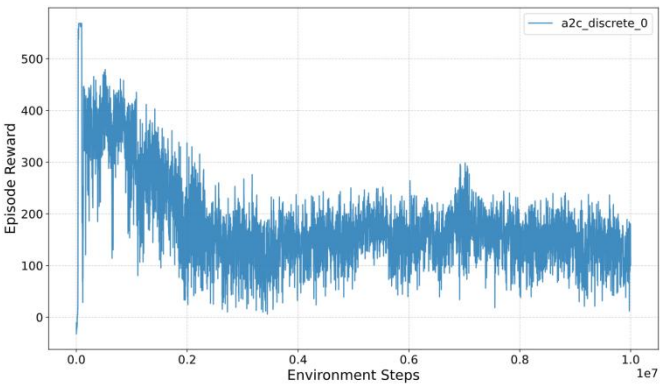

Figure 8. Average episodes reward in final training of Advantage Actor-Critic (A2C)

The graphs show that training for A2C was significantly more stable than for DQN, as there are no sharp changes in episode length and rewards, provided that the map type remains constant. The average number of steps at the end of training is 41.89, which is insufficient to complete the last map type, indicating that the agent often encounters obstacles. This also affects the average reward, which peaks at 310 during training for the most challenging map.

3.3 Proximal Policy Optimization training results

For PPO, training was performed on 20 million steps with the following parameters: learning rate = 0.001, n_steps = 1024, batch size = 128, and n_epochs = 4.

Figure 9 shows the average episode length during PPO training. An increase in episode duration is observed over the first 5 million steps, after which it stabilizes.

Figure 10 shows average episode reward in final training of PPO. We can observe that after 6M steps, PPO model learns to complete the mission in a consistent number of steps (episode length).

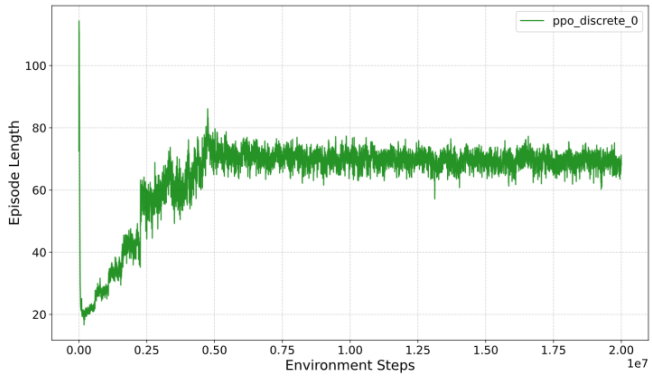

Figure 9. Average episodes length in final training of Proximal Policy Optimization (PPO)

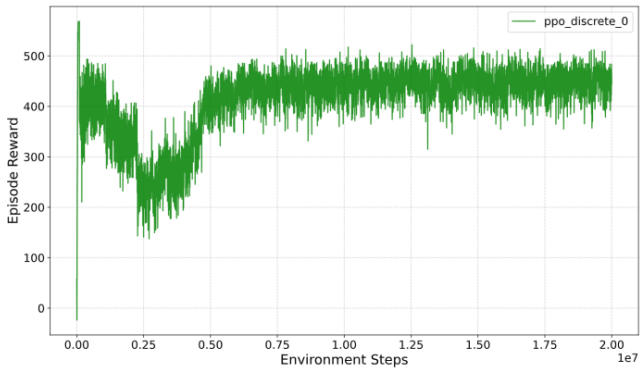

Figure 10. Average episodes reward in final training of Proximal Policy Optimization (PPO)

As can be seen from the average reward graph, after the last map change, there is a sharp decrease in reward. However, over time, the graph levels out, after which the reward begins to increase. This is unlike A2C, which gradually increases. The reward value for this training iteration reaches 513.3 in the middle of training for the last map type, and at the end of training, it is 447.38.

To further assess the consistency of training dynamics, additional training runs were conducted under identical initialization conditions. Figures 11 and 12 present the corresponding learning curves for PPO, showing episode length and episode reward, respectively.

The results demonstrate stable convergence behavior, with similar learning trajectories observed across runs. While variability is present during early training stages, convergence trends remain consistent, indicating that PPO training is reproducible under fixed conditions.

