POSING FLOCKING AS A MARL PROBLEM

A REPORT

submitted in partial fulfillment of the requirements

for the award of the dual degree of

Bachelor of Science - Master of Science

in

ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

by

ANANYA GANDHI

(20319)

Under the guidance of

DR P. B. SUJIT

DEPARTMENT OF ENGINEERING SCIENCES
INDIAN INSTITUTE OF SCIENCE EDUCATION AND RESEARCH BHOPAL, BHOPAL - 462066

April 2025

भारतीय विज्ञान शिक्षा एवं अनुसंधान संस्थान भोपाल
Indian Institute of Science Education and Research
Bhopal
(Estb. By MHRD, Govt. of India)

CERTIFICATE

This is to certify that **Ananya Gandhi**, BS-MS (Electrical Engineering and Computer Science), has worked on the project entitled **'Posing Flocking as a MARL Problem'** under my supervision and guidance. The content of this report is original and has not been submitted elsewhere for the award of any academic or professional degree.

April 2025
IISER Bhopal

Dr P. B. Sujit
(IISER Bhopal)

Committee Member	**Signature**	**Date**
Prof. Sujit Pedda Baliyarasimhuni		
Prof. Santanu Talukder		
Prof. Ankur Raina		
Prof. Arijit Sen		

ACADEMIC INTEGRITY AND COPYRIGHT DISCLAIMER

I hereby declare that this project is my own work and, to the best of my knowledge, it contains no materials previously published or written by another person, or substantial proportions of material which have been accepted for the award of any other degree or diploma at IISER Bhopal or any other educational institution, except where due acknowledgment is made in the document.

I certify that all copyrighted material incorporated into this document is in compliance with the Indian Copyright Act (1957) and that I have received written permission from the copyright owners for my use of their work, which is beyond the scope of the law. I agree to indemnify and save harmless IISER Bhopal from any and all claims that may be asserted or that may arise from any copyright violation.

April 2025 **Ananya Gandhi**
IISER Bhopal

ACKNOWLEDGEMENT

First, I would like to thank my advisor, Prof. Sujit P. B.

I have been extremely fortunate to have had him as my PI and to be a part of his MOON Lab for the last 3 years. My journey has been an exciting and motivating one, and most fun times of my college years have been spent with them. Sujit Sir and MOON Lab have directly and indirectly helped me not only to realise my passion for swarm robotics and machine learning methods but they have also made me better learner in life.

I want to give my sincerest gratitude to Sir for providing us with the space, time and freedom to explore research in a fun way and providing a very healthy lab environment. He has established the image of an ideal lab environment & professor in my mind, and going forth in my academic journey in life, I hope to do my best to live up to this image as well.

I am forever indebted to my parents, for being the unwavering support as always. Gratitude cannot be expressed enough through words for everything they have done for me in the last 23 years which makes it possible for me to be here. Without them, the world stops making sense. Like Dada says, "The only To-Be that matters is being a good human being", and I hope to keep trying my best. Thank you, Mummy-Dada.

Next, I would give credit to the family I made here at IISERB– my brothers: Rehan and Devashish, who made IISERB the home it feels now. Throughout the 2 years, for being a constant source of motivation and laughs and a steadfast sink for all the rants and complaints. Thank you for always rushing in to save me at the last moments of my deadlines. I hope to be better, one day, if only it would mean that I have found another source for annoying you guys XD. *The Radical Dreamers* shall keep on Dreaming!

I would like to thank Isha for being an awesome friend. Taking on roles of a sister, mother, teacher, as required to take care of me XD. Presentations would have been impossible without you.

I am grateful for having lab-mates like my fellow *Rats of MoonLab*. ~~We're pretty sure the Moon isn't made of cheese ;)~~ I am grateful to everyone for always looking out for me and making research seem like an adventure while getting high on uncanny doses of tea and caffeine. I will miss the lab meetings, the MARL discussion sessions, all the inside jokes, the lab, trips and all the fun times with you guys.

A nod to *Illuminati Confirmed* for everything, literally; to *Rule: no pda* for being the most fun partner-in-crimes & partner-detectives; to *Safe Haven* for being safe haven; and to *The Spy-Stalkers Club* for the Treasure Hunts.

I would like to thank Pratik for introducing me to swarms 3 years back; to Rohan for imbuing me with random spells of hope; to Janhavi; to the very lovely Book Club of IISERB; to my family- especially Soham Bhaiyya; and finally the countless researchers, authors, and visionaries whose works have inspired and informed my own. I hope I shall one day be like them and contribute something of meaning to this world.

This journey has been a mosaic of equations, epiphanies, and existential crises— and I wouldn't have had it any other way.

Lastly, I thank the reader for giving this research work your time and hope you find it worthy enough to do so.

Ananya Gandhi

ABSTRACT

This thesis explores the application of Multi-Agent Reinforcement Learning (MARL) to achieve decentralized flocking behavior in autonomous drone swarms. By formulating flocking as a reinforcement learning problem, we develop a mathematical framework defining state spaces, action spaces, and reward functions that balance individual and collective objectives. Our study compares two MARL algorithms—Proximal Policy Optimization (PPO) and Soft Actor-Critic (SAC)— to evaluate their effectiveness in environments with partial observability. Results demonstrate that properly designed reward functions enable emergent flocking behaviours with cohesion, alignment, and collision avoidance capabilities. Experiments further evaluate the policies' performance under varying conditions, including the presence of obstacles, and varying number of drones in the environment.

Keywords: multi-agent reinforcement learning, flocking behaviour, swarm robotics, decentralized control, emergent behaviour, reward designing

LIST OF ABBREVIATIONS

MARL	Multi agent reinforcement learning
PPO	Proximal Policy Optimisation
POMDP	Partially Observable Markov Decision Process
LR	Learning Rate
SAC	Soft Actor-Critic
POMG	Partially Observable Markov Game
MDP	Markov Decision Process
DQN	Deep Q-Network
CTDE	Centralized Training with Decentralized Execution
MADDPG	Multi-Agent Deep Deterministic Policy Gradient
QMIX	Q-learning-based multi-agent mixing network
MAPPO	Multi-agent Proximal Policy Optimisation
MAAC	Multi-Agent Actor-Critic
MA-POCA	Multi-Agent Policy Optimization with Credit Assignment
IRL	Inverse Reinforcement Learning
QTRAN	Q-value Transformer for Multi-Agent Reinforcement Learning
DL	Deep Learning
UAV	Unmanned Aerial Vehicle
GNN	Graph Neural Network
LSTM	Long Short-Term Memory (neural network)
GPU	Graphics Processing Unit
RAM	Random Access Memory
YAML	YAML Ain't Markup Language (used for config files)
PPO-C	Proximal Policy Optimization with Curriculum Learning
CT	Curriculum Training (used contextually, not formalized)

LIST OF SYMBOLS

S	Set of possible states
A	Set of possible actions
$P(s'\|s, a)$	Transition probability function
$R(s, a, s')$	Reward function
γ	Discount factor $\in [0, 1]$
$V^{\pi}(s)$	State value function under policy π
$Q^{\pi}(s, a)$	Action value function under policy π
V^{*}	Optimal state value
Q^{*}	Optimal action value
α	Learning rate
$\pi_{\theta}(a\|s)$	Policy parameterized by θ
$\nabla_{\theta} J(\theta)$	Policy gradient
v_x, v_z	Local velocity of a drone in X and Z directions
$d_{1...24\_obstacles}$	Distances to obstacles via 24 raycasts
$d_{1...24\_drones}$	Distances to other drones via 24 raycasts
currentSensed	Number of drones sensed at current timestep
prevSensed	Number of drones sensed at previous timestep
totalSensed	Total number of drones sensed so far
swarmDrones.Count	Size of the swarm the agent belongs to
action[0]	Left/Right control action (X-axis)
action[1]	Forward/Backward control action (Z-axis)
$\Delta v_x, \Delta v_z$	Continuous control changes to X and Z velocity
Rtotal	Total reward for an agent
Rswarm	Reward for maintaining swarm formation
Rproximity	Reward for being in optimal dist range from others

Rsurvival	Reward for surviving (not crashing)
Robstacle	Penalty for proximity to or collision with obstacles
Rboundary	Penalty for crossing the environment boundary
Rdisintegration	Penalty for losing swarm cohesion
Rcollision	Penalty for drone-to-drone collision/close proximity
Rstagnation	Penalty for staying in the same area too long
rr_factor	Scaling factor for rebound reward component
swarmationReward	Base reward for staying in a swarm (200.0)
insideGoodRegionReward	Reward for being in the optimal zone (80.0)
insideSensingZoneReward	Reward for being within sensing range (20.0)
survivalReward	Reward for staying alive (20.0)
obstacleCollisionPenalty	Penalty for hitting an obstacle (-100.0)
boundaryCollisionPenalty	Penalty for boundary violation (-100.0)
disintegrationPenalty	Penalty per lost swarm member (-750.0)
intraSwarmCollisionPenalty	Penalty for drone-to-drone collision (-180.0)
tooClosePenalty	Penalty for being too close to another drone(-50.0)
insideBadRegionPenalty	Penalty for being in undesirable zone (-30.0)
stagnationPenalty	Penalty for not moving or exploring (-5.0)
Rtooclose	Distance below which tooClosePenalty is applied
Rin	Inner radius of good region
Rout	Outer radius of good region
Rsense	Maximum sensing radius

LIST OF FIGURES

LIST OF TABLES

CONTENTS

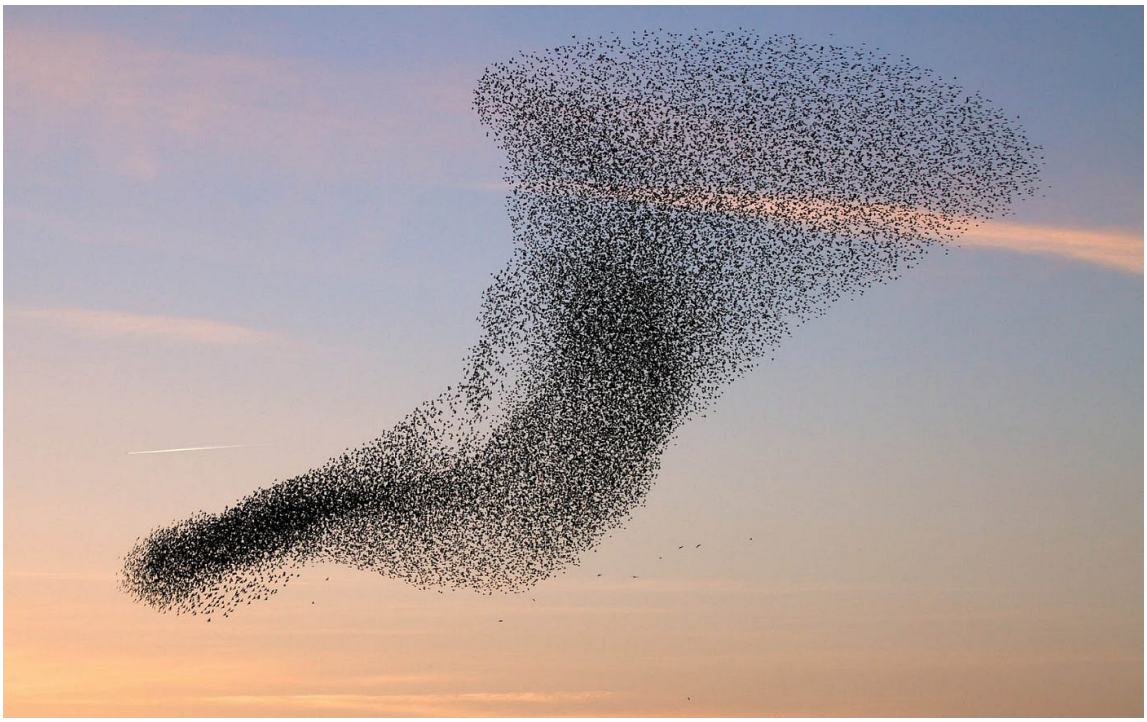


Figure: A murmuration of starlings forms a dynamic, fluid shape against the evening sky — a striking example of collective behavior and natural coordination without a central leader.

1. CHAPTER 1: INTRODUCTION

1.1 Overview

A swarm is a real-time system of individual networks (brains) with feedback loops, which have naturally evolved into highly-rewarding behaviours by adhering to a swarming algorithm. The swarming algorithm assesses the strength of the conviction of each agent and determines how the swarm should move at every instant. Each agent acts, reacts, and interacts with other agents in the environment, and a better path is found as we progress, the one that optimises the collective satisfaction of the group.

Robot swarms are decentralized collective systems of simple embodied agents that act autonomously and rely on local information only. [7]

The inspiration for robotic swarms have been drawn from the natural phenomenon where individual agents, such as birds, fish, or drones, coordinate their actions to achieve collective goals.[4] The aim of imitating such organisms is to increase the system's robustness and leverage the advantages that come with the numerosity of agents in such a swarm. [14]

We look at Multi-Agent Reinforcement Learning (MARL) as a way to model and bring out emergent behavior. MARL is better than traditional rule-based systems for dynamic, unpredictable environments because it lets agents learn adaptive strategies through interaction and optimization. By using MARL, agents can learn to form swarms, avoid obstacles, and other

cooperative strategies without anyone needing to explicitly program these strategies. This approach is more flexible and scalable, because it can adapt to the complex interaction and changes that happen in the field.

1.2 Key Questions

This research seeks to address the following questions:

1. How can MARL overcome the limitations of rule-based flocking models?
2. What are the advantages of MARL in dynamic, real-world scenarios?
3. How can reward structures and architectures be designed to balance local and global objectives?

1.3 Background

1.3.1 Flocking in Nature

Flocking is a natural phenomenon observed in animals like birds and fish, characterized by coordinated group motion. These behaviors are governed by three primary rules:

1. **Alignment:** Matching the direction of nearby agents.
2. **Cohesion:** Staying close to the group center.
3. **Separation:** Avoiding collisions with neighbors.

1.3.2 Traditional Flocking Models

Traditional models, such as Reynolds' Boids and the Vicsek Model, rely on predefined rules to simulate flocking behavior. For eg:

1. **Reynolds' Boids:** Introduced alignment, cohesion, and separation rules for simulating flocking.

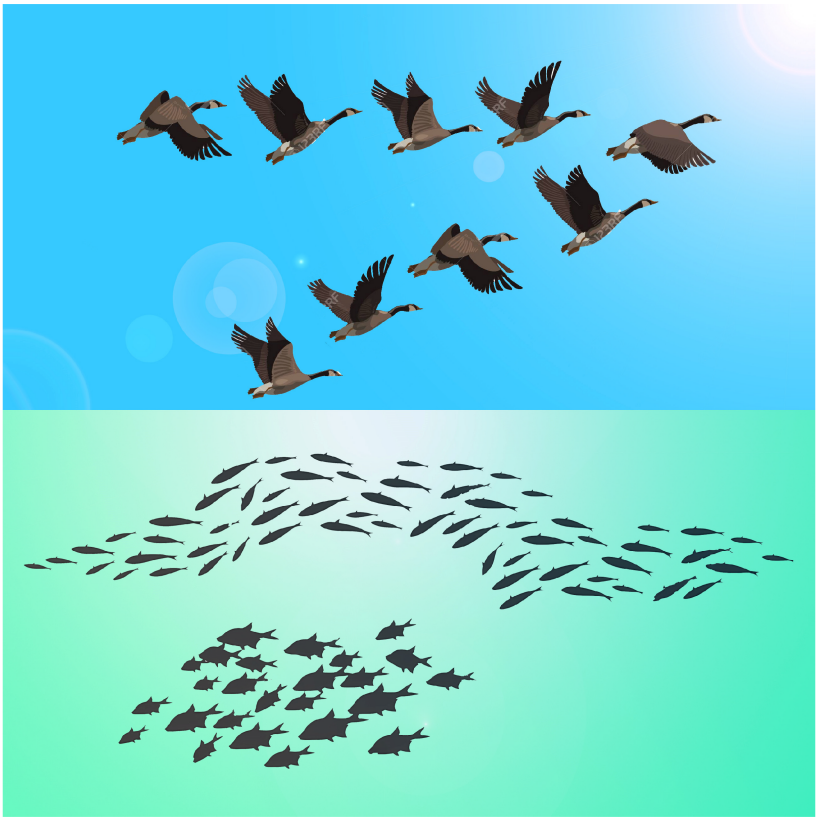


Fig. 1.1: Flocking behavior observed in birds and fish. Image taken from [1]

2. **Vicsek Model:** Models flocking as a system of self-propelled particles with noise.

Tab. 1.1: Comparison of traditional flocking models.

Model	Strengths	Limitations
Reynolds' Boids	Simple and computationally efficient	Struggles in dynamic environments
Vicsek Model	Captures emergent behaviors	Assumes global observability

1.4 Problem Statement

The goal of this work is to explore how Multi-Agent Reinforcement Learning (MARL) can address the limitations of rule-based flocking models in dynamic, partially observable environments. Specifically, we aim to answer:

> *How can MARL agents learn to achieve emergent flocking behaviors while adapting to dynamic and complex environments?*

Key Challenges

1. **Scalability:** Coordinating large swarms of agents while maintaining computational efficiency.

2. **Balancing Local and Global Objectives:** Ensuring agents optimize their individual goals (e.g., avoiding collisions) while contributing to the collective behavior (e.g., forming swarms).

3. **Partial Observability:** Agents have limited sensing capabilities, making it difficult to achieve global coordination.

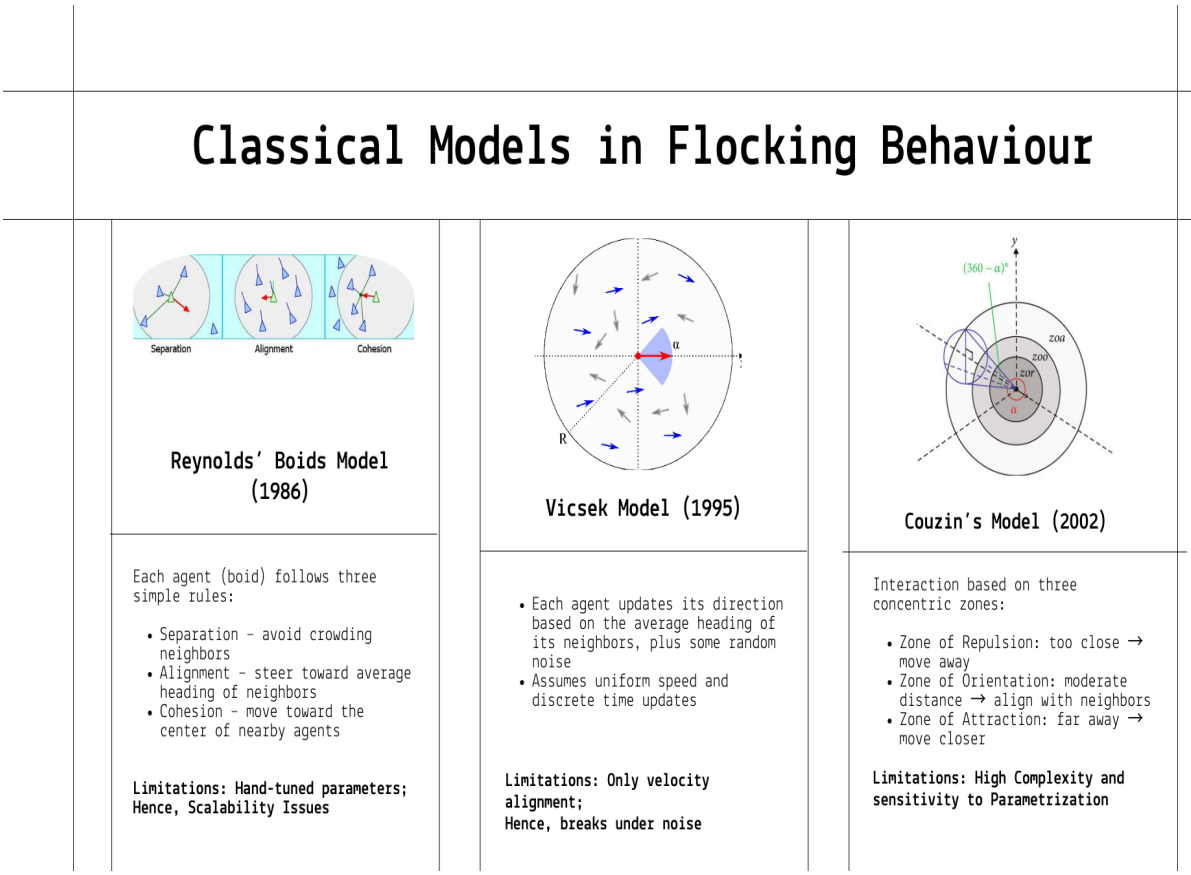


Fig. 1.2: Comparison of traditional flocking models.

1.5 Motivation

1.5.1 Why MARL for Flocking?

1. **Emergent Behaviors:** MARL enables agents to discover strategies beyond handcrafted rules, allowing for more flexible & adaptive behaviors.
2. **Adaptability:** MARL-trained agents can adjust to real-world perturbations, such as wind, obstacles, or changing goals.

1.5.2 Applications

1. **Robotics:** Drone swarms for search-and-rescue or surveillance.
2. **Autonomous Vehicles:** Coordinated navigation in traffic systems.
3. **Crowd Simulation:** Modeling realistic group dynamics in virtual environments.

Tab. 1.2: Applications of MARL-based flocking.

Application	Description
Search-and-Rescue	Coordinated drone swarms in GPS-denied environments.
Traffic Management	Collaborative navigation of autonomous vehicles to reduce congestion.
Crowd Simulation	Realistic modeling of collective motion in virtual environments.

1.6 Contributions

This work makes the following contributions:

1.6.1 Framework

We formalize flocking as a **Partially Observable Markov Game (POMG)**, defining MARL-compatible state spaces, action spaces, and reward structures.

1.6.2 Reward Design

We propose a hybrid reward function that combines Couzin's Model-inspired physical set-up with objectives of survivability and swarm cohesion, along with obstacle avoidance and environment exploration.

1.6.3 Open-Source Implementation

We provide an open-source implementation using **Unity ML-Agents** and **PyTorch**, enabling reproducibility and further research.

1.7 Thesis Outline

This thesis is organized as follows:

1. **Introduction:** Overview, background, problem statement, motivation, and contributions.
2. **Discussions in Flocking and Swarm Intelligence:** Biological foundations, classical models (Boid's, Vicsek, Couzin's), and transition to MARL-based approaches.
3. **Discussions in Reinforcement Learning:** Core algorithms (PPO, SAC), MARL paradigms (CTDE), and challenges in multi-agent systems.
4. **Theoretical Framework:** POMG formulation, state/action space design, and reward function components for decentralized flocking.
5. **System Framework and Methodology:** Simulation environment, agent architecture, and training pipeline (curriculum learning, hyperparameters).
6. **Results:** Analysis of training performance, emergent behaviors, and comparisons with rule-based baselines.
7. **Conclusion and Future Work:** Summary of contributions and directions for future research.