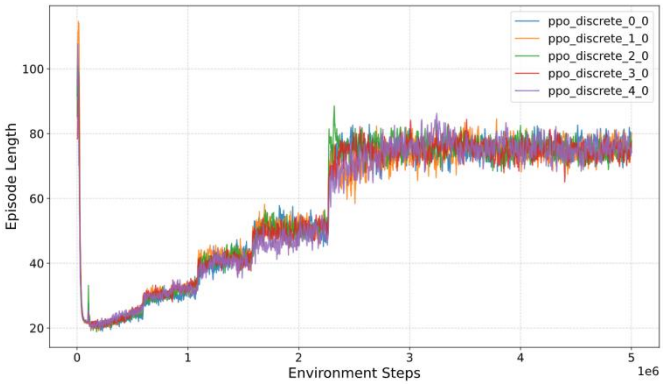

Figure 11. Proximal Policy Optimization (PPO) training – episode length across repeated runs under identical initialization

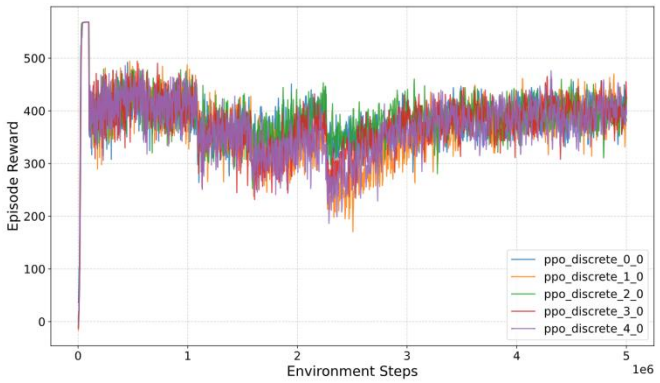

Figure 12. Proximal Policy Optimization (PPO) training – episode reward across repeated runs under identical initialization

3.4 Soft Actor-Critic training results

The SAC model was trained with a batch size of 128, update frequency of 4 steps, learning rate of 1e-3, and target smoothing coefficient of 0.005. As SAC operates in continuous action spaces, actions are represented as bounded continuous vectors consistent with the same physical limits. Figure 13 shows rapid convergence within the first 1.5M steps and stable performance as task complexity increases.

Training for this algorithm was conducted in 5 million steps and lasted over 11 hours, which is significantly longer than for the DQN and PPO algorithms. Figure 14 shows average episode reward in final training of SAC. After 1.5M steps mean episode length is increasing in steps, this is explained by progressive increase of task complexity.

With the change to the third map type, the reward began to grow rapidly, after which it stabilized at values of 540-550 for the last map. The average episode length was 74 steps, which is similar to the results obtained with PPO.

Similarly, additional training runs were performed for SAC under identical initialization conditions. Figures 15 and 16 present the learning curves for episode length and episode reward, respectively.

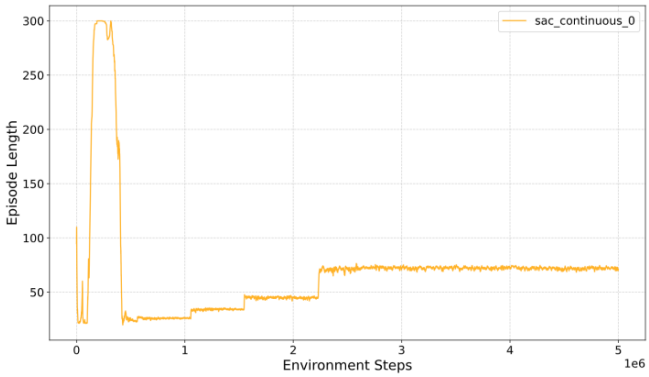

Figure 13. Average episodes length in final training of Soft Actor-Critic (SAC)

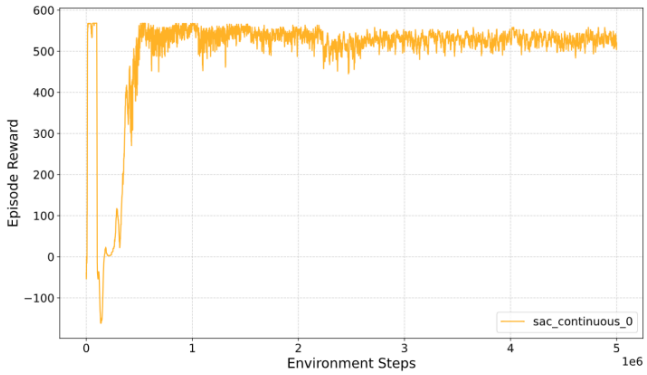

Figure 14. Average episodes reward in final training of Soft Actor-Critic (SAC)