2. CHAPTER 2: DISCUSSIONS IN FLOCKING AND SWARM INTELLIGENCE

2.1 Biological and Classical Foundations of Flocking

2.1.1 Biological Foundations of Flocking

Flocking behaviour emerges in nature as a complex yet self-organized phenomenon governed by local rules rather than centralized control. Starling murmurations, fish schooling, and insect swarming exemplify systems where simple interaction rules lead to intricate, large-scale patterns [3].

1. **Starling murmurations (flocks):** Birds move together quickly as one group, reacting instantaneously to external threats to stay safe from danger.

2. **Fish schools:** Swim together to avoid predators and save energy. They feel water movements to stay in sync. [10].

3. **Insect swarms:** Ants and bees use pheromone trails and local cues to work together to achieve global goals such as foraging for food and nest construction.

Key Biological Principles:

Emergent behaviours observed in nature are governed by a few fundamental principles:

Self-Organization:
Groups achieve global coordination through local interactions, without centralized control.
Example: Each bird in a flock adjusts its position based on its immediate neighbours.

Local Interaction Rules:
Simple rules like alignment, cohesion, and separation drive complex group dynamics.

Positive and Negative Feedback Mechanisms:
Positive feedback amplify beneficial behaviours (e.g., ants reinforcing trails). Negative feedback prevent overcrowding or collisions (e.g., separation in flocks).

2.1.2 Classical Models of Flocking Behaviour

Reynolds' Boids Model

Craig Reynolds' Boids model simulates flocking using three simple rules:

- **Alignment**: Birds adjust their direction to match the average direction of nearby birds.
- **Cohesion**: Birds move toward the centre of their local group.
- **Separation**: Birds maintain some distance to avoid collisions with neighbours.

The alignment force is calculated using this equation:

$$\mathbf{F}_{\text{align}} = \frac{1}{N} \sum_{j \in \mathcal{N}_i} \mathbf{v}_j - \mathbf{v}_i \tag{2.1}$$

Where:

- $\mathbf{F}_{\text{align}}$ is the alignment force vector
- N is the number of neighbouring birds
- $\mathbf{v}_j$ represents the velocity vector of neighbour j
- $\mathbf{v}_i$ is the current bird's velocity vector
- $\mathcal{N}_i$ is the set of all neighbours within perception range

Despite its simplicity, this model creates remarkably realistic flocking patterns.

Strengths: Biologically plausible and computationally efficient.

Limitations: Requires parameter tuning and doesn't adapt well to changing environments.

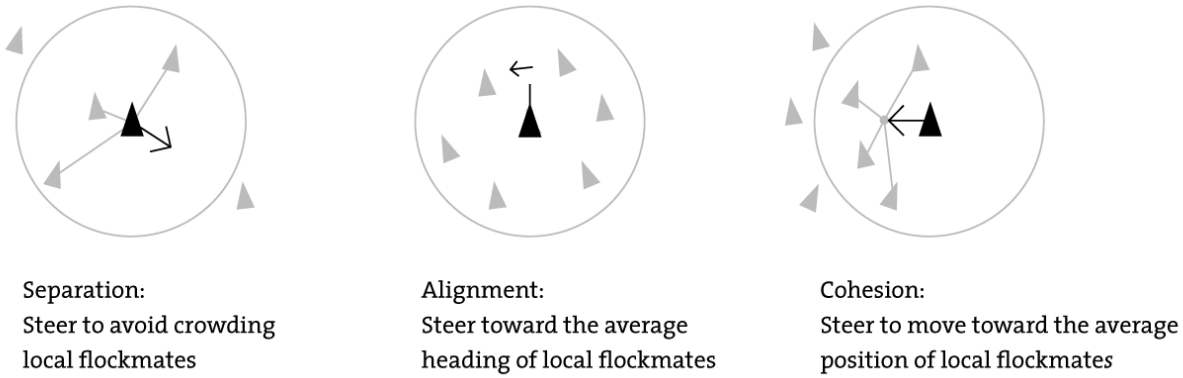


Fig. 2.1: Interaction of the three Boids rules

Vicsek Model

Simplifies flocking by focusing on velocity alignment:
Each agent aligns its velocity to neighbours' average velocity, with some added noise.

Key Features: presence of a phase transition from disordered movement (agents moving randomly) to ordered motion (agents moving in the same direction), depending on two parameters:

1. the noise level (η), and
2. the interaction radius (r_s or r_v).

Comparison to Boids: Simpler but lacks cohesion and separation.

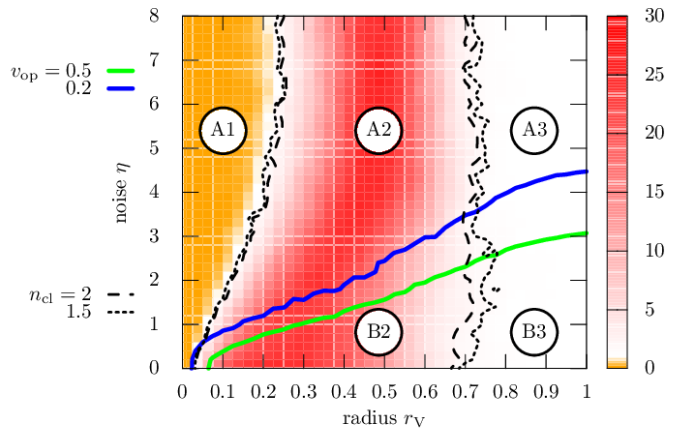


Fig. 2.2: Phase diagram of the Vicsek model showing the effect of noise (η) and interaction radius r_V on collective motion. Coloured regions indicate the degree of clustering, with warmer colours representing higher clustering. Solid lines represent phase boundaries for different particle speeds ($v_{op} = 0.2$ and 0.5), while dashed lines correspond to average cluster numbers ($n\_cl = 1.5$ and 2). Distinct behavioural regimes (A1–A3: disordered; B2–B3: ordered/clustering) illustrate transitions from disorder to collective motion as noise decreases and interaction radius increases.
(As noise decreases or interaction radius increases, agents are more likely to align and move cohesively.)

Olfati-Saber's Potential Fields

Olfati-Saber introduced attractive and repulsive potentials:

1. **Attractive Potential:** Encourage agents to move toward group centre.
2. **Repulsive Potential:** Prevent collisions with agents/obstacles.

$$\mathbf{F}_{\text{total}} = -\nabla U(\mathbf{x}) \tag{2.2}$$

where $U(\mathbf{x})$ defines a potential field influenced by neighbours.
Advantages: Obstacle avoidance, stability analysis.
Limitations: Computationally expensive; requires careful tuning.

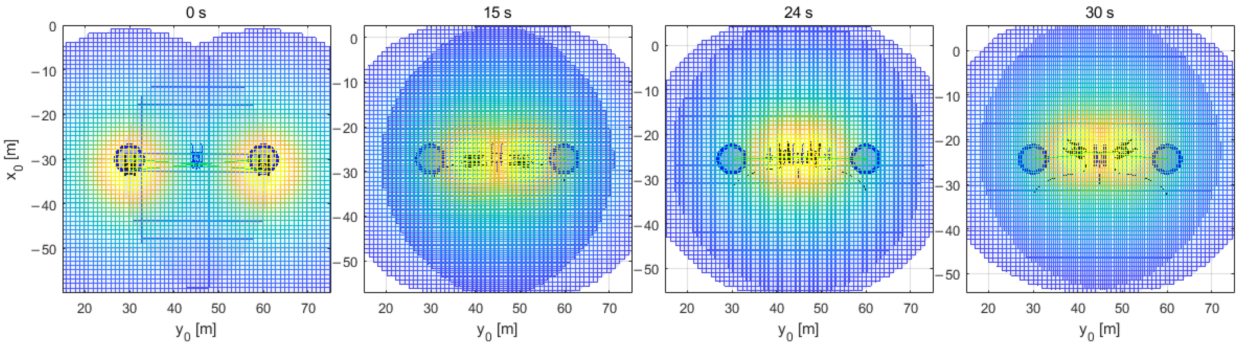


Fig. 2.3: Overlapping Potential fields around agents and obstacles [12].

Couzin's Model

Divides space into zones:

- **Repulsion Zone:** Avoid collisions.
- **Orientation Zone:** Align with neighbours.
- **Attraction Zone:** Maintain group cohesion.

Key Features: Adaptive neighbour ranges, biologically inspired.

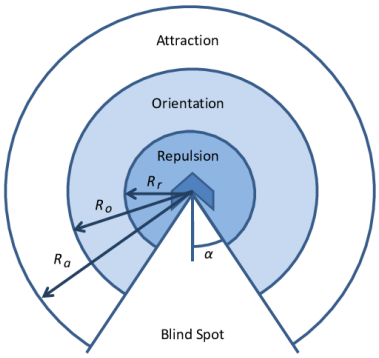


Fig. 2.4: Interaction zones in Couzin's model

Other Notable Approaches

Topological vs. Metric Interactions:

1. Topological: Fixed number of neighbours.
2. Metric: Fixed sensing radius.

- Topological interactions are generally more robust to density changes.

Leader-Follower Variants: Use designated leaders for task-oriented swarms.
Hybrid Physics-Based Models: Combine rule-based and physics-based logic.

2.1.3 Applications of Flocking Algorithms

1) Robotics:

UAV Swarm Coordination: Used in surveillance, search and rescue.
Search and Rescue Formations: Efficient area coverage while avoiding collisions.

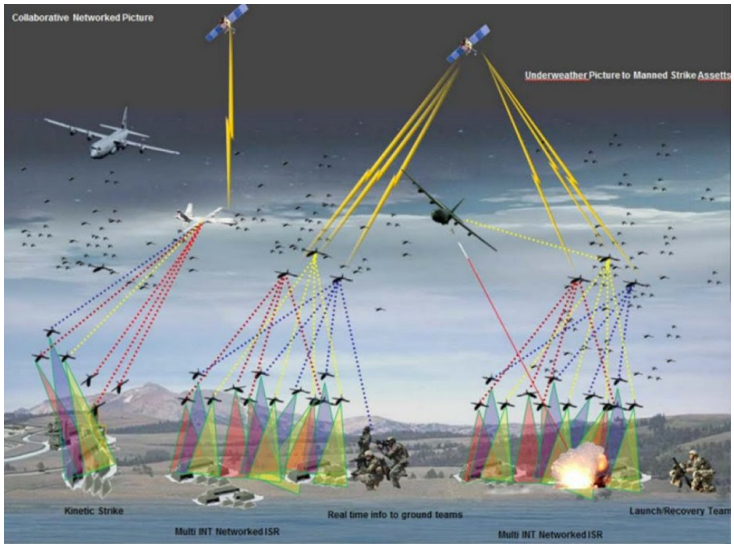


Fig. 2.5: UAV swarm in a search-and-rescue operation

2) Transportation Systems:

Autonomous Vehicle Platooning: Maintain safe distances and optimize flow.
Traffic Flow Optimization: Reduce congestion and improve safety.

3) Computer Graphics:

Crowd Simulation: Used in movies and games.
Virtual Reality: Enhance immersion with natural group dynamics.

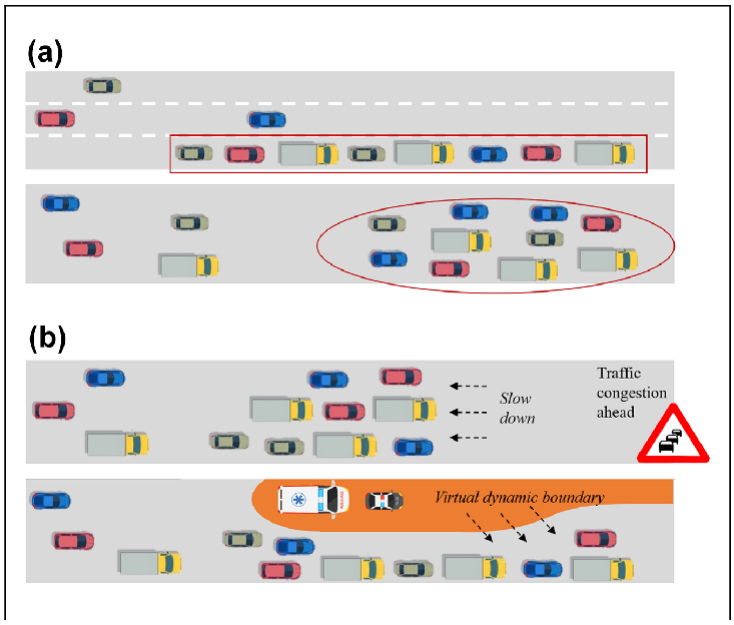


Fig. 2.6: Examples of vehicular flocking: (a) schematic representation of a vehicle platoon and a vehicle flock and (b) flock management use cases.

2.1.4 Challenges and Limitations

1) Scalability Issues:

Classical models have high computational cost in large swarms.

2) Environmental Sensitivity:

Struggle in dynamic environments with clutter and obstacles.

3) Communication Constraints:

Assume perfect detection, unrealistic in real-world systems.

4) Transition to MARL:

Traditional swarm control methods, often rule-based or optimization-driven, suffer from several limitations when deployed in real-world, dynamic environments:

1. **Poor Scalability**: Centralized or handcrafted strategies degrade in performance with increasing swarm size.

2. **Static Policies in Dynamic Environments**: Manually tuned behaviors fail to adapt to changing goals or threats.

3. **Brittleness to Agent Failures**: Most classical approaches assume agent homogeneity and reliability.
4. **Assumption of Full Observability**: They perform poorly when agents only have local or partial views.
5. **Sim-to-Real Transfer Issues**: Policies crafted in simulation often break under real-world noise and uncertainties.
6. **Unreliable in GPS-Denied Zones**: Classical methods often rely on precise localization.

These challenges motivate the adoption of Multi-Agent Reinforcement Learning (MARL), which offers:

- **Decentralized Learning of Adaptive Policies**
- **Robustness to Partial Observability and Agent Failures**
- **Scalability to Large and Heterogeneous Swarms**
- **Improved Generalization Across Simulated and Real Environments**

Tab. 2.1: Comparison of Classical Swarm Models vs. MARL-Based Approaches

Feature	**Classical Models**	**MARL-Based Approaches**
Scalability	Poor in large swarms	Naturally scalable
Adaptability	Manual tuning required	Learns from interaction
Observability	Assumes full observability	Handles partial/local views
Fault Tolerance	Fragile to agent loss	Resilient to agent failures
Policy Design	Rule-based, static	Data-driven, adaptive
Real-World Transfer	Poor (Sim-to-Real gap)	Better generalization
Localization Dependence	Requires GPS	Can operate without GPS

3. CHAPTER 3: DISCUSSIONS IN REINFORCEMENT LEARNING

3.1 Introduction

3.1.1 Brief history of RL

The core principle of reinforcement learning is its capacity to enable autonomous agents to learn from experience in the absence of explicit supervision. In contrast to supervised learning, in which models learn from labelled examples, RL agents adopt effective strategies through exploration and exploitation of their surroundings. They receive only reward signals as guidance, where those signals are determined by the desirability of outcomes. Initially developed to tackle issues in sequential decision-making, reinforcement learning has evolved from its origins in dynamic programming to become a central component of modern artificial intelligence research and applications. Over the past several decades, RL has gone through substantial transformations, from tabular methods to complex, deep neural network frameworks capable of tackling complex, real-world issues. It has found uses in several domains, including robotics, gaming, autonomous systems, and resource management, by allowing agents to acquire ideal behaviour through trial-and-error interactions with their environment.

3.1.2 Key Concepts

At the core of reinforcement learning are several fundamental concepts:

1. **Agents**: Entities that interact with an environment and make decisions to achieve specific goals. In the context of flocking, each individual in the swarm can be modelled as an agent.

2. **Environments**: The external system with which agents interact, characterized by states that change in response to agent actions. For flocking systems, the environment includes the physical space, obstacles, and other agents.

3. **Rewards**: Numerical feedback signals that guide learning by indicating the desirability of state transitions. In flocking, rewards might encourage cohesion, alignment, and collision avoidance.

4. **Policies**: Strategies that map environmental states to agent actions, representing the agent's behaviour. The goal of RL is typically to learn an optimal policy that maximizes cumulative rewards.

3.1.2 Connection to Flocking

Flocking behaviours, originally formalized by **Craig Reynolds' Boids Model** [13] through his three simple rules (alignment, cohesion, and separation), represent a fascinating case study for reinforcement learning applications. These collective behaviours require agents to adapt to dynamic environments and coordinate with others—precisely the type of complex decision-making problem that modern RL algorithms excel at solving.

Reinforcement learning provides a natural framework for achieving adaptive swarm behaviours by learning policies that optimize both individual and collective objectives. Unlike traditional rule-based approaches, RL-based flocking methods adapt to changing environments, learn from experience, and potentially discover emergent strategies that human designers might not anticipate [15].

The decentralized nature of flocking aligns well with multi-agent reinforcement learning (MARL) paradigms, where multiple agents learn simultaneously through individual or collective experiences. This connection makes RL particularly suitable for developing robust, adaptive flocking behaviours

in applications ranging from drone swarms to traffic management systems [6].

3.1.3 Brief purpose of this chapter

This chapter bridges core RL concepts with their application to multi-agent systems, laying the foundation for solving our swarm coordination problem. The following sections cover RL fundamentals, key algorithms, and their extension to MARL for adaptive, scalable swarm behaviours.

3.2 RL Basics

3.2.1 Markov Decision Processes (MDPs)

At the heart of reinforcement learning lies the **Markov Decision Process** (MDP), a mathematical framework for modelling decision-making in environments where outcomes are partly random and partly under the control of a decision-maker [11].

An MDP is defined by the tuple (S, A, P, R, γ), where:

- S: Set of possible states
- A: Set of available actions
- $P(s'|s, a)$: Transition probability function
- $R(s, a, s')$: Reward function
- $\gamma \in [0, 1]$: Discount factor

The **Markov property** states that the future depends only on the current state and action:

$$P(s_{t+1}|s_t, a_t, s_{t-1}, a_{t-1}, ..., s_0, a_0) = P(s_{t+1}|s_t, a_t) \tag{3.1}$$

Value Functions and Optimal Policies

1. **State Value Function**:

$$V^{\pi}(s) = \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s\right] \tag{3.2}$$

2. **Action Value Function**:

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a\right] \tag{3.3}$$

3. **Optimal Value Functions**:

$$V^*(s) = \max_{\pi} V^{\pi}(s) \quad \forall s \in S \tag{3.4}$$

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a) \quad \forall s \in S, a \in A \tag{3.5}$$

3.2.2 Core Algorithms

Q-Learning (Value-Based)

The Q-learning update rule is given by:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[R + \gamma \max_{a'} Q(s',a') - Q(s,a)\right] \tag{3.6}$$

where:

- $Q(s,a)$: Current estimate of the action-value function for state s and action a
- α: Learning rate ($0 < \alpha \leq 1$), which controls how much new information overrides the old value
- R: Immediate reward received after taking action a in state s
- γ: Discount factor ($0 \leq \gamma \leq 1$), which determines the importance of future rewards

- s': The next state reached after executing action a in state s
- $\max_{a'} Q(s', a')$: The estimated maximum future reward achievable from state s' over all possible actions a'

Strengths:

1. Easy to implement and guarantees convergence under certain normal conditions
2. Works well for discrete action spaces
3. Since the nature of Q-Learning Algorithm is off-policy, we have an improved sample efficiency; because it allows an agent to learn from data collected by a different policy than the one it is currently using.
   (This means the agent can leverage past experiences and reuse samples, rather than needing to repeatedly interact with the environment to gather new data for every policy update.)

Limitations:

1. Not suitable for problems where actions are continuous
2. May overestimate the value of actions, which can lead to suboptimal decisions
3. Needs a lot of memory to store Q-values for all state-action pairs in large environments

Policy Gradients (Policy-Based)

The policy gradient theorem provides a way to compute the gradient of the expected return with respect to the policy parameters. It is given by:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta} \left[\nabla_\theta \log \pi_\theta(a|s) \cdot Q^{\pi_\theta}(s, a) \right] \tag{3.7}$$

where:

- $\nabla_\theta J(\theta)$: Gradient of the expected return (performance objective) with respect to the policy parameters θ
- $\pi_\theta(a|s)$: The probability of taking action a in state s under the policy parametrised by θ
- $\nabla_\theta \log \pi_\theta(a|s)$: Gradient of the log-probability of the action under the current policy — directs the gradient in the direction that increases the likelihood of good actions
- $Q^{\pi_\theta}(s, a)$: Action-value function under policy π_θ, representing the expected return from taking action a in state s and following policy π_θ thereafter
- $\mathbb{E}_{\pi_\theta}[\cdot]$: Expectation over the distribution of state-action pairs induced by the current policy π_θ

Strengths:

1. Can work with actions that are continuous instead of just fixed choices
2. They can learn flexible (stochastic) policies; which allows for more exploration
3. Often lead to Stable convergence in many environments

Limitations:

1. The learning signal (gradient) can be noisy and unstable
2. Needs a lot of training data to learn well (i.e. sample-inefficient compared to off-policy methods)
3. Sensitive to right learning rate and step-size selection

Tab. 3.1: Comparison of Q-Learning and Policy Gradients

Aspect	Q-Learning	Policy Gradient
Approach	Value-based	Policy-based
Action Space	Discrete	Continuous
Sample Efficiency	High	Lower

3.2.3 Modern Algorithms

- **Proximal Policy Optimization (PPO):**

$$L^{CLIP}(\theta) = \mathbb{E}\left[\min\left(r_t(\theta)\hat{A}_t,\ \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\right)\right] \tag{3.8}$$

where:

  - $L^{CLIP}(\theta)$: The clipped surrogate objective function to be maximized
  - $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$: The ratio between the new and old policy probabilities
  - $\hat{A}_t$: The estimated advantage at time step t
  - ϵ: A small constant that defines how far the policy is allowed to change (e.g., 0.2)
  - $\text{clip}(r, 1-\epsilon, 1+\epsilon)$: Clips the probability ratio to prevent large updates

  This clipping mechanism helps ensure stable learning by limiting drastic policy changes.

- **Soft Actor-Critic (SAC):**

$$\pi^* = \arg\max_\pi \mathbb{E}\left[\sum_t r(s_t, a_t) + \alpha\mathcal{H}(\pi(\cdot|s_t))\right] \tag{3.9}$$

where:

  - π^*: The optimal policy being learned

- $r(s_t, a_t)$: The reward received at time step t
- α: Temperature parameter controlling the trade-off between reward and entropy
- $\mathcal{H}(\pi(\cdot|s_t))$: Entropy of the policy at state s_t, encouraging exploration by maximizing randomness

SAC promotes both reward maximization and policy entropy, leading to more exploratory and stable behaviour.