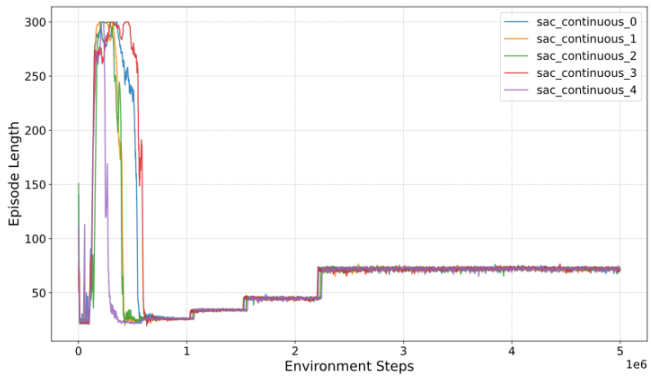

Figure 15. Soft Actor-Critic (SAC) training – episode length across repeated runs under identical initialization

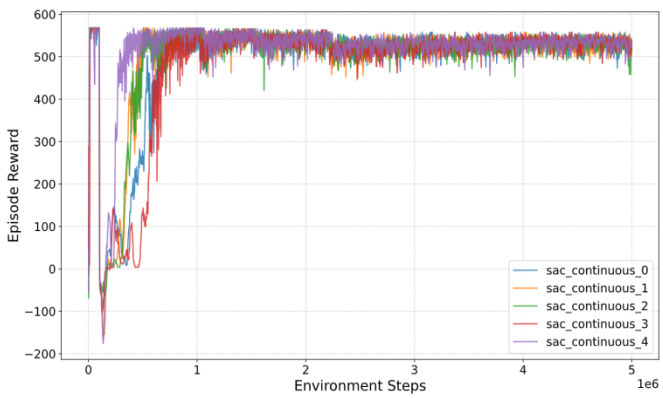

Figure 16. Soft Actor-Critic (SAC) training – episode reward across repeated runs under identical initialization

Compared to PPO, SAC exhibits faster convergence and smoother learning dynamics, with lower variability across runs. This indicates a more stable optimization process and reduced sensitivity to stochastic training effects under fixed initialization.

3.5 Deep Q-Network testing results

To verify the effectiveness of this model, a simulation environment with three UAVs was used, each controlled by a separate model, which exchanged information about position and speed. Testing was carried out on 100 episodes. Identical maps with the same placement of obstacles and targets were used for all algorithms. Figure 17 demonstrates the number of episodes where at least N UAVs reached targets controlled by the DQN model.

Figure 18 shows average episode reward and length grouped by number of episodes, where at least N number of UAVs reaches the target in final testing of DQN.

In most episodes during testing of the DQN algorithm, only one UAV reached the goal. The percentage of episodes without successful UAV was 46%. Low rewards and short episode lengths suggest that agents tend to move close to obstacles and frequently collide with them. As the environment became more complex, DQN lost its ability to learn effectively due to the larger number of input parameters and the increased number of possible states in which the agent could be.

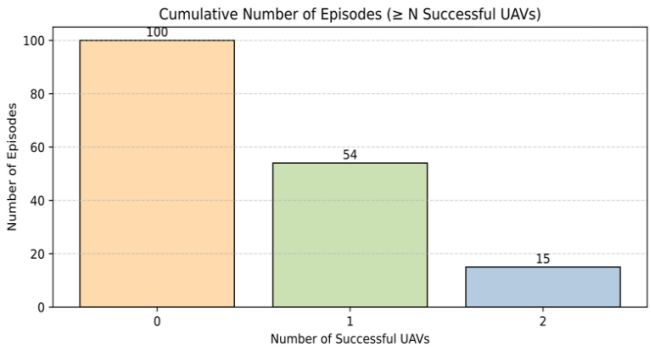

Figure 17. Number of episodes where at least N UAVs reached targets controlled by Deep Q-Network (DQN) model

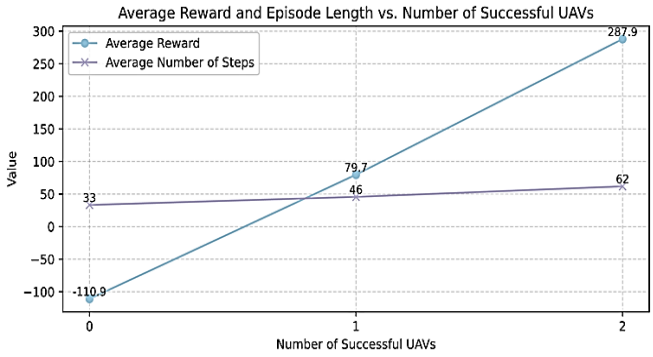

Figure 18. Average episodes reward and length in final testing of Deep Q-Network (DQN)

3.6 Advantage Actor-Critic testing results

Figure 19 demonstrates the number of episodes where at least N UAVs reached targets controlled by A2C model. A2C shows better results than DQN; we observe 2 episodes in which all 3 UAVs reach their targets.

Figure 20 shows average episode reward and length grouped by number of episodes, where at least N number of UAVs reaches the target in final testing of A2C.

The actor-critic algorithm with the priority function in a multi-agent environment yielded similar results to DQN, specifically, in 18 episodes, two agents successfully achieved their goals. In 50 episodes, at least one agent achieved the priority goal. Also, in 2 episodes, all three agents achieved their goals. The use of the preference function reduced the dispersion of gradients, which increased the stability of policy updates.

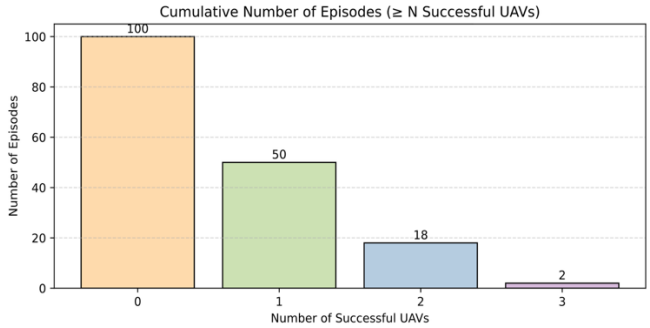

Figure 19. Number of episodes where at least N UAVs reached targets controlled by Advantage Actor-Critic (A2C) model

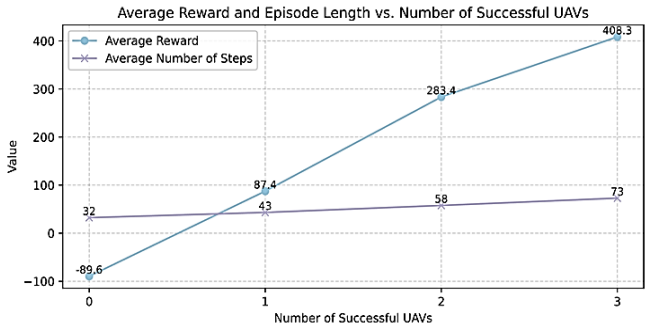

Figure 20. Average episodes reward and length in final testing of Advantage Actor-Critic (A2C)

3.7 Proximal Policy Optimization testing results

Figure 21 shows the number of episodes where at least N UAVs reached targets controlled by the PPO model. We can see that there are 26 episodes where all 3 UAVs reached their targets. This is 13 times better than A2C performance.

Figure 22 shows average episode reward and length grouped by number of episodes, where at least N number of UAVs reaches the target in final testing of PPO.

The results in a multi-agent environment for PPO are significantly better than the results of the two previous algorithms. All three goals were successfully achieved in 26 episodes, two or more goals in 68 episodes, and one or more goals in 95 episodes. Considering the change in priorities during the flight when one of the UAVs failed, the percentage of priority goal achievement for this training was 95%. The average reward for the three agents and the length of the episode naturally increases with the number of successfully achieved goals. PPO is more stable than A2C due to the limitation of policy changes between training iterations, which

reduces the risk of behavior gaps.

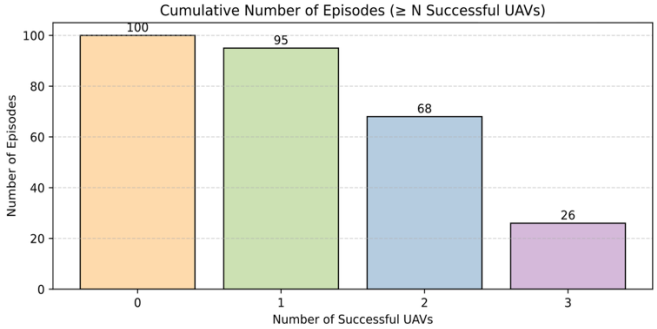

Figure 21. Number of episodes where at least N UAVs reached targets controlled by Proximal Policy Optimization (PPO) model

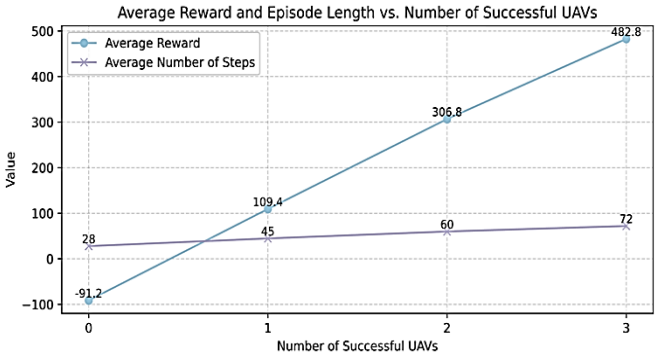

Figure 22. Average episodes reward and length in final testing of Proximal Policy Optimization (PPO)

3.8 Soft Actor-Critic testing results

Figure 23 shows the number of episodes where at least N UAVs reached targets controlled by the SAC model. We can see that the SAC model has the best results. There are 52 episodes where all 3 UAVs reached their targets. This is 2 times increase compared to PPO model results.