Tab. 3.2: Summary of RL Algorithms and Their Applications

Algorithm	**Type**	**Application**
Q-Learning	Value-Based	Discrete Control
PPO	On-Policy	Small-Scale Cooperation
SAC	Off-Policy	Large Continuous Control

3.3 Online vs. Offline RL

The exploration of SAC and PPO policies for swarming is motivated by the need to evaluate different reinforcement learning approaches for achieving robust and adaptive swarm behaviours.

SAC, with its focus on continuous action spaces and stability, is well-suited for fine-grained control in dynamic environments. On the other hand, PPO's simplicity and efficiency make it a strong candidate for scalable swarm coordination. By comparing these two policies, we aim to identify the strengths and trade-offs of each in the context of multi-agent swarming tasks..

Algorithm	**Type**	**Use Case**	**Reference**
PPO	On-Policy	Small-scale cooperation	Schulman et al. (2017)
SAC	Off-Policy	Large-scale flocking	Haarnoja et al. (2018)

Tab. 3.3: Reinforcement Learning Algorithms and Their Applications

Feature	On-Policy (e.g., PPO)	Off-Policy (e.g., SAC)
Data Usage	Current policy only	Any historical data
Sample Efficiency	Lower	Higher
Stability	More stable	Less stable
Exploration	Policy-dependent	Experience replay

Tab. 3.4: Comparison of on-policy and off-policy learning

3.4 Multi-Agent RL (MARL)

Robots can carry out more challenging missions when working together to achieve the same goal and exchange information. Such missions include object manipulation, self-guided navigation, and even tasks involving cooperation with other robots. To understand this, think about behaviours like flocking or swarming where several entities have to coordinate. One of the crucial developments in this area of robots is the movement from single-agent to multi-agent reinforcement learning (MARL). By moving from single-agent to multi-agent reinforcement learning, robots can improve the effectiveness of their teamwork to achieve their assigned missions..

3.4.1 MARL Paradigms

MARL algorithms can be classified based on their training and execution architectures:

Centralized vs. Decentralized Approaches

A fundamental design choice in MARL is the degree of centralization during training and execution [5]:

1. **Centralized Training and Execution:** All agents share information and coordinate through a central controller. While potentially optimal, this approach faces scalability issues and requires constant communication.

2. **Decentralized Training and Execution:** Agents learn and act independently based on local observations. This approach is scalable but

Aspect	Online Reinforcement Learning	Offline Reinforcement Learning
Core Characteristics	• Learns by interacting with the environment • Continuously adapts during training • Requires managing exploration-exploitation trade-off	• Learns from a fixed dataset • No further interaction with the environment • Must handle distributional shift from policy mismatch
Advantages	• Adapts to non-stationary environments • Enables continuous improvement of policy • Free from dataset bias	• Safer, no active exploration during training • More data-efficient with reuse of past experience • Suitable when real-world interaction is risky or expensive
Disadvantages	• Requires many interactions (sample inefficient) • Can be unsafe during learning • May suffer from training instability	• Effectiveness limited by dataset diversity • Prone to extrapolation errors • Often produces overly conservative policies

Tab. 3.5: Comparison of Online and Offline Reinforcement Learning

Approach	Sample Efficiency	Safety	Flexibility
Online RL	Low	Low	High
Offline RL	Medium	High	Low

Tab. 3.6: Comparison of RL paradigms

Aspect	Online RL	Offline RL
Data Source	Real-time interaction	Fixed dataset
Exploration	Active during learning	None during learning
Sample Efficiency	Lower (requires exploration)	Higher (reuses existing data)
Adaptability	High (continues to learn)	Low (fixed after training)
Safety	Potential risks during exploration	No exploration risks
Applications	Simulations, games, non-critical systems	Safety-critical systems, robotics, healthcare

Tab. 3.7: Key differences between Online and Offline Reinforcement Learning (RL) approaches.

may yield suboptimal coordination.

3. **Centralized Training with Decentralized Execution (CTDE):** Combines the benefits of both approaches by leveraging global information during training while requiring only local observations during execution. This paradigm has become increasingly popular for cooperative MARL tasks, including flocking behaviours.

The CTDE approach, used in algorithms like SAC and PPO, offers a flexible way to create coordinated flocking behaviours. It combines centralized training to improve policies with decentralized execution for better scalability. This allows agents to adjust to changing environments while still maintaining their individual autonomy.

3.4.2 Advantages and Disadvantages of CTDE

The CTDE paradigm has emerged as a compelling middle ground, particularly for cooperative tasks like flocking [8].

Tab. 3.8: Advantages and Disadvantages

Advantages	**Disadvantages**
Leverages global information during training for improved coordination	Requires careful design to ensure policies trained with global information perform well with only local observations
Maintains decentralized execution for scalability and robustness	May still face challenges in very large-scale systems
Mitigates non-stationarity issues during training	Implementation complexity
Facilitates credit assignment through centralized critics	

The CTDE approach has been implemented in various MARL algorithms relevant to flocking behaviours (MADDPG [9], QMIX [**?**], MAPPO [**?**]).

Tab. 3.9: Comparison of MARL Training Paradigms

Training Paradigm	**Scalability**	**Comm. Overhead**	**Coordn Quality**	**Impltn Compl.**
Centralized	Low	High	High	Low
Decentralized	High	None	Lower	Medium
CTDE	Medium-High	None (during execution)	Medium-High	High

In flocking applications, where scalability and decentralized execution are essential but coordination quality matters, the CTDE paradigm offers a promising approach that balances these competing objectives. A flock of birds, for example, needs to be large and efficient to cover its territory, but it also needs to remain coordinated to avoid collisions and stay together for better defence and for foraging. The CTDE paradigm is also useful in other applications that involve large numbers of agents or components, such as distributed computing or robotics.

Tab. 3.10: Comparison of MARL Training Paradigms

Paradigm	**Pros**	**Cons**
Centralized	Coordinated learning	Not scalable
Decentralized	Scalable	Less coordination
CTDE	Best of both worlds	Implementation complexity

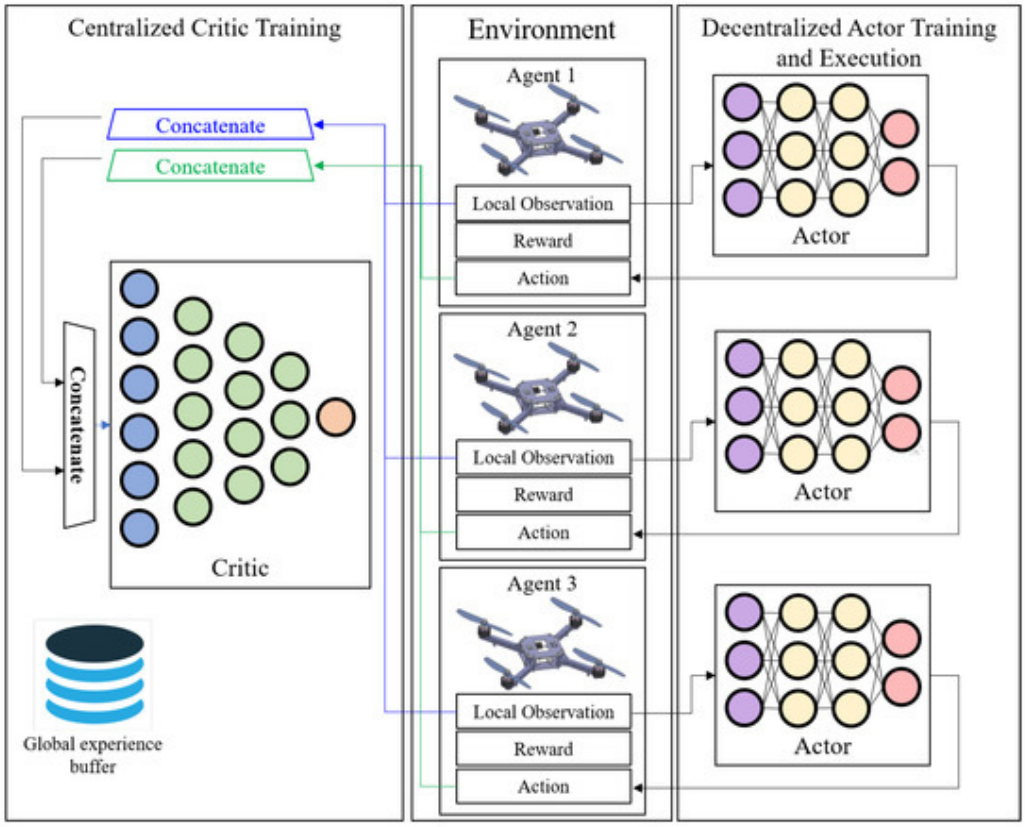


Fig. 3.1: CTDE visualisation. cited from: [2]

4. CHAPTER 4 : THEORETICAL FRAMEWORK

4.1 Formulating Flocking as a MARL Problem

4.1.1 POMG Problem Formulation

What is a POMG (Partially Observable Markov Game)?

A Partially Observable Markov Game is an extension of a Markov Decision Process (MDP) and Partially Observable MDP (POMDP) to the multi-agent setting. It is used to mathematically model environments where multiple agents interact, each with limited information about the full state of the world.

Flocking in a multi-agent system can be modelled as a POMG, which extends the Markov Decision Process (MDP) to multi-agent settings with partial observability.

Definition of POMG: A POMG is defined by the tuple:

$$\langle \mathcal{N}, \mathcal{S}, \{\mathcal{A}_i\}_{i\in\mathcal{N}}, \{\mathcal{O}_i\}_{i\in\mathcal{N}}, \mathcal{P}, \{R_i\}_{i\in\mathcal{N}}, \gamma \rangle$$

- $\mathcal{N}$: Set of agents.
- $\mathcal{S}$: Global state space.
- $\mathcal{A}_i$: Action space of agent i.
- $\mathcal{O}_i$: Observation space of agent i.

- $\mathcal{P}$: State transition probability function.
- R_i: Reward function for agent i.
- γ: Discount factor.

Key Features:

- *Partial Observability*: Each agent observes only a subset of the global state.
- *Decentralized Execution*: Agents act based on local observations.
- *Shared Environment*: Agents interact with each other and the environment.

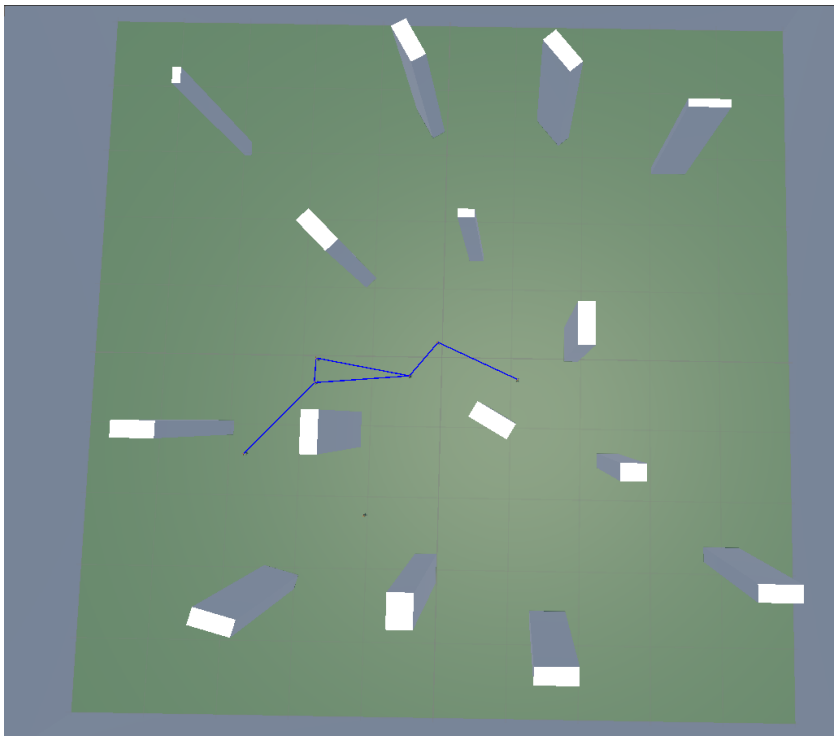


Fig. 4.1: Illustration of the physical environment for the POMG framework showing agents, local observations, and shared environment.

4.1.2 State Space Vector Representation

The state space captures the information required for decision-making by each agent.

Agent States:

Each agent's state is represented as:

$$s_i = [\mathbf{v}_i, \mathbf{d}_{\text{obstacles}}, \mathbf{d}_{\text{drones}}, \text{swarm\_info}]$$

Where:

- $\mathbf{v}_i$: Local velocity of the agent in its coordinate frame (v_x, v_z).
- $\mathbf{d}_{\text{obstacles}}$: Distances to obstacles detected via raycasting in N directions.
- $\mathbf{d}_{\text{drones}}$: Distances to other drones detected via raycasting in N directions.
- swarm_info: Swarm-related information, including:
  - Number of drones sensed in the current step (currentSensed).
  - Total number of drones sensed so far (totalSensed).
  - Size of the swarm the agent belongs to (swarmDrones.Count).

Complete State Space:

The complete state space for all agents is:

$$S = \{s_1, s_2, \ldots, s_N\}$$

Where N is the total number of drones.

Global State vs. Individual Agent States:

- **Global State:** Includes the positions and velocities of all drones, as well as obstacle positions.
- **Individual Agent States:** Each agent only observes its local state due to partial observability.

Mathematical Representation:

For agent i:

$$s_i = [v_x, v_z, d^1_{\text{obstacles}}, \ldots, d^N_{\text{obstacles}}, d^1_{\text{drones}}, \ldots, d^N_{\text{drones}}, \text{currentSensed}, \text{totalSensed}, \text{swarmSize}]$$

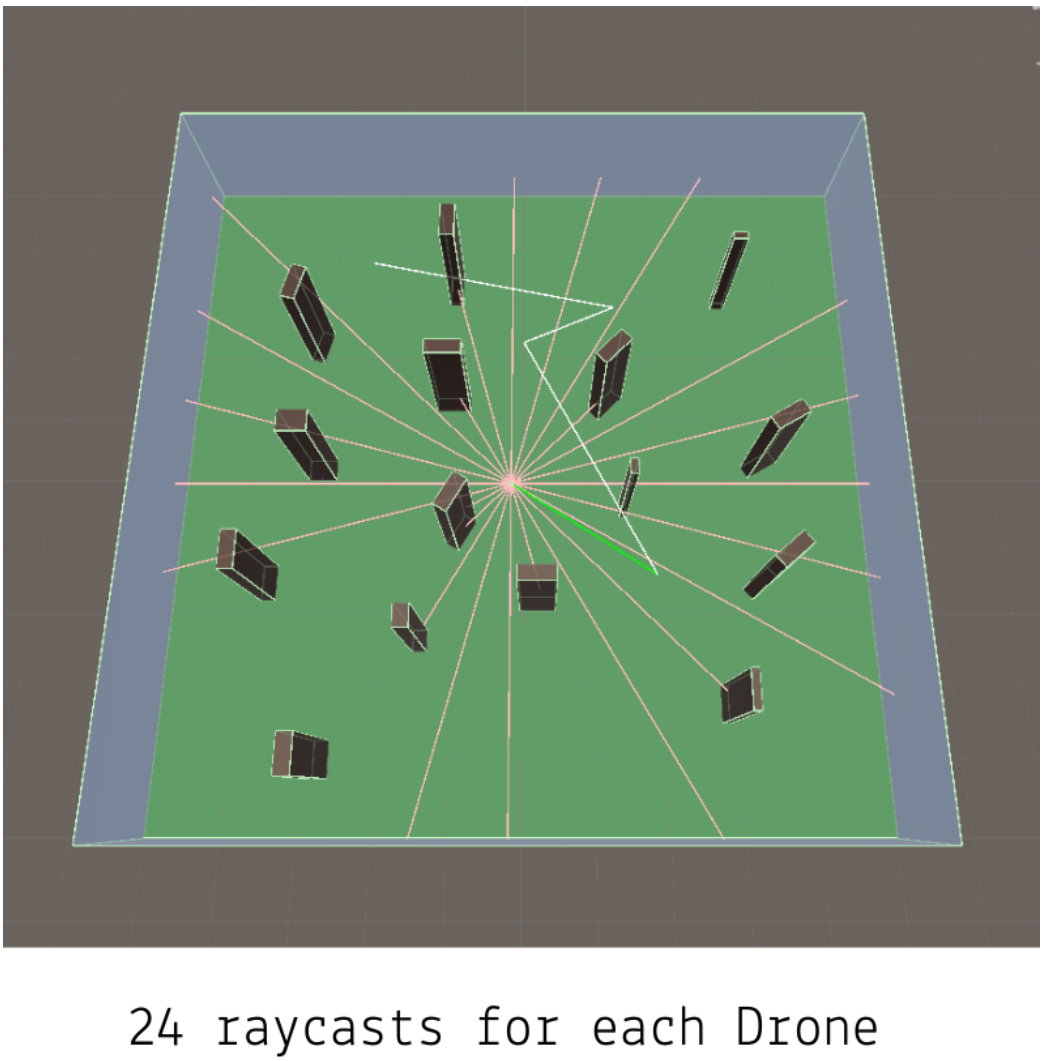


Fig. 4.2: Diagram showing the local observation range of an agent and its neighbors.

4.1.3 Action Space Representation

The action space defines the set of possible actions an agent can take.

Continuous Action Space:

Each agent takes continuous actions to control its movement in the X and Z directions:

$$a_i = [\Delta v_x, \Delta v_z]$$

Where:

- Δv_x: Change in velocity along the X-axis.
- Δv_z: Change in velocity along the Z-axis.

The actions are bounded within $[-1, 1]$ and scaled by the maximum force (Drone_Values.MaxForce).

Action Constraints:

The velocity is clamped to the maximum speed (Drone_Values.MaxSpeed):

$$\mathbf{v}_i = \text{Clamp}(\mathbf{v}_i, -\text{MaxSpeed}, \text{MaxSpeed})$$

Mathematical Representation:

For agent i:

$$a_i = [\Delta v_x, \Delta v_z]$$

Where:

$$\Delta v_x, \Delta v_z \in [-1, 1]$$

Note: We use only Continuous Action Space in our model. Discrete Action Space is not used in our model.

4.2 Reward Function Design

4.2.1 Reward Components

The total reward for agent i is a weighted combination of proximity, cohesion, separation, obstacle avoidance, and task-specific rewards:

$$R_i = w_1 R_{\text{prox}} + w_2 R_{\text{coh}} + w_3 R_{\text{sep}} + w_4 R_{\text{obs}} + w_5 R_{\text{task}}$$

1) Proximity Reward (R_{prox}):
Encourages agents to maintain an optimal distance from their neighbors:

$$R_{\text{prox}} = \sum_{j \in \mathcal{N}_i} \text{gradient\_reward}(d_{ij})$$

Where d_{ij} is the distance between agent i and neighbor j, and the reward is scaled based on proximity to the optimal distance range (R_{in} to R_{out}).

2) Cohesion Reward (R_{coh}):
Encourages agents to move toward the center of their neighbors:

$$R_{\text{coh}} = - \|\bar{p}_{\mathcal{N}_i} - p_i\|$$

Where $\bar{p}_{\mathcal{N}_i}$ is the average position of neighbors.

3) Separation Penalty (R_{sep}):
Penalizes agents for being too close to their neighbors:

$$R_{\text{sep}} = \sum_{j \in \mathcal{N}_i} \max\left(0, d_{\min} - \|p_i - p_j\|\right)$$

4) Obstacle Avoidance Penalty (R_{obs}):

Penalizes agents for being too close to obstacles:

$$R_{\text{obs}} = -\sum_{k \in \mathcal{O}} \max\left(0, d_{\text{obs}} - \|p_i - o_k\|\right)$$

5) Task-Specific Rewards (R_{task}):
Includes rewards for survival, swarm formation, and staying in optimal zones.

Component	Description
R_{prox}	Reward for maintaining optimal distance from neighbors
R_{coh}	Reward for moving toward the center of neighbors
R_{sep}	Penalty for being too close to neighbors
R_{obs}	Penalty for being too close to obstacles
R_{task}	Rewards for survival, swarm formation, and staying in optimal zones

Tab. 4.1: Summary of reward components and their descriptions.

4.2.2 Weighted Combination of Rewards

The total reward is a weighted combination of the above components:

$$R_i = w_1 R_{\text{prox}} + w_2 R_{\text{coh}} + w_3 R_{\text{sep}} + w_4 R_{\text{obs}} + w_5 R_{\text{task}}$$

where w_1, w_2, w_3, w_4, w_5 are scalar weights used to balance the contribution of each reward component.

Note: *In our implementation, we set all weights to unity:*

$$w_1 = w_2 = w_3 = w_4 = w_5 = 1$$

Tab. 4.2: Reward and Penalty Components in Drone Swarm Simulation

Category	**Reward/Penalty Name**	**Description**	**Symbol**
Proximity & Collision Rewards	Obstacle Collision Penalty	Penalizes drones for colliding with obstacles.	R_{ocp}
	Intra-Swarm Collision Penalty	Penalizes collisions between drones in the same swarm to maintain swarm integrity.	R_{iscp}
	Boundary Collision Penalty	Penalizes drones for colliding with the environment boundaries.	$R_{boundary}$
Spatial Rewards (Region-Based)	Too Close Penalty	Penalizes drones for getting too close to others or obstacles.	$R_{tooclose}$
	Good Region Reward	Rewards drones for staying within optimal zones.	R_{good}
	Bad Region Penalty	Penalizes drones for being in undesirable areas.	R_{bad}
Exploration & Formation	Swarmation Reward	Rewards drones for maintaining swarm formation.	R_{swarm}
	Rebound Reward	Cascading reward shared across the swarm to promote exploration and cohesion.	$R_{rebound}$

4.3 Observation Models

4.3.1 Partial Observability

The observation model is based on partial observability and uses raycasting for sensing the environment within a sensing radius R_{sense}:

1. Relative distances of neighbouring agents encounter via agent's raycasts within R_{sense}.
2. Relative distances from obstacles and boundaries encounter via agent's raycasts within R_{sense}.

4.3.2 Sensor Model

The agent uses raycasting to detect obstacles and drones within the sensing radius R_{sense}. For each ray k, the distance d_k is calculated as:

$$d_k = \begin{cases} |\mathbf{p}_i - \mathbf{p}_{\text{hit}}|, & \text{if a hit is detected within } R_{\text{sense}} \\ R_{\text{sense}}, & \text{if no hit is detected} \end{cases}$$

Where:

- $\mathbf{p}_i$: Position of the agent.
- $\mathbf{p}_{\text{hit}}$: Position of the hit object (obstacle or drone).
- R_{sense}: Maximum sensing radius.

4.3.3 Noise

Not Implemented. The observed distances d_k are directly taken from raycasting without any perturbation.

4.3.4 Field-of-View

The agent has a 360° field-of-view, divided into N ray directions. (N = 24, here.)

4.3.5 Communication

In our model, explicit inter-agent communication is not enabled. Formally, for each agent i, there is no access to the internal states or actions of any other agent $j \neq i$ beyond what can be inferred through local observations. That is:

$$\text{Comm}_{i \leftarrow j} = \varnothing, \quad \forall i \neq j$$

Agents rely solely on their own observations and partial sensing of the environment (e.g., via raycasts) to make decisions.

4.4 Proposed Architecture

4.4.1 CTDE Framework

We adopt a Centralized Training with Decentralized Execution (CTDE) framework:

- **Centralized Critic:** Accesses global state during training.
- **Decentralized Actors:** Use local observations during execution.

4.4.2 Neural Networks

Actor Network

The **Actor Network** represents the policy $\pi(a|s)$, which maps the observed state $s \in \mathbb{R}^n$ to a continuous action $a \in \mathbb{R}^2$ corresponding to the movement in the X and Z directions:

$$a = \pi_\theta(s) = [\Delta v_x, \Delta v_z] \in [-1, 1]^2$$

Architecture:

- **Type:** 2-layer Multi-Layer Perceptron (MLP)
- **Hidden Units:** 128 units per layer

- **Activation Function:** ReLU
- **Normalization:** Observations s are normalized before input

Mathematically, the Actor network can be expressed as:

$$\pi_\theta(s) = \tanh\left(W_2 \cdot \sigma(W_1 \cdot \hat{s} + b_1) + b_2\right)$$

where:

- $\hat{s}$: Normalized observation vector
- W_1, W_2: Weight matrices of the first and second layers
- b_1, b_2: Bias vectors
- σ: ReLU activation function, $\sigma(x) = \max(0, x)$
- tanh: Squashing function to ensure output actions are in $[-1, 1]$

Critic Network

The **Critic Network** estimates the value function $V(s)$ or the action-value function $Q(s, a)$, which represents the expected return from a given state (or state-action pair).

Architecture:

- **Type:** 2-layer MLP
- **Hidden Units:** 128 units per layer
- **Activation Function:** ReLU
- **Input:** (s) or (s, a) depending on whether value or action-value function is used

Mathematically, if estimating $Q(s, a)$:

$$Q_\phi(s, a) = W_2' \cdot \sigma(W_1' \cdot [s \| a] + b_1') + b_2'$$

where $[s\|a]$ denotes the concatenation of state and action vectors.

Temporal Dependencies:

- While no explicit recurrent structure (e.g., LSTM) is used, the network is trained with a `time_horizon` of 256 steps, allowing it to learn long-term reward dependencies.

Summary

Tab. 4.3: Summary of Actor-Critic Network Design

Network	Architecture	Purpose
Actor Network	2-layer MLP, 128 units per layer, ReLU activation	Generates actions based on observations to maximize expected cumulative reward ($\pi(a\|s)$).
Critic Network	2-layer MLP, 128 units per layer, ReLU activation	Evaluates the quality of actions using $V(s)$ or $Q(s,a)$, trained over sequences of length 256.

4.4.3 Training Pipeline

1) Soft Actor-Critic (SAC) Training Pipeline:

The SAC algorithm optimizes a stochastic policy by maximizing expected reward and entropy to encourage exploration.

Step 1: Sampling

From N agents interacting with the environment, we collect a set of transitions:

$$\mathcal{D} = \left\{(s_t^i, a_t^i, r_t^i, s_{t+1}^i)\right\}_{t=0}^{T} \quad \text{for } i = 1, \ldots, N$$

Step 2: Critic Update

The Critic is updated using TD(λ):

$$y_t = r_t + \gamma \left[(1-\lambda)V(s_{t+1}) + \lambda y_{t+1}\right]$$

$$\mathcal{L}_{\text{critic}} = \mathbb{E}_{(s_t,a_t,y_t)\sim\mathcal{D}}\left[(Q(s_t,a_t) - y_t)^2\right]$$

Step 3: Actor Update (SAC)

The Actor maximizes the entropy-regularized objective:

$$J_\pi(\theta) = \mathbb{E}_{s\sim\mathcal{D},a\sim\pi_\theta}\left[\alpha\mathcal{H}[\pi_\theta(\cdot|s)] + Q(s,a)\right]$$

$$\text{where } \mathcal{H}[\pi(\cdot|s)] = -\mathbb{E}_{a\sim\pi}[\log\pi(a|s)]$$

Step 4: Training Configuration

Training hyperparameters:

- Batch size: `batch_size`
- Replay buffer size: `buffer_size`
- Update frequency: `steps_per_update`

Tab. 4.4: SAC Training Pipeline

Step	Description	Implementation
Sampling	Collect rollouts from N agents	`CollectObservations`, `OnActionReceived`, `AddNetReward`
Critic Update	TD(λ), $\gamma = 0.99$	`reward_signals`, `time_horizon`
Policy Update	Maximize entropy + Q-value	`network_settings`, `learning_rate`
Training Config	Buffer, batch, steps per update	`batch_size`, `buffer_size`, `steps_per_update`

2) Proximal Policy Optimization (PPO) Training Pipeline:
PPO uses a clipped surrogate objective to constrain policy updates.

Step 1: Sampling

Rollouts from N agents are collected sequentially:

$$\mathcal{D} = \left\{(s_t^i, a_t^i, r_t^i, s_{t+1}^i)\right\}_{t=0}^{T}$$

Step 2: Critic Update

Using TD(λ) with discount γ:

$$y_t = r_t + \gamma\left[(1-\lambda)V(s_{t+1}) + \lambda y_{t+1}\right]$$

$$\mathcal{L}_{\text{value}} = \left(V(s_t) - y_t\right)^2$$

Step 3: Actor Update (PPO)

The policy is optimized using a clipped objective:

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$$

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

Step 4: Training Configuration

- Learning rate schedule: `learning_rate_schedule`
- Clipping range: ϵ
- Epochs per update: `num_epoch`

Tab. 4.5: PPO Training Pipeline

Step	Description	Implementation
Sampling	Collect rollouts from N agents	`CollectObservations`, `OnActionReceived`, `AddNetReward`
Critic Update	TD(λ), $\gamma = 0.99$	`reward_signals`, `time_horizon`
Policy Update	PPO with KL clipping	`batch_size`, `epsilon`, `num_epoch`
Training Config	LR and clipping schedules	`learning_rate_schedule`, `network_settings`

4.4.4 Key Differences Between PPO and SAC

Tab. 4.6: Comparison: PPO vs SAC

Aspect	**PPO**	**SAC**
Policy Type	Stochastic policy with clipped objective	Entropy-regularized stochastic policy
Critic	Single value function $V(s)$	Q-function $Q(s, a)$ over state-action pairs
Sampling	On-policy; collects rollouts and discards them after each update	Off-policy; uses a replay buffer to sample past transitions
Exploration	Implicitly controlled via clipping	Encouraged explicitly using entropy bonus

5. CHAPTER 5: SYSTEM FRAMEWORK AND METHODOLOGY

5.1 MARL Set-Up as POMG

5.1.1 MARL Framework Overview

In the last chapter, we formulated the flocking problem within a Multi-Agent Reinforcement Learning (MARL) framework, where each drone operates as an independent agent making decisions based solely on local observations. The fundamental challenge addressed is coordinating multiple autonomous drones to achieve cohesive flocking behaviour under partial observability constraints, particularly in environments where global position information may be unavailable or unreliable.

The problem was mathematically modelled as a Partially Observable Markov Game (POMG), which extended the concept of Partially Observable Markov Decision Processes (POMDPs) to multi-agent settings. Within this framework, each agent:

1. Makes decisions independently
2. Acts based only on its own local observations
3. Receives individual rewards while contributing to collective behavior
4. Must adapt to both environmental challenges and the actions of other agents

This decentralized decision-making approach is critical for scalable and robust flocking systems, as it eliminates single points of failure and reduces communication overhead commonly found in centralized control architectures.

5.1.2 Agent Architecture

Our system consists of two types of agents:

1. **Common Drones**: These are learning agents that make up the bulk of the swarm. Each common drone:
   (a) Operates independently using its own policy network
   (b) Perceives only local information through a limited sensing range
   (c) Makes decisions based on a partial view of the environment
   (d) Learns through reinforcement to maintain optimal positioning within the swarm

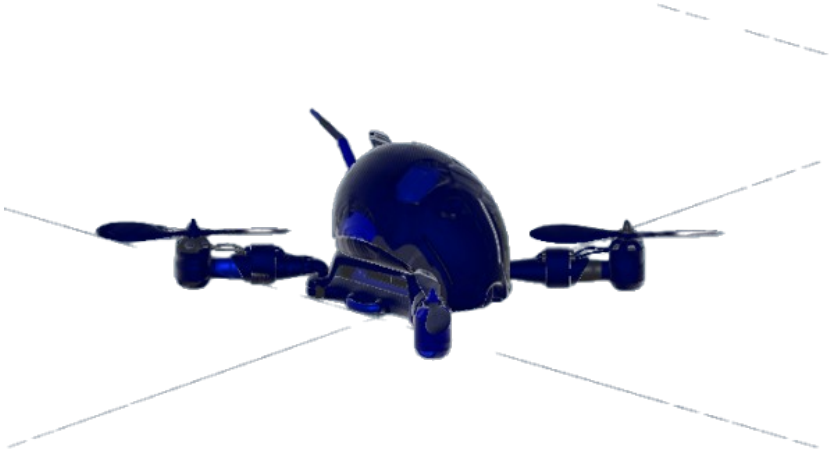


Fig. 5.1: Common Drone Image

2. **Leader Drone**: A non-learning agent that serves as a navigational guide:

   (a) Generates waypoints and follows a predetermined or user-controlled path

   (b) Uses heuristic control rather than learned policies

   (c) Has natural obstacle avoidance capabilities

   (d) Guides the swarm during training without explicit communication

Fig. 5.2: Leader Drone Image

This division creates a semi-supervised swarm architecture where common drones learn to follow and maintain formation around the leader while simultaneously adapting to environmental constraints.

5.1.3 State Space Definition

Each drone constructs its state space from local observations, resulting in a 54-dimensional vector that captures motion, environmental context, and swarm information. The state space components include:

Tab. 5.1: State Space Components

Aspect	**Details**	**Dimension**
Local Velocity	The drone's velocity in its local coordinate frame (X and Z components)	2
Obstacle Distances	Distances to obstacles detected via raycasting in 24 directions	24
Drone Distances	Distances to other drones detected via raycasting in 24 directions	24
Swarm Information	Previous sensed drones, current sensed drones, total sensed drones, current swarm size	4
Total State Space		**54**

For raycast observations, if no obstacle or drone is detected in a particular direction, the maximum sensing range (R_{sense}) is used as the distance value. This ensures a consistent state vector dimension regardless of the environment configuration or swarm density.

The raycast-based perception system mimics the limited field of view of real-world drones and enforces the partial observability constraint. Each drone must make decisions without access to global state information, similar to how birds in natural flocks rely only on what they can directly perceive.

5.1.4 Action Space Definition

The action space is defined as continuous control in the X-Z plane (horizontal plane), with the Y-axis (altitude) fixed. This two-dimensional continuous action space is represented as:

Action Vector: [action[0], action[1]] $\in$ [-1, 1]^2

- action[0]: Left/Right movement (X-axis)
- action[1]: Forward/Backward movement (Z-axis)

For example, an action of [0.5, -0.8] would indicate a diagonal movement in the right-backward direction.

These continuous action values are scaled by a maximum force parameter (`Drone_Values.MaxForce`) and applied directly to the drone's rigid body physics. To maintain realistic behavior, the resulting velocity is clamped to a maximum speed value (`Drone_Values.MaxSpeed`).

5.2 Reward Function Design

Reward Function Overview

Our RL system includes reward functions to not only encourage flocking behavior, but also to reward optimal system function as a whole. To put it another way, the rewards from each behavior are based on the behavior and its usefulness. This ensures that both the overall system and each individual agent receive the benefit of rewards.

$$R_{total} = R_{swarm} + R_{proximity} + R_{survival} + R_{obstacle} + R_{boundary} + R_{disintegration} + R_{stagnation} + R_{collision} \tag{5.1}$$

Where each component addresses specific aspects of the flocking objective:

5.2.1 Positive Reward Components

Swarm Formation Reward (R_{swarm})

This reward encourages drones to form and maintain a cohesive swarm:

$$R_{swarm} = swarmationReward + (swarmationReward \times rr\_factor \times (swarmDrones.Count - 1)) \tag{5.2}$$

Where:

- `swarmationReward` = 200.0 (base reward for maintaining swarm contact)
- `rr_factor` is a scaling parameter for the rebound reward
- `swarmDrones.Count` is the number of drones currently in the swarm

The second term is the novel Intra-Swarm Rebound Reward, which provides cascading rewards across all swarm members when any drone encounters another drone. This mechanism promotes both swarm integrity and

exploration, while enabling natural leadership transitions if the leader drone is compromised.

Proximity Reward ($R_{proximity}$)

This reward incentivizes drones to maintain optimal distances from each other:

$$R_{proximity} = \sum(gradient\_reward \times scaling\_factor) + insideGoodRegionReward + insideSensing$$ (5.3)

Where:

- `gradient_reward` is calculated based on the distance to other drones within sensing range
- `insideGoodRegionReward` = 80.0 (reward for being in the optimal distance range R_{in} to R_{out})
- `insideSensingZoneReward` = 20.0 (reward for being within the sensing range R_{out} to R_{sense})

Survival Reward ($R_{survival}$)

A basic reward for remaining active:

$$R_{survival} = survivalReward = 20.0$$ (5.4)

5.2.2 Negative Reward Components (Penalties)

Obstacle Proximity Penalty ($R_{obstacle}$)

Penalizes drones for approaching obstacles too closely:

$$R_{obstacle} = \sum(gradient\_penalty) + obstacleCollisionPenalty$$ (5.5)

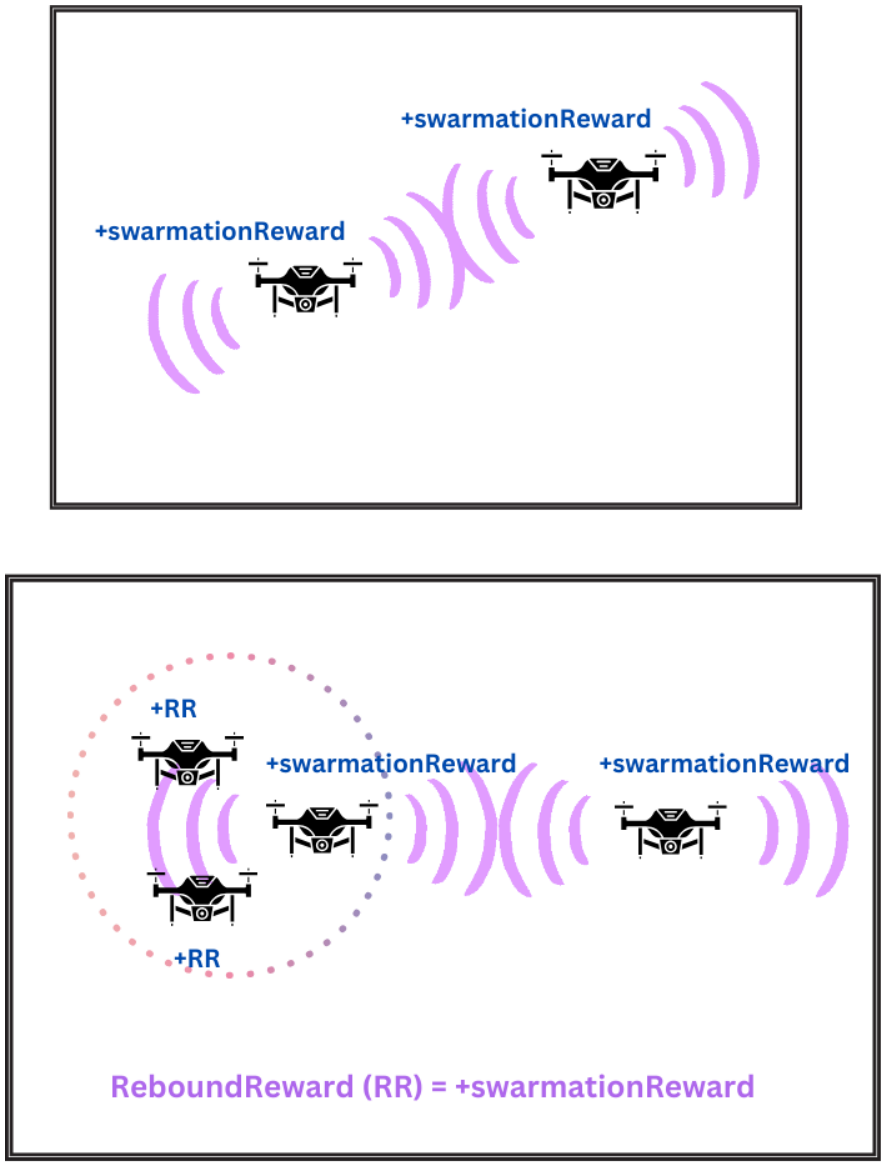


Fig. 5.3: Swarmation and Rebound Reward Visualisation

Where:

- `gradient_penalty` increases as the drone approaches obstacles within $1.5 \times R_{out}$
- `obstacleCollisionPenalty` = -100.0 (fixed penalty for colliding with obstacles)

Boundary Penalty ($R_{boundary}$)

Discourages drones from leaving the defined operational area:

$$R_{boundary} = boundaryCollisionPenalty = -100.0 \tag{5.6}$$

Disintegration Penalty ($R_{disintegration}$)

Heavily penalizes the loss of swarm members:

$$R_{disintegration} = (prevSensed - currentSensed) \times disintegrationPenalty \tag{5.7}$$

Where:

- `prevSensed` is the number of drones sensed in the previous time step
- `currentSensed` is the number of drones currently sensed
- `disintegrationPenalty` = -750.0

Collision Penalty ($R_{collision}$)

Discourages collisions with other drones:

$$R_{collision} = intraSwarmCollisionPenalty + tooClosePenalty \tag{5.8}$$

Where:

- `intraSwarmCollisionPenalty` = -180.0 (penalty for colliding with other drones)
- `tooClosePenalty` = -50.0 (penalty for being within $R_{tooclose}$ of another drone)

Stagnation Penalty

A small penalty to discourage inactivity:

$$stagnationPenalty = -5.0 \tag{5.9}$$

5.2.3 Spatial Zones and Distance Thresholds

The reward function utilizes a series of concentric zones around each drone to define optimal positioning:

1. **Too Close Zone** (0 to $R_{tooclose}$): Being in this zone triggers the `tooClosePenalty`
2. **Bad Zone** ($R_{tooclose}$ to R_{in}): Being in this zone triggers the `insideBadRegionPenalty` (-30.0)
3. **Good Zone** (R_{in} to R_{out}): Being in this zone provides the `insideGoodRegionReward`
4. **Sensing Zone** (R_{out} to R_{sense}): Being in this zone provides the `insideSensingZoneReward`

These zones create gradients that guide the drones toward maintaining optimal inter-agent distances while avoiding both excessive crowding and dispersion.

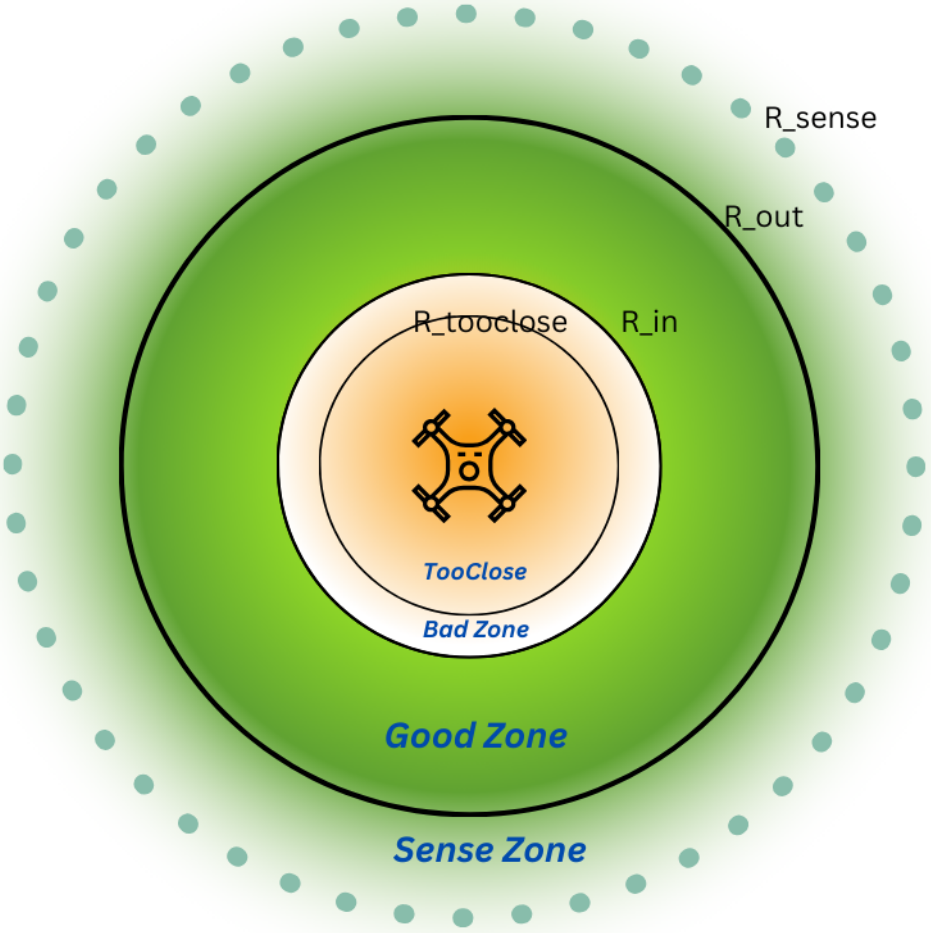


Fig. 5.4: Visualization of spatial zones around each drone

5.3 Simulation Environment

5.3.1 Physical Setup

The simulation environment was designed to provide realistic drone dynamics while enabling efficient parallel training. Key components include:

1. **Multiple Training Areas**: The environment contains multiple separate training areas, which allows for faster training through parallel episodes and better convergence.

2. **Randomized Obstacles**: Each training area includes procedurally generated obstacles that vary between episodes, ensuring that learned policies generalize across different environmental configurations.