Figure 24 shows average episode reward and length grouped by number of episodes, where at least N number of UAVs reaches the target in final testing of SAC.

For SAC, the overall percentage of successful goal achievement by all UAVs increased significantly compared to PPO, reaching 52%. This result can be explained by the entropic approach to policy, which encourages broader exploration of actions and avoids premature convergence. The priority goal was achieved in 98% of cases, indicating the high reliability of this method, even in cases of collisions between one or more agents.

Summarizing the results, DQN performs the worst, with none of the episodes achieving all three goals. A2C is more stable during training, but the percentage of achieving the priority goal remains close to that of DQN, namely 50%. Figure 25 shows the number of episodes where at least N UAVs reached targets controlled by model. PPO proved to be the fastest of all algorithms in terms of training time, namely, for 20 million training steps, training took 16 hours. PPO also showed much better results, namely 95% for achieving the priority goal. The SAC algorithm demonstrated the best performance, reaching 98% of the priority goal and meeting all three goals in 52% of episodes.

These results provide empirical support for our hypothesis. The high success rates of PPO and SAC in the multi-agent testbed - despite only being trained on single-agent scenarios- confirm that sophisticated swarm behaviors can indeed emerge from individual training. The agents successfully generalized their "dynamic obstacle avoidance" policy to coordinate with other agents, effectively treating their swarm-mates as communicative, predictable obstacles.

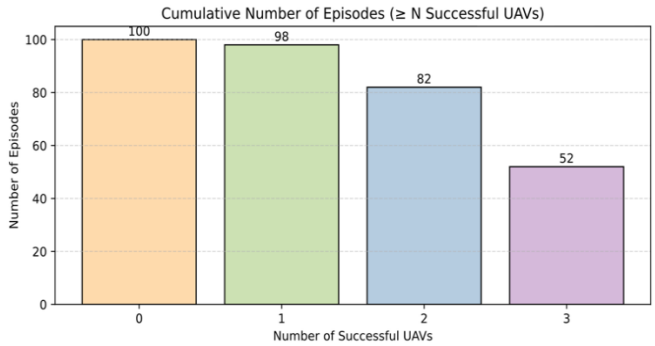

Figure 23. Number of episodes where at least N UAVs reached targets controlled by Soft Actor-Critic (SAC) model

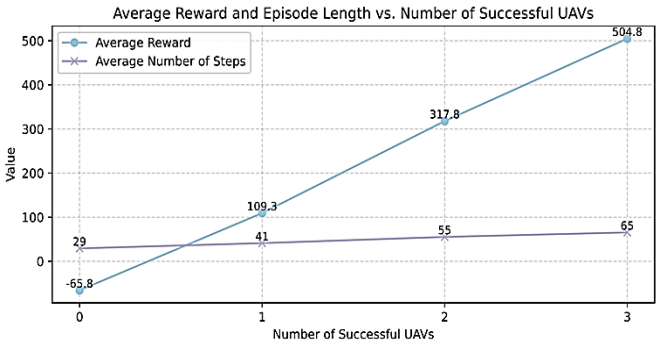

Figure 24. Average episodes reward and length in final testing of Soft Actor-Critic (SAC)

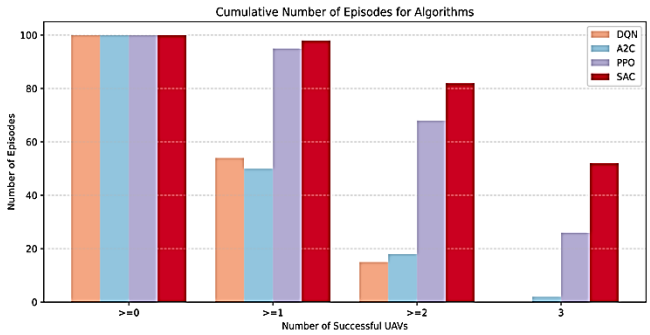

Figure 25. Number of episodes where at least N UAVs reached targets controlled by model

The aggregated results presented in Table 2 are obtained from a separate evaluation procedure and should be distinguished from the testing runs illustrated in Figures 17–25. In Figures 17–25, each algorithm is evaluated on the same set of predefined maps with a single episode per map, primarily to illustrate qualitative performance and outcome distribution. In contrast, for evaluation, each algorithm was tested on 10 predefined maps with identical initialization conditions (fixed seeds), and 10 independent episodes were executed per map, resulting in 100 evaluation runs per algorithm.