3. **Randomized Drone Count:** The number of drones in each training environment is randomly varied, encouraging the policy to adapt to different swarm sizes and improving generalization.

4. **Boundary Constraints**: Clear boundaries define the operational area for each swarm, with penalties for boundary violations.

5. **Physical Architecture**: Each drone is represented as a physical entity with:

    - Realistic physics-based movement constraints
    - Raycast-based perception (24 rays per drone)
    - Collision detection with appropriate mass and momentum properties

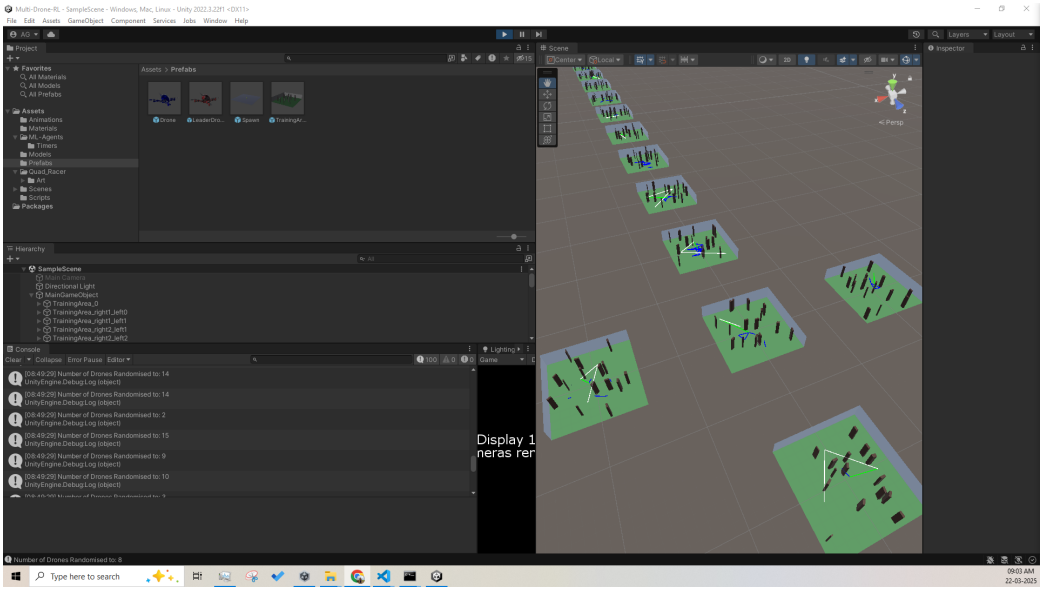


Fig. 5.5: Simulation Environment Set-up in Unity with training parameters set-up

5.3.2 Sensing Mechanism

The sensing mechanism implements the partial observability constraint through:

1. **Raycast Detection**: Each drone casts 24 rays evenly distributed in a circular pattern to detect:
    - Obstacles and boundaries
    - Other drones within sensing range
2. **Detection Zones**: Concentric zones around each drone define:
    - $R_{tooclose}$: Minimum safe distance between drones
    - R_{in}: Inner radius of the optimal zone
    - R_{out}: Outer radius of the optimal zone
    - R_{sense}: Maximum sensing range

This setup ensures that each agent has access only to local information, maintaining the partial observability requirement of the POMG formulation.

5.4 Learning Algorithms

5.4.1 Algorithm Selection

Two state-of-the-art MARL algorithms were implemented and compared:

1. **Proximal Policy Optimization (PPO)**:
   - On-policy algorithm with clipped objective function
   - Stable learning behaviour with bounded policy updates
   - Effective for environments with continuous action spaces
   - Hyperparameters were tuned to balance exploration and exploitation
2. **Soft Actor-Critic (SAC)**:
   - Off-policy algorithm with entropy maximization
   - Encourages exploration through entropy regularization
   - Sample-efficient due to experience replay
   - Well-suited for continuous control tasks

The implementation of both algorithms enabled a comparative analysis of their effectiveness in solving the flocking problem.

5.4.2 Curriculum Learning

A curriculum learning approach with the PPO algorithm was developed to accelerate training and improve policy robustness:

1. **Progressive Difficulty**: Training proceeded through phases of increasing difficulty:
   - Phase 1: Stationary leader drone with minimal obstacles [Refer here for implementation plot results: **??**]
   - Phase 2: Mobile leader drone with moderate obstacles

- Phase 3: Complex environments with dynamic obstacles and leader behaviour *(This phase is yet to be implemented)*

2. **Transfer Learning**: Policies learned in simpler environments were used as initialization for more complex scenarios, allowing for efficient knowledge transfer.

3. **Progressive Swarm Complexity in terms of Drone Count (Swarm Size) in the Environment**: Training proceeded gradually through phases of increasing number of drones in each environment starting from 2 drones, and rising up to 12+ drones.

The curriculum learning approach was compared against direct learning (training immediately on the full task complexity) to evaluate its effectiveness in improving convergence speed and final policy quality.

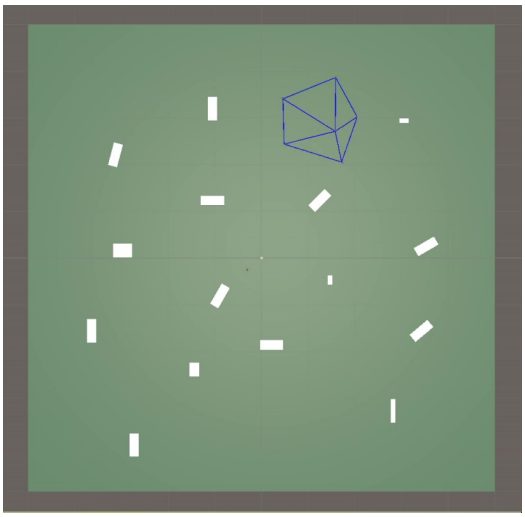


Fig. 5.6: Glimpse from the Trainings of Phase 1 of Curriculum Learning in the **Initial stages** (pic. 1) *(work covered until Mid-Year for Thesis)*
Training with knowledge of self's location in terms of global coordinates:
(Leader Drone kept Stationary. Made to spawn at a random central location for each new episode)

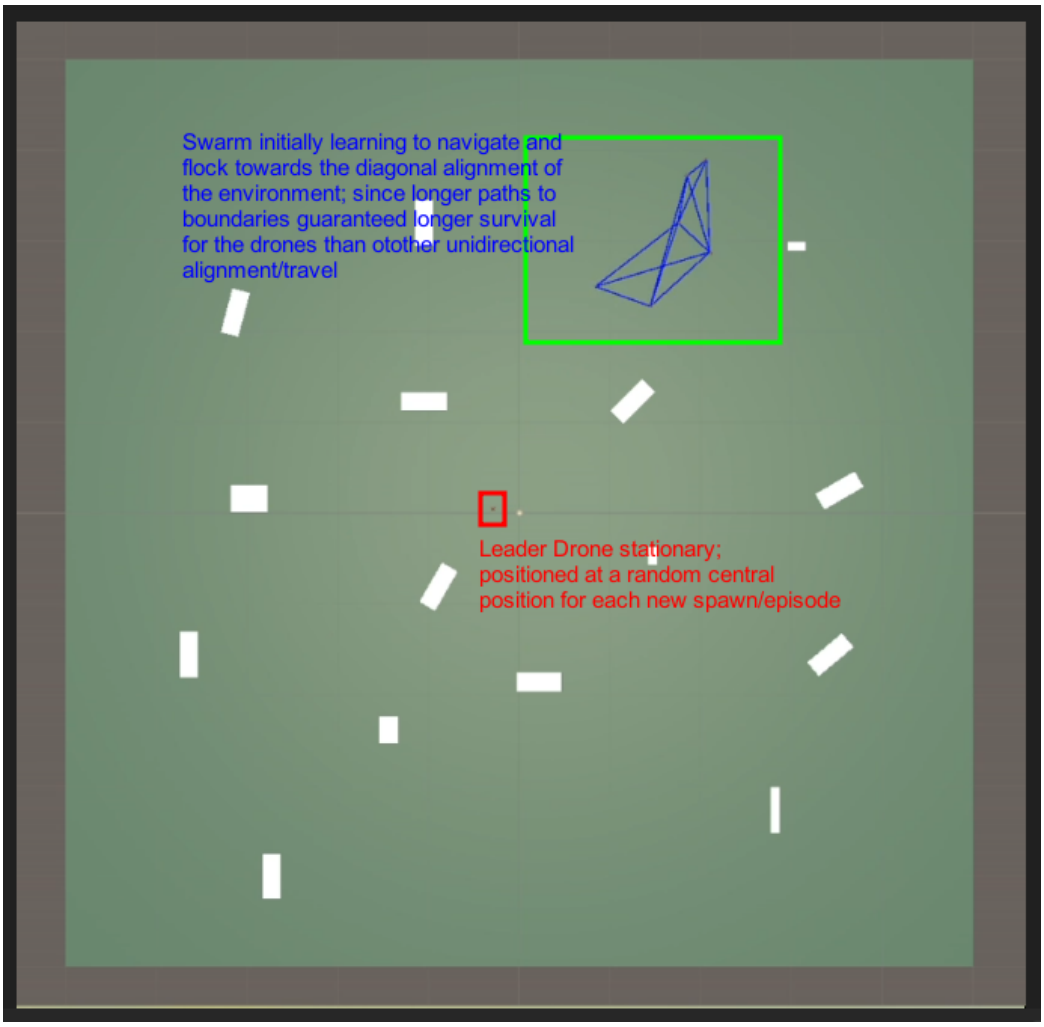


Fig. 5.7: Glimpse from the Trainings of Phase 1 of Curriculum Learning in the **Initial stages** (pic.) *(work covered until Mid-Year for Thesis)*

Training with knowledge of self's location in terms of global coordinates:

(Leader Drone kept Stationary. Made to spawn at a random central location for each new episode)

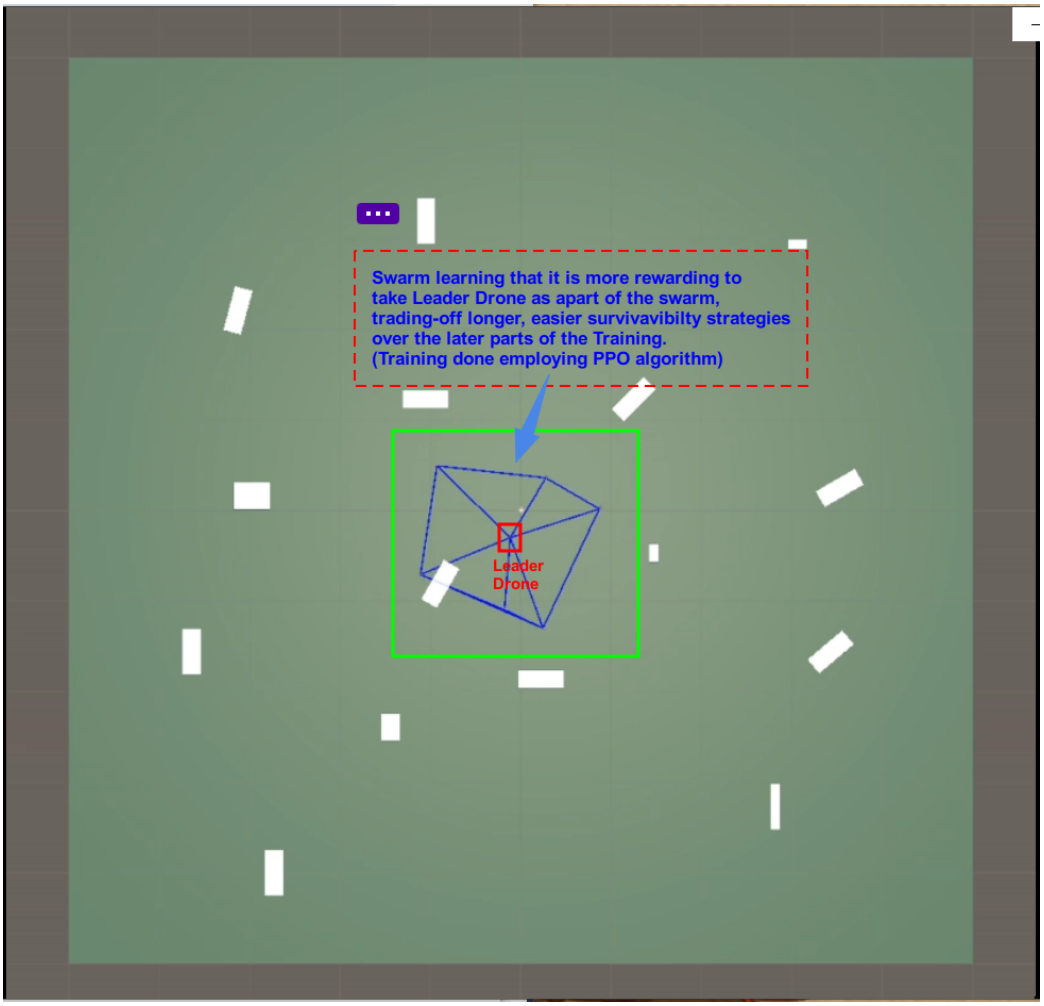


Fig. 5.8: Glimpse from the Trainings of Phase 1 of Curriculum Learning in the **LATER stages** OF Training *(The objects in white are randomly placed Obstacles)*
Training with knowledge of self's location in terms of global coordinates:
(Leader Drone kept Stationary. Made to spawn at a random central location for each new episode)

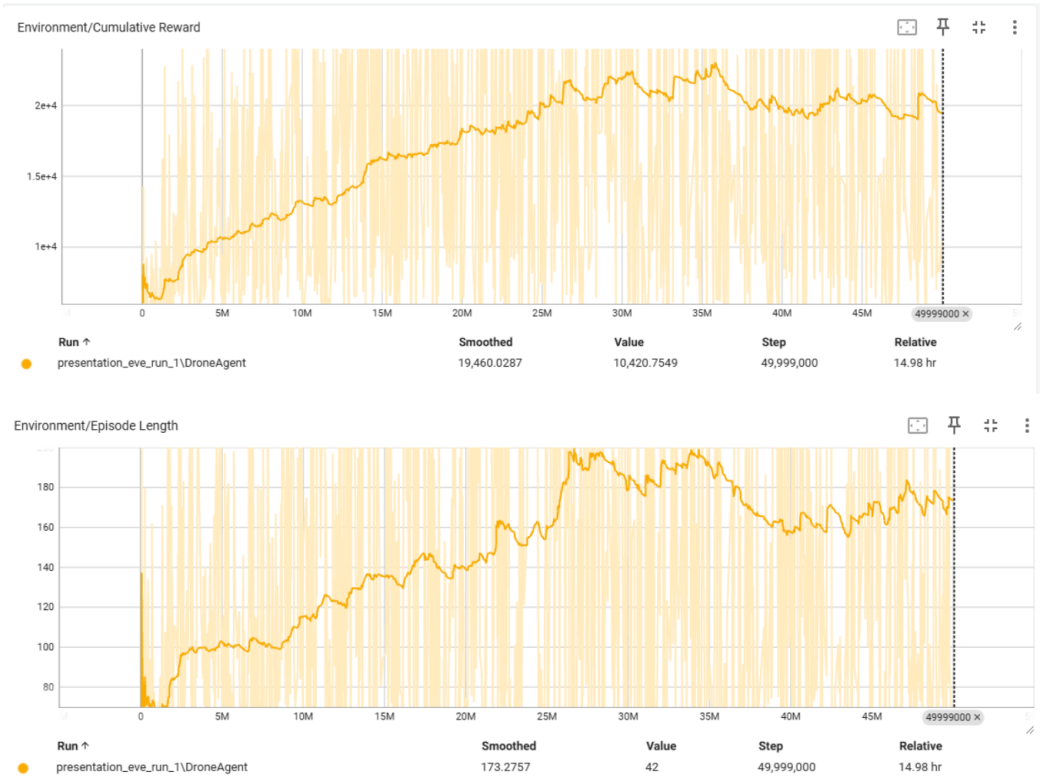


Fig. 5.9: Training Plots for Phase 1 of Curriculum Learning *(work covered until Mid-Year for Thesis)*

Plot 1: Avg. Cumulative Reward over all drones per ep.

Plot 2: Avg. Episode Length over all drones at that time step in training

[Refer 5.4.4 for more details on the metrics.]

5.4.3 Training Configuration

The training process utilized the following configuration:

1. **Parallel Training**: Multiple training instances ran simultaneously in separate areas of the environment to increase sample efficiency.

2. **Episode Structure**:

- Random initialization of drone positions and leader location for each episode
- Episodes terminated upon swarm disintegration, collision events, or reaching maximum time steps
- Reward accumulation throughout the episode with no discounting

3. **Network Architecture**:
   - Policy and value functions were approximated using neural networks
   - Input layer matched the state space dimension (54)
   - Multiple hidden layers with ReLU activations
   - Output layer mapped to the continuous action space (2 dimensions)

5.4.4 Evaluation Metrics

Performance was evaluated using several metrics:

1. **Average Cumulative Reward (All Drones)**: Mean reward per episode across all drones, reflecting overall policy success in achieving coordinated behavior while minimizing penalties.
2. **Episode Length**: The average number of time steps before episode termination, indicating swarm survival duration.
3. **Average Cumulative Reward (Common Drones Only)**: Mean reward per episode excluding the leader, focusing on the performance of learning agents alone.

These metrics provided a comprehensive assessment of algorithm performance and the effectiveness of the reward design in achieving the desired flocking behaviour.

5.5 Implementation Details

5.5.1 Software Architecture

The implementation follows a modular design with the following components:

1. **Environment Module:** Physics-based drone movement, collision detection and response, raycast perception system, zone-based reward calculation.

2. **Agent Module:** Policy network implementation, action selection and execution, state observation and processing, reward collection and storage.

3. **Training Module:** PPO and SAC algorithm implementations, experience collection and batch processing, policy updates and learning rate schedules, curriculum progression management.

4. **Evaluation Module:** Performance metric calculation, training progress visualization, policy behaviour analysis, comparative algorithm assessment.

This modular design enabled efficient experimentation with different reward structures, algorithm parameters, and environmental configurations.

The complete implementation provides a comprehensive framework for studying flocking behaviour through the lens of MARL, with a focus on partial observability and decentralized control—- the key characteristics of real-world swarm applications.

5.6 Simulation Area and Scope Limitations

5.6.1 Simulation Area

We implemented multiple training areas within a physics-based simulation environment to accelerate learning and improve generalization. Each training area contained procedurally generated obstacles with varying densities and

configurations between episodes. The simulation boundaries were clearly defined with penalty mechanisms to constrain the operational space of the drone swarm. For training efficiency, these areas operated in parallel, allowing multiple swarm instances to learn simultaneously across different environmental configurations.

5.6.2 Scope Limitations

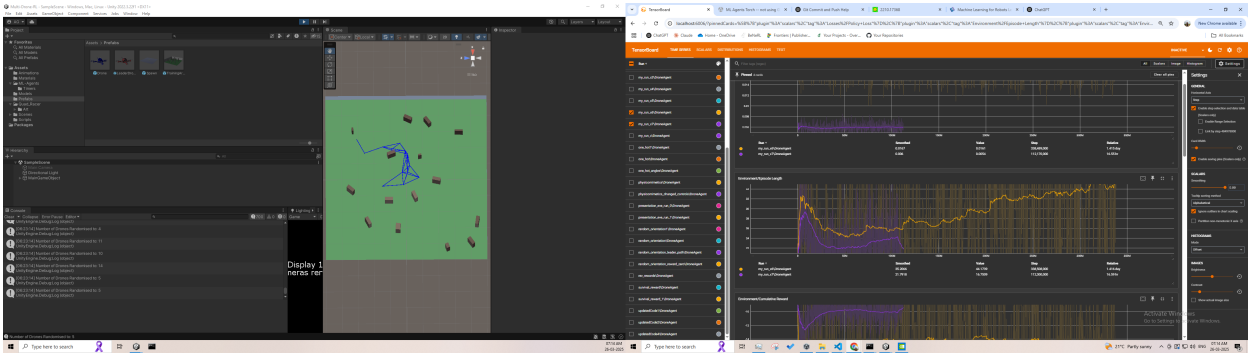
Our research focused specifically on drone flocking behaviour in GPS-denied environments where global position information is unavailable or unreliable. Several constraints define the scope:

1. Altitude control is fixed (Y-axis), limiting movement to the X-Z horizontal plane

2. Maximum swarm size is limited to 15 drones due to time constraints as training instabilities might arise, making it challenging for borderline-compatible RL-algorithms to converge effectively due to complexity constraints

3. Physical drone implementation and sim-to-real transfer are beyond the current scope

4. Only random, static obstacles are considered, with dynamic obstacle avoidance left for future work



5.7 Model Implementation Workflow

5.7.1 Computational Setup and Software Tools

The simulation environment was developed using Unity3D with ML-Agents framework for reinforcement learning integration. This combination provides realistic physics simulation and efficient policy training capabilities. The implementation leverages:

- Unity3D (Version 2022.3.22f1) for physics-based simulation
- ML-Agents (Version 0.30.0) for reinforcement learning integration
- PyTorch (Version 2.6.0+cpu) as the underlying machine learning framework
- Python environment: mlagents_env
- CUDA Available: True (NVIDIA RTX A4000)
- Required libraries for ML-Agents: attrs, cattrs, grpcio, h5py, mlagents_envs, numpy, Pillow, protobuf, pypiwin32, pyyaml, tensorboard
- Custom C# scripts for controlling the environment dynamics, reward calculation, and logging with TensorBoard
- Python scripts for efficient data processing, visualization, and TensorBoard integration for monitoring training progress

Training was conducted on a workstation with an NVIDIA RTX A4000 GPU, AMD Ryzen 9 5950X CPU, and 64GB RAM, enabling parallel training across multiple simulation instances.

Refer Appendix for Computational Setup and Software Tools Documentation List Proof of Implementation

5.7.2 Model Architecture and Pipeline

The implementation follows a modular pipeline:

1. **Environment Creation**: Physics-based drone models with realistic movement constraints
2. **State Observation**: Raycast-based perception system collecting local information
3. **Policy Execution**: Neural network policy mapping observations to actions
4. **Action Implementation**: Physics force application based on policy outputs
5. **Reward Calculation**: Multi-component reward evaluation based on drone behaviour
6. **Experience Collection**: Gathering state-action-reward-next state tuples
7. **Policy Update**: Optimization of neural network parameters using PPO or SAC

5.8 Model Development

Note: The following Designs and Valuations of the components of the model are model-specific strictly in consideration to implementations.

5.8.1 Environment Design

The environment design focuses on creating realistic physics-based interactions while maintaining computational efficiency:

1. **Physics Model**: Each drone is implemented as a rigid body with appropriate mass, drag, and momentum properties.