Results are reported as mean ± standard deviation over 100

evaluation episodes (10 maps × 10 runs).

The results (see Table 2) indicate that PPO and SAC achieve higher mean performance, with SAC additionally exhibiting lower variance, indicating more stable behavior across evaluation runs.

Table 2. Aggregated performance metrics (mean ± standard deviation over 100 evaluation episodes)

Algorithm	Priority Success	Agents Reached	Reward	Episode Time
DQN	0.58 ± 0.45	0.71 ± 0.60	436.3 ± 47.7	55.8 ± 2.74
A2C	0.46 ± 0.34	0.52 ± 0.42	403.7 ± 24.5	47.6 ± 1.51
PPO	0.91 ± 0.15	1.93 ± 0.87	424.2 ± 57.0	54.7 ± 2.37
SAC	0.92 ± 0.12	2.03 ± 0.68	472.7 ± 18.1	49.2 ± 1.14

4. DISCUSSION

Classical decentralized collision avoidance methods, such as Optimal Reciprocal Collision Avoidance, are widely used in multi-agent robotics due to their computational efficiency and ability to guarantee collision-free motion under ideal assumptions. While these methods provide deterministic collision avoidance, their integration with dynamic task allocation and operation under perception uncertainty remains non-trivial in civil protection scenarios. They rely on precise state estimation and reciprocal agent behavior, typically constructing velocity constraints to ensure safe navigation, but do not inherently address higher-level objectives such as adaptive task allocation.

In contrast, multi-agent reinforcement learning approaches (e.g., MADDPG, MAPPO) explicitly model inter-agent interactions during training. While effective, they introduce significant computational complexity, require careful reward design, and face scalability challenges.

The approach proposed in this work differs from both paradigms by training agents individually in dynamic environments and deploying them in a shared setting, where other agents are treated as interacting entities. This reduces training complexity while enabling coordinated behaviors such as collision avoidance and dynamic goal reassignment. Although this approach does not provide formal guarantees, the results demonstrate a favorable balance between performance, scalability, and computational efficiency for resource-constrained UAV platforms.

A limitation of this study is the absence of direct comparison with classical decentralized methods and multi-agent reinforcement learning approaches, which is planned for future work. Additionally, SAC operates in a continuous action space, while DQN, A2C, and PPO use discretized control, which may influence performance differences. Finally, no formal stability analysis under varying priority conditions is provided; the proposed mechanism is evaluated empirically and remains an important direction for future research.

While a formal stability analysis under dynamic priority reassignment is beyond the scope of this study, empirical observations indicate that the system maintains stable behavior across all evaluated scenarios, without oscillatory goal switching or persistent conflicts between agents. In practice, priority updates resulted in smooth trajectory adaptation, with agents converging to new targets without destabilizing the overall swarm motion. These findings suggest that the proposed decentralized policy-transfer approach provides sufficient robustness for real-world deployment, although formal guarantees remain an important direction for future research.

5. CONCLUSIONS

This study utilized the Gymnasium library, Stable-Baselines3, and AirSim to develop and evaluate a decentralized control framework for UAV swarms operating in dynamic civil emergency environments requiring high fault tolerance. Four Deep Reinforcement Learning algorithms—DQN, A2C, PPO, and SAC—were trained and compared across progressively more complex navigation scenarios. The results demonstrate significant differences in algorithm performance. DQN showed promise in simple 2D environments but failed to generalize to complex 3D multi-agent scenarios, achieving only a 54% priority-goal success rate. A2C improved upon DQN but remained insufficient for critical multi-agent tasks. PPO substantially increased performance, achieving a 95% success rate for individual goal completion, with all agents successfully reaching their initial goals in 26 episodes. SAC delivered the best overall results, achieving a 98% priority-goal success rate and increasing the successful completion of cooperative three-agent missions to 52%.

These findings indicate that SAC is the most suitable algorithm among those evaluated for autonomous UAV swarm deployment in civil protection and disaster management applications, where reliable navigation in hazardous, unknown environments is essential. Moreover, the results validate the central hypothesis of this work: coordinated swarm behavior, including collision avoidance and task coordination, can emerge from individually trained agents using only limited local state information, thereby enabling scalable decentralized multi-robot systems.

A limitation of this study is the use of both discrete (DQN, A2C, PPO) and continuous (SAC) action spaces, which affects the direct comparability of the evaluated algorithms. Although all methods operate under identical physical constraints, differences in action-space representation influence learning dynamics and final performance. A more uniform comparison across action spaces remains an important direction for future research.

This work also represents a foundational step toward deploying autonomous UAV swarms on low-power edge hardware. Future research will focus on validating the trained SAC models on physical UAV platforms equipped with Single Board Computers (SBCs), integrating the learned control policies with real-world perception systems, including visual odometry for localization and depth estimation for obstacle detection. These experiments will evaluate whether the emergent swarm behaviors observed in simulation, such as decentralized coordination, collision avoidance, and fault tolerance, remain robust under real-world sensing uncertainties and communication constraints. Additionally, future work will benchmark the proposed DRL approach against classical path-planning methods, including A*, Rapidly-exploring Random Trees (RRT), and Potential Fields, as well as decentralized planners and multi-agent

reinforcement learning techniques. We also plan to investigate hybrid navigation architectures that combine classical global path planning with DRL-based local reactive control and dynamic collision avoidance, providing a balanced solution for computationally constrained civil protection missions.

ACKNOWLEDGMENT

The authors extend their appreciation to Universiti Teknikal Malaysia Melaka (UTeM) for their support in this research.

REFERENCES

[1] Alqudsi, Y., Makaraci, M. (2025). UAV Swarms: Research, challenges, and future directions. Journal of Engineering and Applied Science, 72: 12. https://doi.org/10.1186/s44147-025-00582-3

[2] Bayerlein, H., Theile, M., Caccamo, M., Gesbert, D. (2021). Multi-UAV path planning for wireless data harvesting with deep reinforcement learning. IEEE Open Journal of the Communications Society, 2: 1171-1187. https://doi.org/10.1109/OJCOMS.2021.3081996

[3] Chaurasia, R., Mohindru, V. (2021). Unmanned aerial vehicle (UAV): A comprehensive survey. In Unmanned Aerial Vehicles for Internet of Things (IoT): Concepts, Techniques, and Applications, Wiley, pp. 1-27. https://doi.org/10.1002/9781119769170.ch1

[4] Mochurad, L., Shakhovska, N., Bacarra, R., Alsayaydeh, J.A.J. (2025). Decentralized drone swarm coordination methods using reinforcement learning and parallel computing for optimized management systems. International Journal of Intelligent Engineering and Systems, 18(5): 83-99. https://doi.org/10.22266/ijies2025.0630.07

[5] Chen, Y., Dong, Q., Shang, X.Z., Wu, Z.Y., Wang, J.Y. (2023). Multi-UAV autonomous path planning in reconnaissance missions considering incomplete information: A reinforcement learning method. Drones, 7(1): 10. https://doi.org/10.3390/drones7010010

[6] Garg, P.K. (2022). Characterisation of fixed-wing versus multirotors UAVs/drones. Journal of Geomatics, 16(2): 152-159. https://doi.org/10.58825/jog.2022.16.2.44

[7] Puente-Castro, A., Rivero, D., Pazos, A., Fernandez-Blanco, E. (2022). A review of artificial intelligence applied to path planning in UAV swarms. Neural Computing and Applications, 34: 153-170. https://doi.org/10.1007/s00521-021-06569-4

[8] Marek, D., Paszkuta, M., Szyguła, J., Biernacki, P., Domański, A., Szczygieł, M., Król, M., Wojciechowski, K. (2024). Swarm of drones in a simulation environment-efficiency and adaptation. Applied Sciences, 14(9): 3703. https://doi.org/10.3390/app14093703

[9] Haarnoja, T., Zhou, A., Abbeel, P., Levine, S. (2018). Soft Actor-Critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. arXiv preprint arXiv:1801.01290. https://doi.org/10.48550/arXiv.1801.01290

[10] Meier, L., Honegger, D., Pollefeys, M. (2015). PX4: A node-based multithreaded open source robotics framework for deeply embedded platforms. In 2015 IEEE International Conference on Robotics and Automation (ICRA), Seattle, WA, USA, pp. 6235-6240. https://doi.org/10.1109/ICRA.2015.7140074

[11] Masroor, R., Naeem, M., Ejaz, W. (2021). Resource management in UAV-assisted wireless networks: An optimization perspective. Ad Hoc Networks, 121: 102596. https://doi.org/10.1016/j.adhoc.2021.102596