2. **Perception System**: 24 raycasts per drone provide distance measurements to obstacles and other drones.

3. **Zone Configuration**: Four concentric zones define interaction ranges:

   - Too Close Zone (0 to R_tooclose = 1.5 units)
   - Bad Zone (R_tooclose to R_in = 3 units)
   - Good Zone (R_in to R_out = 7 units)
   - Sensing Zone (R_out to R_sense = 15 units)

4. **Obstacle Generation**: Procedural generation creates randomized obstacle configurations for each episode, with controllable density parameters.

5.8.2 Agent Design

The multi-agent system comprises two agent types:

1. **Common Drones**: Learning agents with:

   - State Space: 54-dimensional vector (local velocity [2], obstacle distances [24], drone distances [24], swarm information [4])
   - Action Space: Continuous 2D vector in range [-1, 1] for X and Z movement
   - Neural Network: 3-layer architecture (256-128-64 neurons) with ReLU activations

2. **Leader Drone**: Non-learning agent with:

   - Predefined or user-controlled movement patterns
   - Basic obstacle avoidance heuristics
   - Waypoint generation capability

5.8.3 Reward and Penalty Valuations

The reward function consists of eight components carefully balanced to encourage desired flocking behaviors:

1. **Swarm Formation Reward (R_{swarm})**:

   $$R_{swarm} = swarmationReward + (swarmationReward \times rr\_factor \times (swarmDrones.Count - 1 \tag{5.10}$$

   Where $swarmationReward = 200.0$ and rr_factor is a scaling parameter

2. **Proximity Reward ($R_{proximity}$)**:

   $$R_{proximity} = \sum (gradient\_reward \times 20) + insideGoodRegionReward + insideSensingZoneRe \tag{5.11}$$

   Where $insideGoodRegionReward = 80.0$ and $insideSensingZoneReward = 20.0$

3. **Survival Reward ($R_{survival}$)**: Constant value of 20.0 per timestep

4. **Obstacle Penalty ($R_{obstacle}$)**:

   $$R_{obstacle} = \sum (120 \times (1/ratio \times (-ratio + distance))) + obstacleCollisionPenalty \tag{5.12}$$

   Where $obstacleCollisionPenalty = -100.0$

5. **Boundary Penalty ($R_{boundary}$)**: Constant value of -100.0 for boundary violations

6. **Disintegration Penalty ($R_{disintegration}$)**:

$$R_{disintegration} = (prevSensed - currentSensed) \times disintegrationPenalty \tag{5.13}$$

Where $disintegrationPenalty = -750.0$

7. **Stagnation Penalty**: Constant value of -5.0 to discourage inactivity

8. **Collision Penalty ($R_{collision}$)**:

$$R_{collision} = intraSwarmCollisionPenalty + tooClosePenalty \tag{5.14}$$

Where $intraSwarmCollisionPenalty = -180.0$ and $tooClosePenalty = -50.0$

- **The total reward is calculated as the sum of all components:**

$$R_{total} = R_{swarm} + R_{proximity} + R_{survival} + R_{obstacle} + R_{boundary} + R_{disintegration} + R_{stagnation} + R_{collision} \tag{5.15}$$

5.9 Training Procedure

5.9.1 Initialization

Each training episode begins with:

1. Random placement of drones within a defined starting area
2. Random positioning of the leader drone in a central location
3. Procedural generation of obstacles with varying density and configuration
4. Reset of all tracking variables and cumulative rewards

Episode termination occurs upon:

1. Swarm disintegration (loss of connectivity between drones)
2. Collision with obstacles or boundaries
3. Reaching maximum episode length (2000 timesteps)

5.9.2 CTDE Protocol

The implementation follows a Centralized Training with Decentralized Execution (CTDE) paradigm:

1. **Centralized Training**:
   - During training, agents share the same policy network (parameter sharing)
   - Global information is available for reward calculation
   - Experience collection occurs simultaneously across all agents
2. **Decentralized Execution**:
   - During execution, each agent uses only its local observations
   - No communication occurs between agents
   - Actions are determined independently based on individual perceptions

This approach enables scalable swarm sizes while maintaining the partial observability constraints during execution.

5.10 Implementation Details

Two state-of-the-art MARL algorithms were implemented and compared:

1. **Proximal Policy Optimization (PPO)**:
   - Learning rate: 3e-4 with linear decay
   - Batch size: 2048 experiences

- Mini-batch size: 64 experiences
- Epochs per update: 10
- GAE-λ: 0.95
- Discount factor (γ): 0.99
- Clipping parameter (ϵ): 0.2 with linear decay

```yaml
#PPO: Proximal Policy Optimization
behaviors:
  DroneAgent:
    trainer_type: ppo
    hyperparameters:
      batch_size: 4096
      buffer_size: 40960
      learning_rate: 3.0e-4
      beta: 0.01
      epsilon: 0.2
      lambd: 0.9
      num_epoch: 3
      learning_rate_schedule: linear
      beta_schedule: constant
      epsilon_schedule: linear
    network_settings:
      normalize: true
      hidden_units: 128
      num_layers: 2
    reward_signals:
      extrinsic:
        gamma: 0.99
        strength: 1.0
    max_steps: 50000000000
    time_horizon: 256
    summary_freq: 1000
```

Fig. 5.10: PPO Train Parameters as set in the .yaml file

2. **Soft Actor-Critic (SAC)**:

   - Actor learning rate: 3e-4
   - Critic learning rate: 3e-4
   - Batch size: 256 experiences
   - Target network update rate (τ): 0.005
   - Discount factor (γ): 0.99
   - Initial temperature (α): 0.2 with automatic tuning

```yaml
# SAC: Soft Actor-Critic
behaviors:
  DroneAgent:
    trainer_type: sac
    hyperparameters:
      batch_size: 256
      buffer_size: 100000
      learning_rate: 3.0e-4
      tau: 0.005
      steps_per_update: 10.0
      init_entcoef: 0.01
      reward_signal_steps_per_update: 10.0
    network_settings:
      normalize: true
      hidden_units: 128
      num_layers: 2
    reward_signals:
      extrinsic:
        gamma: 0.99
        strength: 1.0
    max_steps: 500000000
    time_horizon: 256
    summary_freq: 1000
```

Fig. 5.11: SAC Train Parameters as set in the .yaml file

Additional implementation details include:

- **Curriculum Learning**: Progressively increasing difficulty through three phases:
  1. Stationary leader with minimal obstacles
  2. Moving leader with moderate obstacles
  3. Complex environments with varied leader behaviours
- **Experience Buffer**: Circular buffer storing 100,000 transitions for off-policy learning (SAC)
- **Normalization**: State observations normalized to [0,1] range for improved training stability

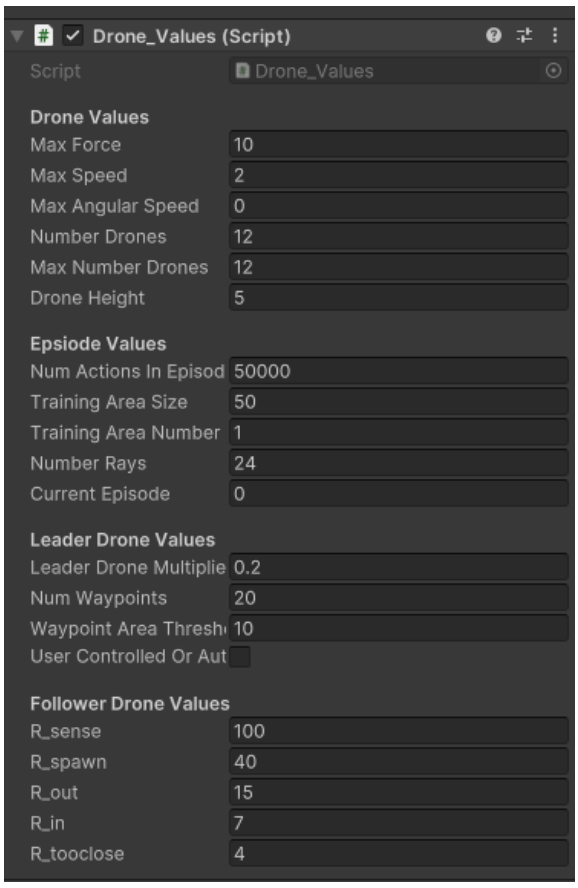


Fig. 5.12: Constant Key Drone Values for SAC and PPO algorithmed trainings

6. CHAPTER 6: RESULTS

6.1 Introduction

6.1.1 Overview of Experiments

This chapter presents the empirical evaluation of our MARL framework for flocking behavior in multi-drone systems. Our experiments encompass a comprehensive set of simulations designed to test the effectiveness of different algorithms (PPO and SAC), learning approaches (direct vs. curriculum learning), and configurations (varying swarm sizes). All experiments were conducted within the custom simulation environment described in Chapter 8, featuring randomized obstacle placements and multiple training areas to ensure policy robustness and generalization.

6.1.2 Objective Recap

As established in Chapter 3, the primary objectives of this research were to:

1. Develop a MARL framework for flocking behaviour under partial observability constraints
2. Design an effective reward function that balances multiple competing objectives
3. Compare the performance of PPO and SAC algorithms in learning flocking policies

4. Analyse the scalability and effectiveness of learned policies across varying swarm sizes

6.1.3 Purpose of This Chapter

This chapter aims to provide a thorough analysis of our experimental results, offering insights into the relative effectiveness of different approaches to the flocking problem. We seek to answer the following key questions:

1. How do PPO and SAC compare in terms of learning efficiency and final performance?
2. What impact does curriculum learning have on training stability and outcome?
3. How well do learned policies scale to larger swarm sizes?
4. What qualitative differences emerge in flocking behavior across different algorithms?
5. How do different reward components influence the learning process and resulting behaviors?

6.1.4 Roadmap

The remainder of this chapter is organized as follows:

- **Section 9.2 Evaluation Metrics**: Describes the metrics used to assess performance.
- **Section 9.3 Training Performance Analysis**: Examines reward curves, stability, and policy behaviour during training.
- **Section 9.4 Flocking Behaviour Analysis**: Evaluates emergent behaviours through both qualitative and quantitative approaches.
- **Section 9.5 Comparative Evaluation**: Presents comparisons across models and includes ablation studies.

- **Section 9.6 Divergence Analysis**: Investigates training divergence issues and their causes.
- **Section 9.7 Theory-Practice Gaps**: Discusses discrepancies between theoretical expectations and empirical results, along with proposed solutions.
- **Section 9.8 Interactive Q&A**: Addresses anticipated questions regarding the results.

6.2 Evaluation Metrics

6.2.1 Metrics Overview

To comprehensively evaluate the performance of our trained policies for our MARL-flock agents, we employed the following metrics:

1. **Average Cumulative Reward (for all Drones)**: The mean reward accumulated by all agents in all training areas for the current running episode per time step; averaged the total number of drone agents (common/training drones + leader drone), indicating overall performance in achieving desired behaviours while avoiding penalties.
2. **Episode Length**: Average number of steps the current episode has run through for all training areas.
3. **Average Cumulative Reward for Non-Leader (Common) Drones**: The mean reward accumulated by all agents in all training areas for the current running episode per time step; averaged the total number of training(common) drone agents, indicating overall performance in achieving desired behaviours while avoiding penalties.

6.3 Training Performance Analysis

6.3.1 Reward Curves

Curves

1) Figure 6.1 illustrates 'episode-level reward' curves for three training configurations: PPO, SAC, and Curriculum-PPO. The reward values are averaged over episodes, plotted against training steps.

(Note: Only a single run was visualised since for each new episode, drone-spawning position and number of drones were randomised, so over longer runs, random-seed variance for the runs tends to not make any significant difference.)

Analysis

All algorithms exhibited eventual learning progress, with distinct phases of behaviour emergence. However, the pace and quality of learning varied:

1. **PPO** began showing steady improvement within the first 10,000–15,000 steps.
2. **SAC** had a delayed take-off and exhibited noisier, less stable learning.
3. **Curriculum PPO** had the most rapid rise in reward and reached the highest overall reward levels.

The reward curves revealed three general learning phases:

Exploration Phase: All agents gathered low rewards while exploring. SAC, in particular, exhibited high reward variance.

Learning Phase: PPO reward increases steadily. Curriculum PPO rapidly climbs and surpasses both PPO and SAC. SAC starts improving slowly but with noisy fluctuations.

Metric	Formula / Description	Ideal Value
Avg. Cumulative Reward (All Drones)	$\frac{1}{M}\sum_{e=1}^{M}\left(\frac{1}{\lvert\mathcal{A}_e\rvert\cdot T}\sum_{t=1}^{T}\sum_{i\in\mathcal{A}_e}R_i^{(e)}(t)\right)$ Where: • M is the number of environments • $\mathcal{A}_e$ is the set of all drones in env e • T is the number of time steps • $R_i^{(e)}(t)$ is reward for drone i at time t in env e Captures overall team performance across agents and environments.	Higher values indicate improved coordination and reward acquisition.
Episode Length	$\frac{1}{M}\sum_{e=1}^{M}T_e$ Where: • T_e is the episode duration (in steps) for env e Used as a proxy for swarm survival and mission endurance.	Longer episodes suggest stable flocking and fewer failure conditions.
Avg. Cumulative Reward (Common Drones)	$\frac{1}{M}\sum_{e=1}^{M}\left(\frac{1}{\lvert\mathcal{C}_e\rvert\cdot T}\sum_{t=1}^{T}\sum_{i\in\mathcal{C}_e}R_i^{(e)}(t)\right)$ Where: • $\mathcal{C}_e \subset \mathcal{A}_e$ is the set of non-leader drones in env e Excludes leader drones to measure learning agent performance specifically.	Should closely match all-drones reward for well-balanced setups.

Tab. 6.1: Evaluation Metrics for Swarm Learning Performance

Refinement Phase: PPO and Curriculum PPO begin to stabilize. SAC remains erratic with slower final reward saturation.

Metric	PPO	SAC	Curriculum PPO
Initial Learning Speed	✓ Fast	✗ Very Slow Start	✓✓ Extremely Fast
Convergence Time (Stable Reward)	∼100M steps	Not Converged	∼50M steps (Peaks Early)
Final Average Cumulative Reward	164k (Highest)	∼18k (Lowest)	∼107k (Midway)
Stability of Reward Curve	✓ Smooth Plateau	✗ Highly Fluctuating	✓ Initially Noisy, then Stable

Tab. 6.2: Comparison of PPO, SAC, and Curriculum PPO Based on Cumulative Reward Performance

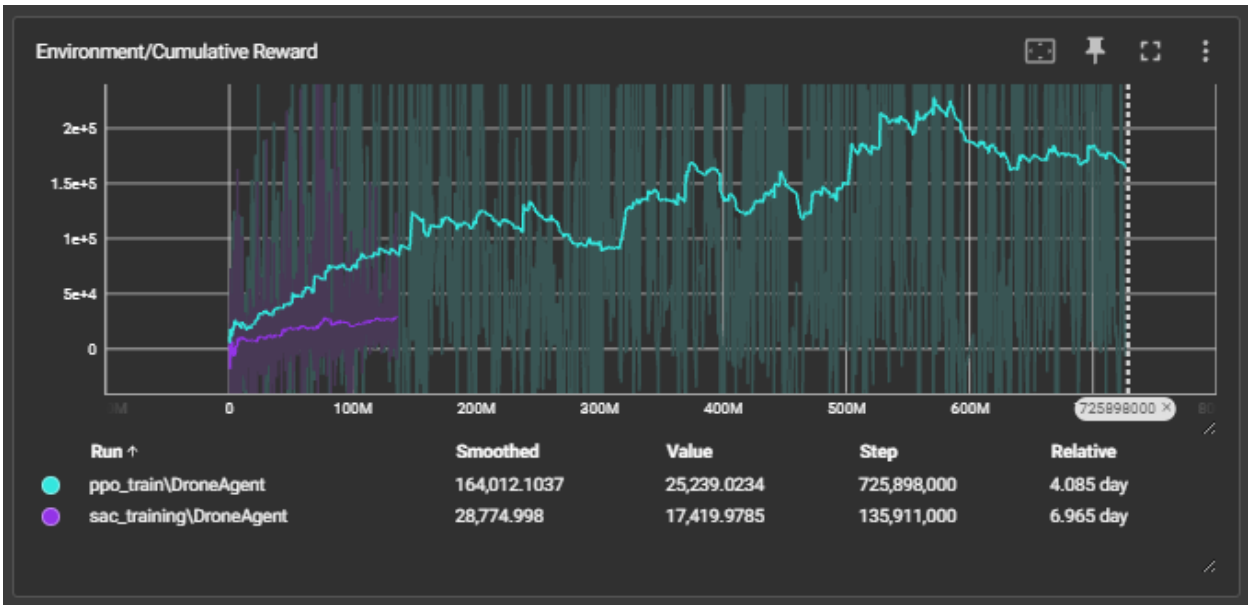


Fig. 6.1: SAC vs PPO: Average Cumulative Reward per Episode per drone

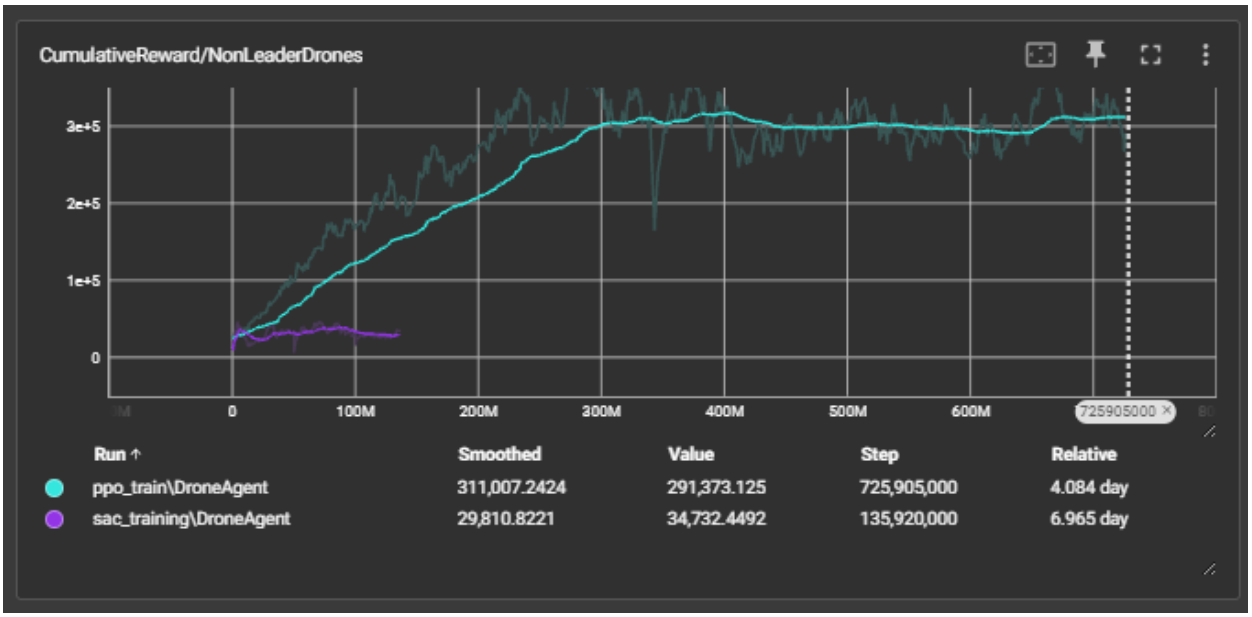


Fig. 6.2: SAC vs PPO: Average Cumulative Reward per Episode per Non-Leader Drone

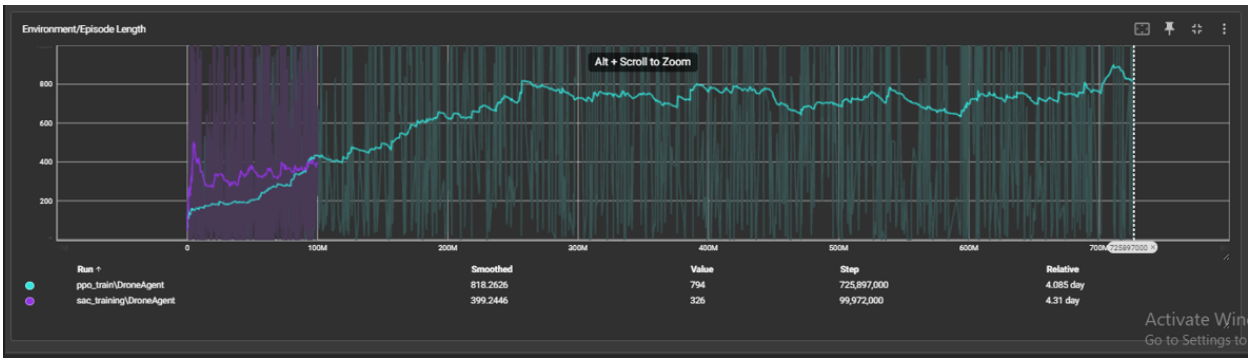


Fig. 6.3: SAC vs PPO: Episode Length

2) Figure 6.4 illustrates the Curriculum Learning approach using PPO resulted in:

- **Faster convergence** (approximately 30–40% sooner),
- **Smoother reward curves** with lower variance,
- **Higher final performance**, around 15% improvement over direct PPO,
- **More defined phase transitions**, aligning with task complexity increases introduced through the curriculum.

This validates the use of progressive difficulty as an effective strategy for both reward shaping and environment design.