[12] Katona, K., Neamah, H.A., Korondi, P. (2024). Obstacle avoidance and path planning methods for autonomous navigation of mobile robot. Sensors, 24(11): 3573. https://doi.org/10.3390/s24113573

[13] Yang, Y.H., Xiong, X.Z., Yan, Y.H. (2023). UAV formation trajectory planning algorithms: A review. Drones, 7(1): 62. https://doi.org/10.3390/drones7010062

[14] Wang, H., Pan, W.J. (2021). Research on UAV path planning algorithms. IOP Conference Series: Earth and Environmental Science, 693(1): 012120. https://doi.org/10.1088/1755-1315/693/1/012120

[15] Xushi, W.S. (2024). Research on quadrotor UAV control and path planning based on PID controller and Dijkstra algorithm. AIP Conference Proceedings, 3144(1): 030015. https://doi.org/10.1063/5.0214314

[16] Alsayaydeh, J.A.J., Irianto, Aziz, A., Xin, C.K., Hossain, A.K.M.Z., Herawan, S.G. (2022). Face Recognition System Design and Implementation using Neural Networks. International Journal of Advanced Computer Science and Applications, 13(6): 519-526. https://doi.org/10.14569/IJACSA.2022.0130663

[17] Liaskovska, S., Izonin, I., Martyn, Y. (2022). Investigation of anomalous situations in the machine-building industry using phase trajectories method. In Advances in Computer Science for Engineering and Manufacturing. Springer, Cham, pp. 49-59. https://doi.org/10.1007/978-3-031-03877-8_5

[18] Al-Andoli, M.N., Irianto, Alsayaydeh, J.A., Alwayle, I.M., Che Ku Mohd, C.K.N., Abuhoureyah, F. (2024). Robust overlapping community detection in complex networks with graph convolutional networks and fuzzy C-means. IEEE Access, 12: 70129-70145. https://doi.org/10.1109/ACCESS.2024.3399883

[19] Rabarinjatovo, T., Ravalison, F. (2025). Systemic determinants of equipment failure in paper mills: A hybrid FAST-PCA approach for maintenance optimization. Journal of Industrial Intelligence, 3(2), 60-68. https://doi.org/10.56578/jii030201

[20] Alsayaydeh, J.A.J., Irianto, Zainon, M., Baskaran, H., Herawan, S.G. (2022). Intelligent interfaces for assisting blind people using object recognition methods. International Journal of Advanced Computer Science and Applications, 13(5): 84-92. https://doi.org/10.14569/IJACSA.2022.0130584

[21] Jiménez, G.A., de la Escalera Hueso, A., Gómez-Silva, M.J. (2023). Reinforcement learning algorithms for autonomous mission accomplishment by unmanned aerial vehicles: A comparative view with DQN, SARSA and A2C. Sensors, 23(21): 9013. https://doi.org/10.3390/s23219013

[22] Khan, A.A., Laghari, A.A., Bacarra, R., Alroobaea, R., Algamdi, S., Baqasah, A.M., Alsayaydeh, J.A.J. (2025). Cybersecurity, digital forensics, and the IoT for deepfake investigation on social media platforms: A review. Human-Centric Intelligent Systems, 15: 38. https://doi.org/10.22967/HCIS.2025.15.038

[23] Alsayaydeh, J.A.J., Irianto, Ali, M.F., Al-Andoli, M.N.M., Herawan, S.G. (2024). Improving the Robustness of IoT-Powered Smart City Applications

Through Service-Reliant Application Authentication Technique. IEEE Access, 12: 19405-19417. https://doi.org/10.1109/ACCESS.2024.3361407

[24] Iatsyshyn, V. (2025). Particle filter application to radio-based range-only UAV self-localization. In Developments in Information and Knowledge Management Systems for Business Applications, Springer, Cham, pp. 149-165. https://doi.org/10.1007/978-3-031-80935-4_8

[25] Terzi, S., Eriskin, E. (2026). A hybrid modelling architecture for predicting rheological performance of waste frying oil–modified asphalt binders via stochastic–physical integration. Journal of Hybrid Modelling and Intelligent Engineering Systems, 1(1): 1-7. https://doi.org/10.56578/jhmies010101

[26] Hou, Q.T., Yuan, X.X., Zhao, J.W., Zhang, Y.Y., Gao, H.Y., Fu, X.W. (2024). Automatic leveling algorithm for a three-degree-of-freedom air-floating platform with uncertain inertia. Journal of Intelligent Systems and Control, 3(4): 201-212. https://doi.org/10.56578/jisc030401