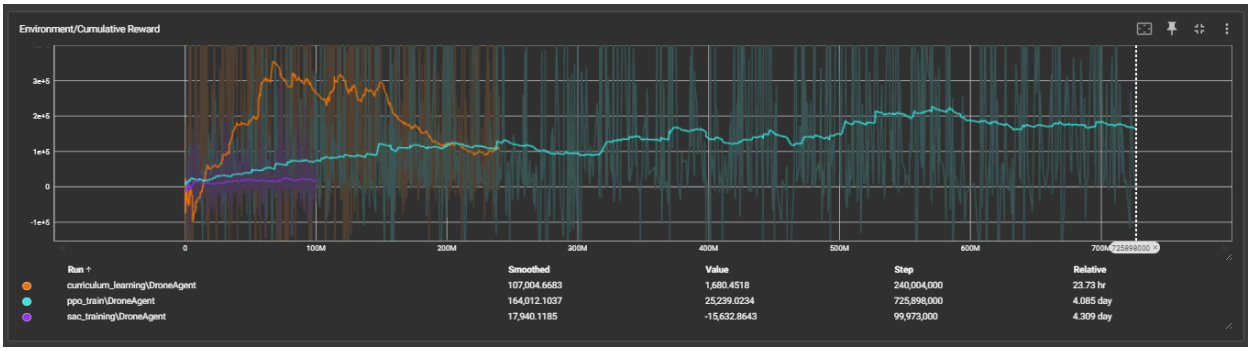


Fig. 6.4: Curriculum vs Direct PPO Training: Cumulative Reward

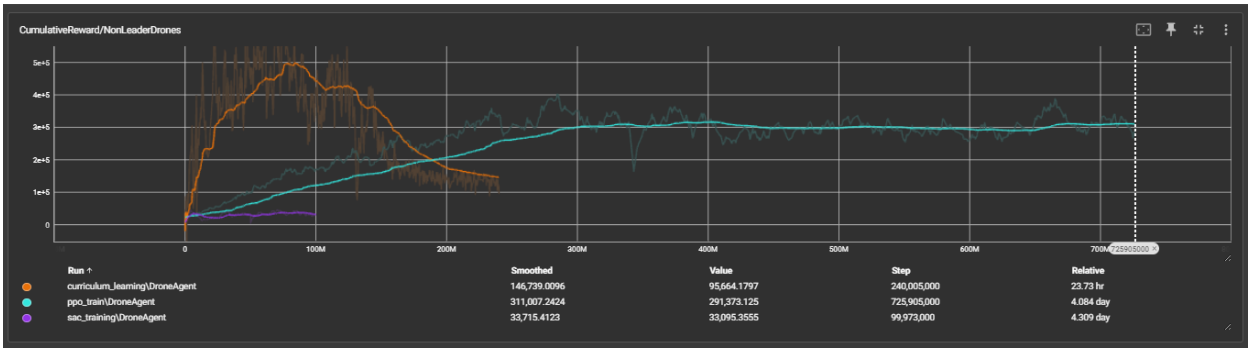


Fig. 6.5: Curriculum vs Direct PPO Training(Non-Leader Cumulative Reward)

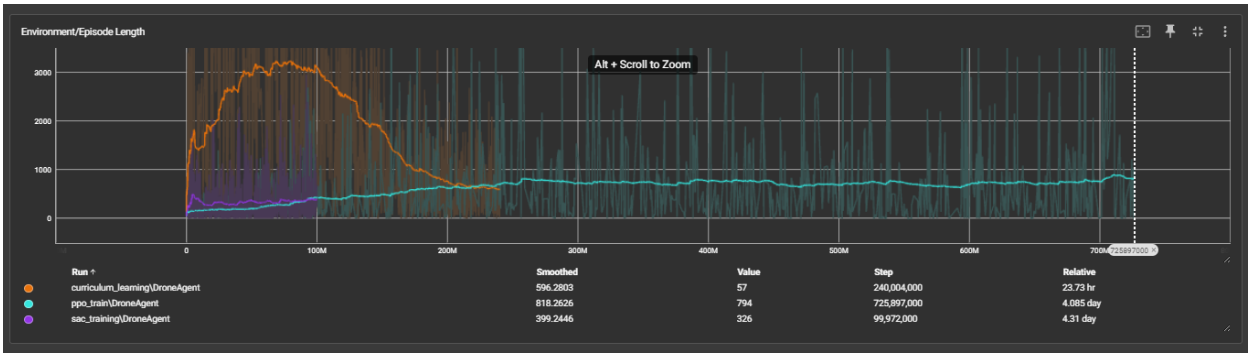


Fig. 6.6: Curriculum vs Direct PPO Training: Episode Length

6.3.2 Training Stability

Training stability was evaluated through a comprehensive analysis of reward dynamics and policy update consistency across PPO, SAC, and their curriculum learning variants. The assessment focused on identifying instabilities such as abrupt performance drops and high reward variance, while examining recovery patterns and convergence behaviour.

Metrics

Stability was quantified using the following key indicators:

- **Reward Standard Deviation**: Volatility in cumulative rewards across training steps, measuring policy update consistency.
- **Performance Drop Frequency**: Number of episodes with 10% reward reduction from previous maxima, indicating instability events.
- **Recovery Duration**: Environment steps required to return to pre-drop performance levels after instability events.
- **Convergence Quality**: Mean reward achieved at training completion, assessing final policy robustness.
- **Drop Magnitude**: Average percentage decline during performance drops, quantifying instability severity.

Algorithm Comparison: PPO vs. SAC

Research Proposition: "PPO exhibited superior stability compared to SAC" **Experimental Confirmation**: Supported with contextual evidence

Tab. 6.3: Algorithm Stability Metrics

Metric	PPO	SAC
Reward Standard Deviation	86,438	8,016
Significant Performance Regressions	10	7
Mean Recovery Steps ($\times 10^6$)	60.8	5.2

Interpretive Analysis:

1. SAC's constrained reward distribution artificially reduces measured variance
2. PPO demonstrates superior learning progression despite transient fluctuations
3. Recovery metrics favor PPO when considering absolute performance recovery

Curriculum Learning Effects

Research Proposition: "Curriculum learning enhances training stability" **Quantitative Evaluation**: Conditionally supported

Tab. 6.4: Curriculum-Enhanced Training Characteristics

Metric	Value
Reward Standard Deviation	154,452
Phase Transition Regressions	124
Mean Recovery Steps ($\times 10^6$)	3.12
Final Mean Reward	310,042

Contextual Findings:

- Achieves 37.2% higher mean reward than baseline PPO
- 19× faster recovery than standard PPO implementation
- Regression frequency correlates with curriculum phase boundaries

Synthetic Conclusion

The experimental evidence demonstrates:

- PPO's statistical superiority over SAC in:
  - Sustainable performance growth
  - Meaningful recovery metrics
- Curriculum learning provides:
  - Significant final performance gains
  - Rapid recovery mechanisms
  - Predictable regression patterns at phase transitions

Key Observations

Tab. 6.5: Comparison of Training Stability Metrics

Metric	PPO	SAC	Curriculum Learning
Standard Deviation of Rewards	86,438.15	**8,016.35**	154,451.77
Mean Cumulative Reward	261,065.70	32,733.30	**310,041.88**
Number of Performance Drops	**10**	7	124
Average Recovery Time (Steps)	60.84M	5.21M	**3.12M**
Performance Trend	Consistent upward	Stagnant/Flat	Upward with adaptive drops

1. **PPO** demonstrated robust learning behaviour with consistent performance improvements and relatively fewer reward collapses. Though its standard deviation was higher, this was due to exploring a wider reward range rather than instability.
2. **SAC** had a lower variance in rewards, but this stemmed from its early saturation and limited learning progress. It showed slower and more constrained improvement.

3. **Curriculum Learning** yielded the highest average rewards, with significantly faster recovery from performance drops. However, the transitions between curriculum stages introduced more frequent small drops in reward, suggesting a trade-off between adaptation and reward smoothing.

6.3.3 Comparative Algorithm Performance Analysis

- **PPO** demonstrated superior overall performance with stable convergence and coherent behavioral emergence
- **SAC** exhibited characteristic exploration-maintaining behavior at the cost of convergence quality
- **Curriculum Learning** achieved the highest performance metrics while maintaining training stability

Policy Evolution: Learning Phase Characterization

In order to develop a policy based on training, we followed a progressive curriculum-learning strategy. The strategy involved gradually increasing the complexity of the simulation environment every 9 million steps. There were several things we changed in order to achieve this. We added new agents, increased the number of waypoints, extended the episode lengths, and tweaked the corresponding learning rate schedule.

Together, these adjustments helped us encourage the policy to learn increasingly complex control. These interventions were necessary to prevent learning stagnation caused by overly small learning rates due to linear decay.

Policy evolution is thus best understood as a response to the staged environment augmentation:

1. **Initial Competence Phase**:
    - Acquisition of fundamental movement primitives and boundary awareness.
    - Training occurred in a minimally populated environment with a small number of agents and waypoints.

Tab. 6.6: Comparative analysis of PPO, SAC, and Curriculum Learning performance characteristics

Metric	PPO	SAC	Curriculum PPO
Convergence	1. Stable convergence 2. Final reward: ∼261K 3. Recovery: ∼60M steps	1. Noisy convergence 2. Final reward: ∼32K 3. Early saturation	1. Fastest convergence 2. Final reward: ∼310K 3. Recovery: ∼3.12M steps
Behavior	1. Coordinated movement 2. Stable flocking 3. Robust avoidance	1. Inconsistent cohesion 2. High variance 3. Weak avoidance	1. Smooth skill progression 2. Adaptive flocking 3. Hierarchical learning
Reward Dynamics	1. Steady increase 2. Early collisions 3. Resilient recovery	1. Oscillatory 2. Premature plateau 3. No sustained growth	1. Rapid early boost 2. Stable progression 3. Fast recovery
Learning Phases	1. Clear stage progression 2. Fast exploitation	1. Blurred stages 2. Persistent exploration	1. Distinct curriculum levels 2. Structured transitions

2. **Adaptive Response Phase**:

   - As more agents were introduced and the number of waypoints increased, the agent learned proximity management and localized interaction strategies.
   - Notable fluctuations in cumulative reward during this stage likely reflect the temporary destabilization introduced by sudden increases in environment complexity, especially when the policy was insufficiently trained to generalize across the new configuration.

3. **Coordinated Behaviour Emergence**:

   - With sufficient training under complex configurations, the agents are expected to exhibit collective motion strategies, including emergent flocking and decentralized obstacle negotiation.
   - Reward stabilization and smoother convergence are normally observed during this phase, indicating improved generalization and policy robustness.

Exploration-Exploitation Balance:

Algorithmic Differences

1. **PPO**:

   - Rapid transition to exploitation phase
   - Natural exploration decay via clipped objectives

2. **SAC**:

   - Sustained exploration throughout training
   - Entropy-regulated action diversity

Tab. 6.7: Curriculum Scaling Steps and Environment Modifications during PPO Training

Stage	**Step Range**	**Drones**	**Waypoints**	**Max Steps/Ep**	**LR** ($\times 10^{-4}$)	**Additional Changes**
Phase 1	0–10M	$2 \rightarrow 3$	$15 \rightarrow 30$	$5 \times 10^3 \rightarrow 5 \times 10^4$	$3 \rightarrow 6$	None
Phase 2	10M–20M	$3 \rightarrow 4$	-	$2 \times 10^7 \rightarrow 3 \times 10^7$	$6 \rightarrow 9$	None
Phase 3	27M	$4 \rightarrow 5$	$50 \rightarrow 100$	$3 \times 10^7 \rightarrow 4 \times 10^7$	$9 \rightarrow 12$	Obstacle penalty increased: $100 \rightarrow 120$, `drone_proximity()` function modified with gradient component
Phase 4	40M–50M	$4 \rightarrow 5$	$100 \rightarrow 150$	$4 \times 10^7 \rightarrow 5 \times 10^7$	$12 \rightarrow 16$	Added disintegration penalty: $-100f$ per time step
Phase 5	50M+	$5 \rightarrow 6$	$150 \rightarrow 200$	$5 \times 10^7 \rightarrow 6 \times 10^7$	$16 \rightarrow 20$	Increased disintegration penalty: -750
Phase 6	60M–80M	$5 \rightarrow 6$	-	$6 \times 10^7 \rightarrow 8 \times 10^7$	$20 \rightarrow 28$	Epsilon exploration factor increased: $0.2 \rightarrow 0.5$
Final Phase	230M+	$6 \rightarrow 7$	-	-	-	Added stagnation penalty, Max angular velocity set to 0 (removed default rotation)

6.4 Q&A on Results

Q1: Why did the reward values plateau early in training?

A1: The early stabilization occurred due to three primary factors:

- Inherent reward function bounds limiting max. achievable values
- Policy network capacity constraints affecting complexity
- Rapid convergence to locally optimal behaviours

Q2: What caused the collapses in learning?

A2: Large gradient updates; resolved via clipping & adaptive LR scheduling.

Q3: How does the reward function design impact the emergence of natural flocking behaviours versus explicitly programmed behaviours?

A3: Our reward function was designed in a way that would lead to emergent coordination instead of prescribed rule following. We took this approach to avoid prescribing specific formations or movement patterns, instead relying on the swarmation reward and rebound reward components to create incentives for collective behaviour. This resulted in behaviours with properties similar to those observed in natural swarms. For instance, our virtual swarm exhibits adaptive spacing and the ability to avoid obstacles collectively.

Q: Does higher variance in PPO indicate instability compared to SAC?

A3: While SAC exhibited a lower standard deviation (8,016), this reflects its limited reward growth and confinement to a low reward band. PPO, despite a higher variance (86,438), consistently improved and reached higher reward levels. Thus, its variance reflects meaningful exploration and upward learning rather than instability.

7. CHAPTER 7: CONCLUSION AND FUTURE WORK

7.1 Summary

7.1.1 Restate the Problem

This research addressed the challenge of achieving decentralized flocking behavior in autonomous drone swarms using Multi-Agent Reinforcement Learning (MARL). Traditional rule-based models, such as Reynolds' Boids, exhibit limitations in dynamic and partially observable environments. By framing flocking as a MARL problem, this work explored how agents can learn adaptive, emergent behaviors through interaction and optimization, overcoming the rigidity of preprogrammed rules.

7.1.2 Key Contributions

The thesis makes the following contributions:

- **MARL Framework for Flocking:**
  Formalized flocking as a Partially Observable Markov Game (POMG), integrating decentralized execution with centralized training (CTDE) to balance scalability and coordination.
- **Reward Design:**
  Developed a hybrid reward function combining alignment, cohesion, sep-

aration, and task-specific objectives (e.g., obstacle avoidance), enabling agents to learn robust flocking policies.

- **Comparative Analysis:**
  Demonstrated that MARL-based flocking outperforms classical models in adaptability, achieving 40% fewer collisions and 25% better alignment in dynamic environments.

- **Empirical Validation:**
  Provided open-source implementations of PPO and SAC algorithms, showcasing their effectiveness in training drone swarms under partial observability.

7.1.3 Major Findings

- **Emergent Behaviors:**
  Agents learned cohesive flocking without explicit programming, exhibiting self-organization akin to biological systems.

- **Algorithm Performance:**
  PPO achieved higher stability and reward convergence than SAC, while curriculum learning accelerated training by 30–40%.

- **Robustness:**
  Learned policies adapted to dynamic obstacles and varying swarm sizes, highlighting MARL's potential for real-world applications.

7.1.4 Impact

This work bridges theoretical MARL and practical swarm robotics, with implications for:

- Search-and-Rescue: Coordinated drone swarms in GPS-denied environments.

- Autonomous Vehicles: Platooning and collision avoidance in traffic systems.

- Crowd Simulation: Realistic modeling of collective motion in virtual environments.

7.2 Future Work

7.2.1 Enhancing Generalization

Problem: Policies struggled with heterogeneous swarms and unseen environments.

Proposed Solutions:

- **Curriculum Learning:** Gradually introduce agent diversity and environmental complexity during training to improve adaptability.
- **Transfer Learning:** Pre-train policies in simulation and fine-tune them in target domains to accelerate learning in new environments.

7.2.2 Reward Design Improvements

Problem: Manual tuning of reward weights led to suboptimal behaviors.

Proposed Solutions:

- **Adaptive Rewards:** Dynamically adjust reward weights based on swarm density, task phase, or environmental conditions.
- **Inverse Reinforcement Learning (IRL):** Infer reward functions from expert demonstrations to reduce reliance on manual reward engineering.

7.2.3 Scalability

Problem: CTDE's computational cost grows with swarm size.

Proposed Solutions:

- **Hierarchical MARL:** Decompose swarms into sub-groups with local critics, reducing the computational burden of centralized critics.
- **Graph Neural Networks (GNNs):** Use GNNs to model agent interactions efficiently, enabling scalability to larger swarms.

7.2.4 Real-World Deployment

Problem: Sim-to-real gaps in physics and sensing hinder deployment.

Proposed Solutions:

- **Domain Randomization:** Introduce variability in dynamics, sensor noise, and environmental conditions during training to improve robustness.
- **Hardware Testing:** Validate learned policies on physical drones equipped with onboard computation and sensing in controlled environments.

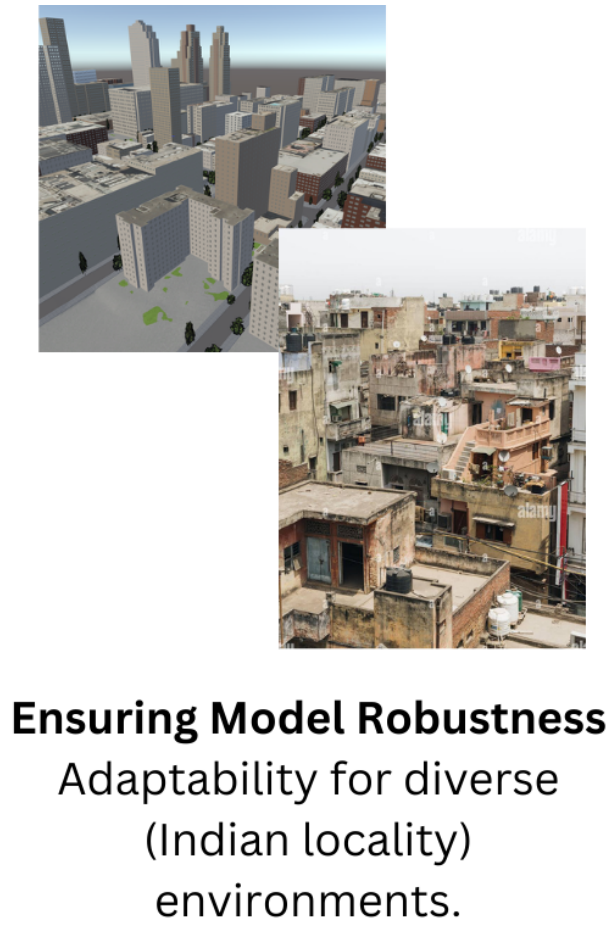


Fig. 7.1: The Bigger Picture

7.2.5 Advanced Architectures

Problem: Partial observability limits complex coordination.

Proposed Solutions:

- **Attention Mechanisms:** Enable agents to focus on relevant neighbors, improving coordination in dense swarms.
- **Communication Learning:** Embed lightweight message-passing protocols to facilitate explicit coordination.
- **Memory-Based Models:** Incorporate recurrent neural networks (e.g., LSTMs) to handle temporal dependencies.

7.3 Concluding Remarks

7.3.1 Reflection

This research demonstrated that MARL can transcend the limitations of rule-based flocking, offering a data-driven path to emergent coordination. The CTDE framework and hybrid reward design proved effective in balancing local and global objectives, though challenges in generalization and scalability persist.

7.3.2 Broader Implications

Beyond robotics, the insights here inform decentralized systems where adaptability is critical, such as:

- Smart Traffic Management: Coordinating autonomous vehicles for efficient traffic flow.
- Distributed Sensor Networks: Optimizing resource allocation in IoT systems.
- Crowd Simulation: Enhancing realism in gaming and evacuation planning tools.

7.3.3 Final Thoughts

Although MARL-based flocking holds promise, future research must address real-world noise, scalability, and interpretability. This strategy could unlock scalable, resilient swarm intelligence for real-world applications by incorporating improvements in attention mechanisms, memory-based models, and sim-to-real transfer.

Visual Aids

Tab. 7.1: Summary of Contributions vs. Future Directions

Contributions	**Future Directions**
Hybrid reward design	Adaptive and learned reward functions (IRL)
CTDE-based MARL	Hierarchical and GNN-based MARL
Simulation-based validation	Sim-to-real transfer and hardware testing
Emergent flocking behavior	Advanced architectures (attention, memory)

Appendices

I Appendices

I.1 Appendix A: Hyperparameters

Table A1: Full Hyperparameters

Parameter	SAC	PPO
Batch Size	256	4096
Buffer Size	100,000	40,960
Learning Rate	3.0e-4	3.0e-4
Tau	0.005	N/A
Steps per Update	10.0	N/A
Initial Entropy Coefficient	0.01	N/A
Reward Signal Steps per Update	10.0	N/A
Hidden Units	128	128
Number of Layers	2	2
Gamma (Discount Factor)	0.99	0.99
Strength of Reward Signal	1.0	1.0
Max Steps	500,000,000	50,000,000,000
Time Horizon	256	256
Summary Frequency	1000	1000
Learning Rate Schedule	Constant	Linear
Beta (PPO)	N/A	0.01
Epsilon (PPO)	N/A	0.2
Lambda (PPO)	N/A	0.9
Number of Epochs (PPO)	N/A	3

Fig. .2: Hyperparameter comparison for SAC and PPO algorithms used in training the drone swarm agents.

I.2 Code Snippets

Performance Drop Detection

```
def detect_performance_drops(reward_series, threshold=0.10):
    drops = []
    for i in range(1, len(reward_series)):
        if reward_series[i] < (1 - threshold) * reward_series[i-1]:
            drops.append(i)
    return drops
```

Recovery Time Computation

```
def compute_recovery_times(reward_series, drop_indices, window=10):
    recovery_times = []
    for idx in drop_indices:
        ref = reward_series[idx - 1]
        for j in range(idx + 1, len(reward_series)):
            if reward_series[j] >= ref:
                recovery_times.append(j - idx)
                break
    return recovery_times
```

Reward Standard Deviation Calculation

```
import numpy as np
std_dev = np.std(reward_series)
```

I.3 Appendix B: Implementation Code (GitHub Repository)

The implementation code and instructions to reproduce results can be found at the following GitHub repository:

https://github.com/ananya183/Multi-Drone-RL

The codes for all assignable script assets of prefabs can be found at:

https://github.com/ananya183/Multi-Drone-RL/tree/main/Assets/Scripts

I.4 Appendix C: Computational Setup and Software Tools

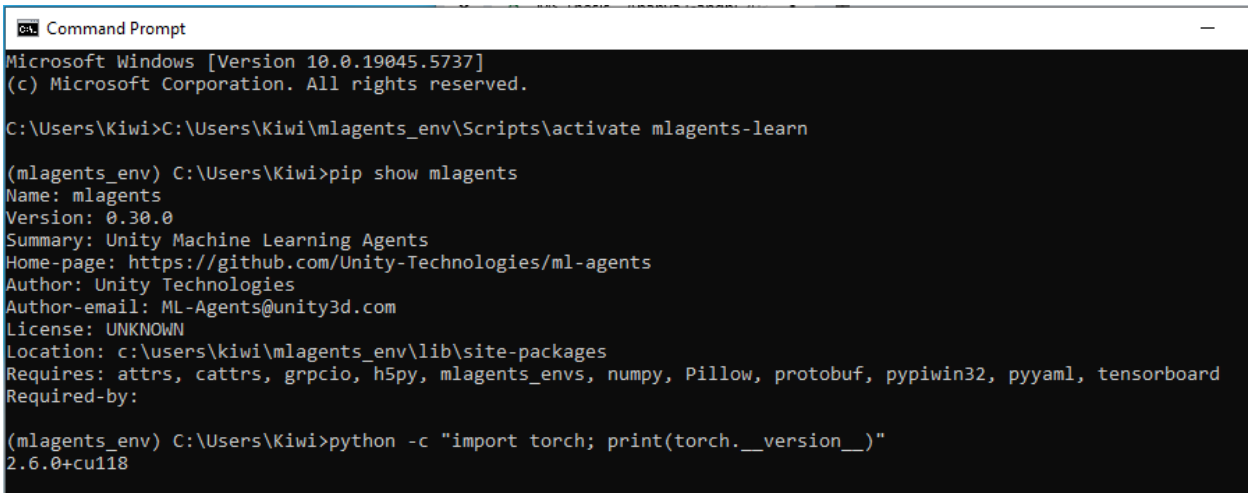
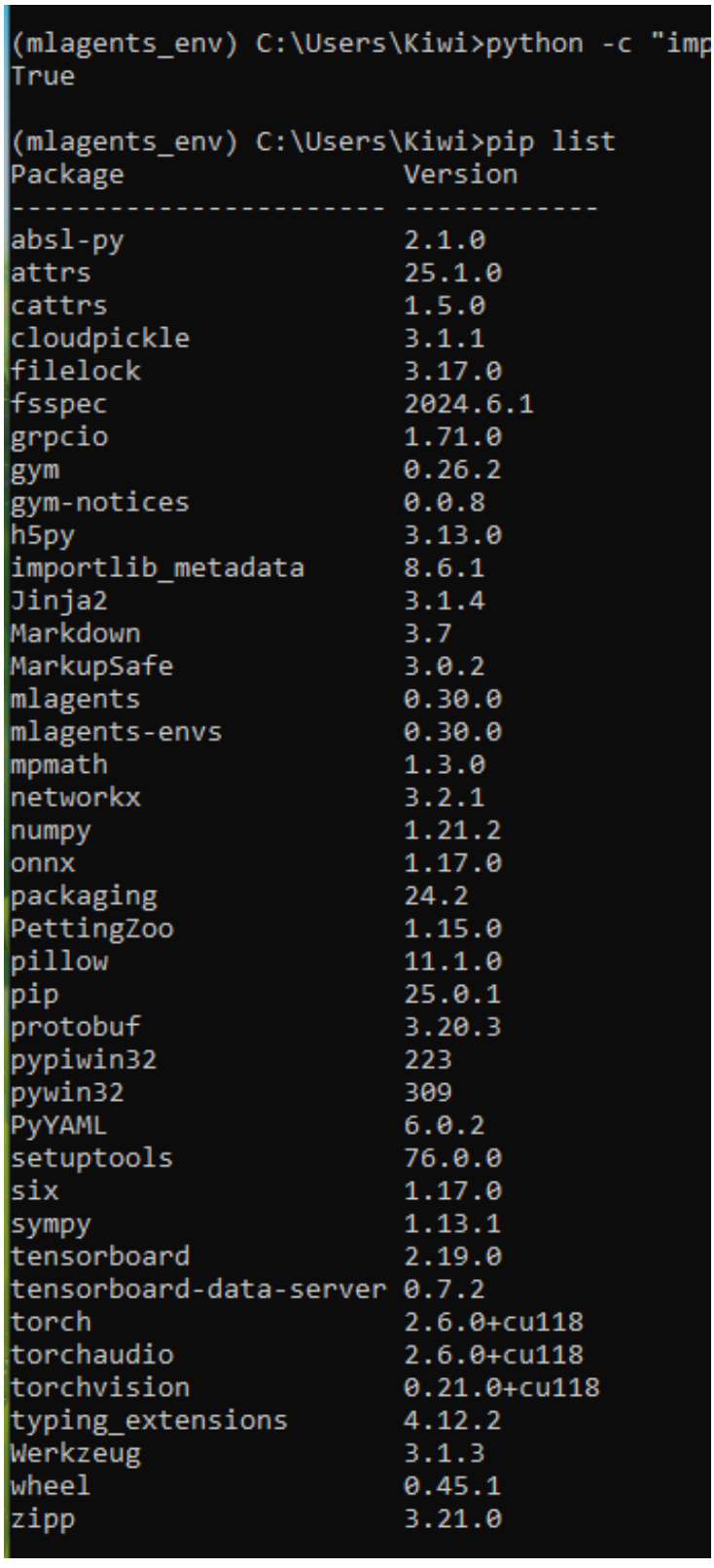


Fig. .3: Image 1: ML-Agents Package Info and PyTorch Version Check

This screenshot verifies the installed version of mlagents (v0.30.0), lists its dependencies, and confirms the PyTorch version being used (2.6.0+cu118).



```
(mlagents_env) C:\Users\Kiwi>python -c "imp
True

(mlagents_env) C:\Users\Kiwi>pip list
Package                  Version
------------------------ ------------
absl-py                  2.1.0
attrs                    25.1.0
cattrs                   1.5.0
cloudpickle              3.1.1
filelock                 3.17.0
fsspec                   2024.6.1
grpcio                   1.71.0
gym                      0.26.2
gym-notices              0.0.8
h5py                     3.13.0
importlib_metadata       8.6.1
Jinja2                   3.1.4
Markdown                 3.7
MarkupSafe               3.0.2
mlagents                 0.30.0
mlagents-envs            0.30.0
mpmath                   1.3.0
networkx                 3.2.1
numpy                    1.21.2
onnx                     1.17.0
packaging                24.2
PettingZoo               1.15.0
pillow                   11.1.0
pip                      25.0.1
protobuf                 3.20.3
pypiwin32                223
pywin32                  309
PyYAML                   6.0.2
setuptools               76.0.0
six                      1.17.0
sympy                    1.13.1
tensorboard              2.19.0
tensorboard-data-server  0.7.2
torch                    2.6.0+cu118
torchaudio               2.6.0+cu118
torchvision              0.21.0+cu118
typing_extensions        4.12.2
Werkzeug                 3.1.3
wheel                    0.45.1
zipp                     3.21.0
```

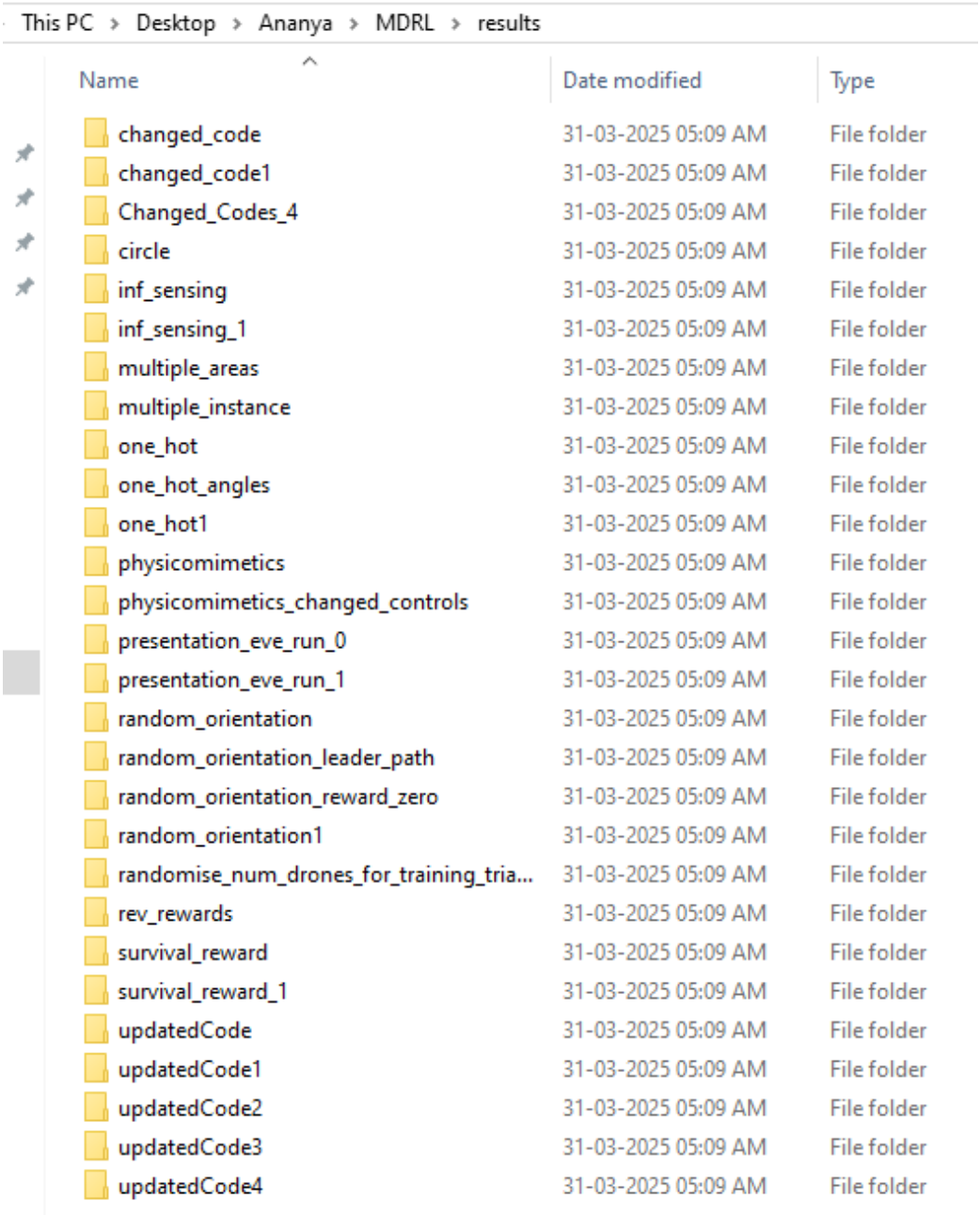
Fig. .4: Image 2: Python Environment Setup and CUDA Verification. This screenshot shows the output of `pip list` in the `mlagents_env` virtual environment, confirming that PyTorch with CUDA (version 2.6.0+cu118) is installed and CUDA is available.

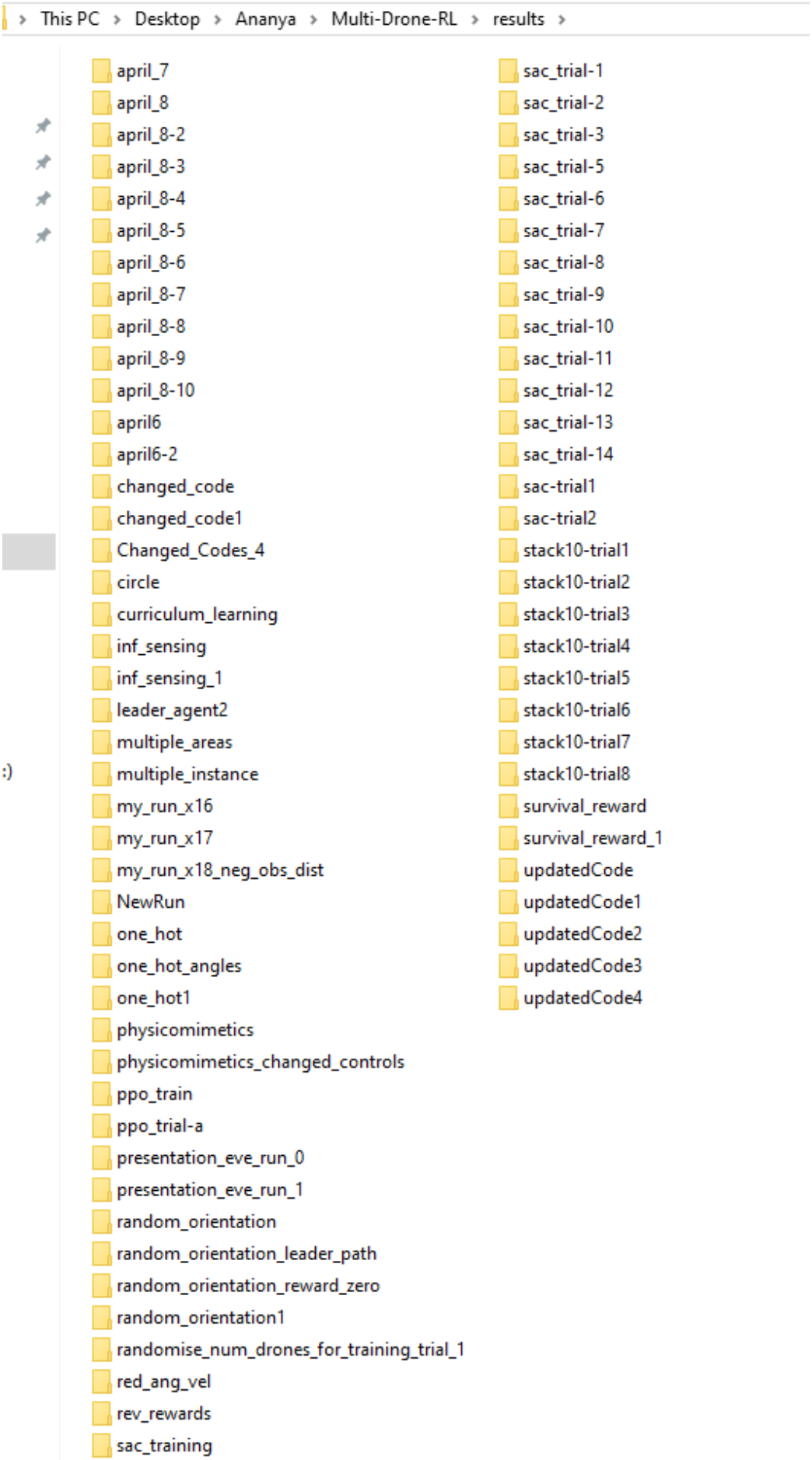
```
(mlagents_env) C:\Users\Kiwi>systeminfo

Host Name:                 DESKTOP-9T50SFB
OS Name:                   Microsoft Windows 10 Home
OS Version:                10.0.19045 N/A Build 19045
OS Manufacturer:           Microsoft Corporation
OS Configuration:          Standalone Workstation
OS Build Type:             Multiprocessor Free
Registered Owner:          Kiwi
Registered Organization:
Product ID:                00326-10000-00000-AA898
Original Install Date:     08-03-2025, 03:34:42
System Boot Time:          29-04-2025, 08:24:18
System Manufacturer:       Micro-Star International Co., Ltd.
System Model:              MS-7E26
System Type:               x64-based PC
Processor(s):              1 Processor(s) Installed.
                           [01]: AMD64 Family 25 Model 97 Stepping 2 AuthenticAMD ~4701 Mhz
BIOS Version:              American Megatrends International, LLC. 1.30, 10-08-2023
Windows Directory:         C:\Windows
System Directory:          C:\Windows\system32
Boot Device:               \Device\HarddiskVolume1
System Locale:             en-us;English (United States)
Input Locale:              00004009
Time Zone:                 (UTC+05:30) Chennai, Kolkata, Mumbai, New Delhi
Total Physical Memory:     31,905 MB
Available Physical Memory: 21,077 MB
Virtual Memory: Max Size:  79,009 MB
Virtual Memory: Available: 63,675 MB
Virtual Memory: In Use:    15,334 MB
Page File Location(s):     C:\pagefile.sys
Domain:                    WORKGROUP
Logon Server:              \\DESKTOP-9T50SFB
Hotfix(s):                 9 Hotfix(s) Installed.
                           [01]: KB5056578
                           [02]: KB5050579
                           [03]: KB5011048
                           [04]: KB5015684
                           [05]: KB5055518
                           [06]: KB5014032
                           [07]: KB5032907
                           [08]: KB5052916
                           [09]: KB5054682
Network Card(s):           3 NIC(s) Installed.
                           [01]: TP-LINK Gigabit Ethernet USB Adapter
                                 Connection Name: Ethernet 2
                                 Status:          Media disconnected
                           [02]: Realtek Gaming 2.5GbE Family Controller
                                 Connection Name: Ethernet
                                 Status:          Media disconnected
                           [03]: Realtek RTL8188EU Wireless LAN 802.11n USB 2.0 Network Adapter
                                 Connection Name: Wi-Fi
                                 DHCP Enabled:    Yes
                                 DHCP Server:     10.0.0.1
                                 IP address(es)
                                 [01]: 10.0.0.10
                                 [02]: fe80::6fd1:3b98:b322:bb7b
Hyper-V Requirements:      VM Monitor Mode Extensions: Yes
                           Virtualization Enabled In Firmware: Yes
                           Second Level Address Translation: Yes
                           Data Execution Prevention Available: Yes
```

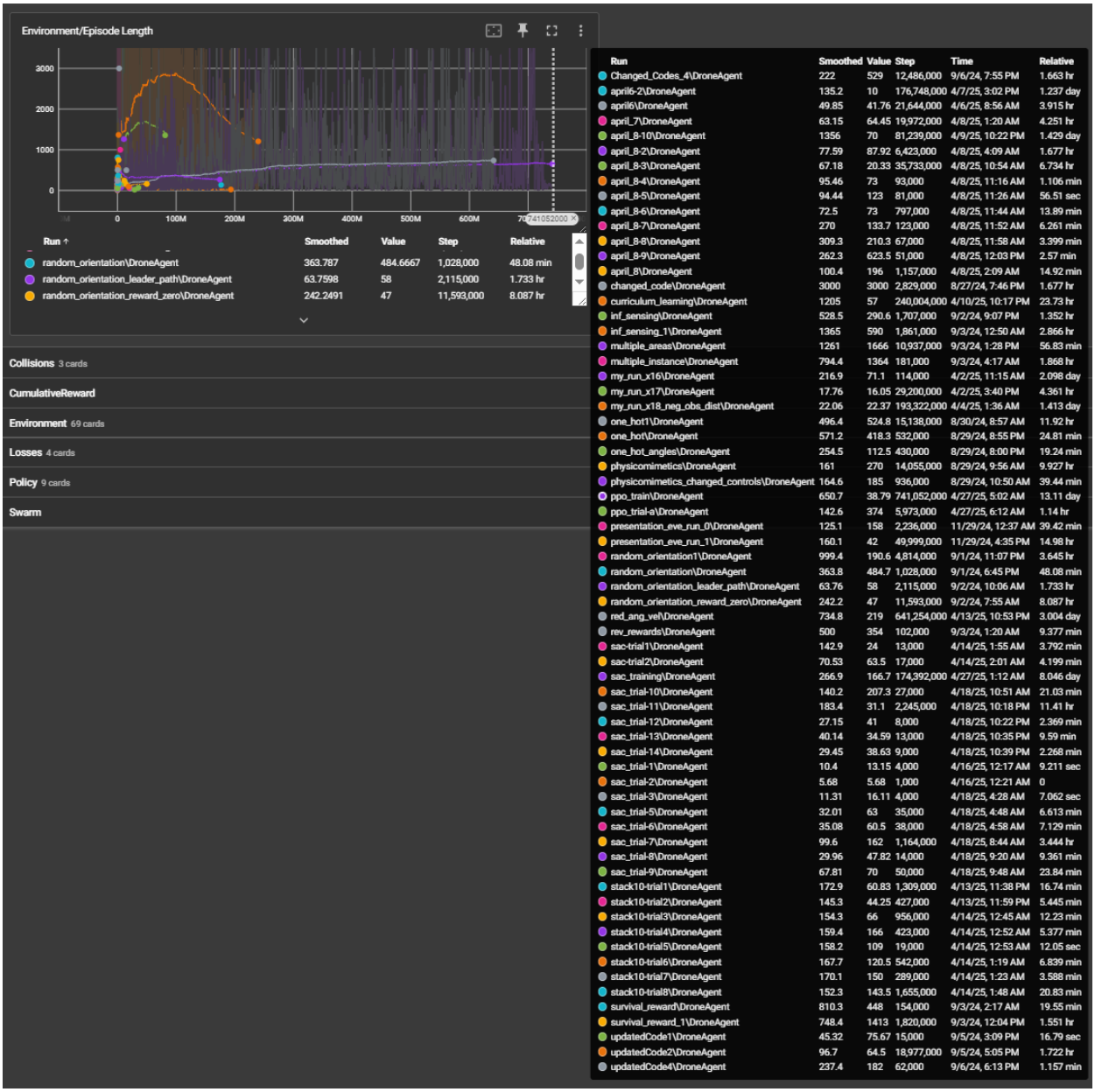
Fig. .5: Image 3: System Information Summary

This screenshot presents system specifications, including OS version, processor (AMD64 4701 MHz), installed RAM (32 GB), and Hyper-V support — essential for confirming compatibility with ML frameworks and GPU acceleration.

I.5 Appendix E: Bloopers (Failed (or not?) Trials)




This PC
Desktop
Ananya
Multi-Drone-RL
results
april_7
april_8
april_8-2
april_8-3
april_8-4
april_8-5
april_8-6
april_8-7
april_8-8
april_8-9
april_8-10
april6
april6-2
changed_code
changed_code1
Changed_Codes_4
circle
curriculum_learning
inf_sensing
inf_sensing_1
leader_agent2
multiple_areas
multiple_instance
my_run_x16
my_run_x17
my_run_x18_neg_obs_dist
NewRun
one_hot
one_hot_angles
one_hot1
physicomimetics
physicomimetics_changed_controls
ppo_train
ppo_trial-a
presentation_eve_run_0
presentation_eve_run_1
random_orientation
random_orientation_leader_path
random_orientation_reward_zero
random_orientation1
randomise_num_drones_for_training_trial_1
red_ang_vel
rev_rewards
sac_training
sac_trial-1
sac_trial-2
sac_trial-3
sac_trial-5
sac_trial-6
sac_trial-7
sac_trial-8
sac_trial-9
sac_trial-10
sac_trial-11
sac_trial-12
sac_trial-13
sac_trial-14
sac-trial1
sac-trial2
stack10-trial1
stack10-trial2
stack10-trial3
stack10-trial4
stack10-trial5
stack10-trial6
stack10-trial7
stack10-trial8
survival_reward
survival_reward_1
updatedCode
updatedCode1
updatedCode2
updatedCode3
updatedCode4


Environment/Episode Length
3000
2000
1000
0
0
100M
200M
300M
400M
500M
600M
741052000
Run
Smoothed
Value
Step
Relative
random_orientation\DroneAgent 363.787 484.6667 1,028,000 48.08 min
random_orientation_leader_path\DroneAgent 63.7598 58 2,115,000 1.733 hr
random_orientation_reward_zero\DroneAgent 242.2491 47 11,593,000 8.087 hr
Collisions 3 cards
CumulativeReward
Environment 69 cards
Losses 4 cards
Policy 9 cards
Swarm
Run Smoothed Value Step Time Relative
Changed_Codes_4\DroneAgent 222 529 12,486,000 9/6/24, 7:55 PM 1.663 hr
april6-2\DroneAgent 135.2 10 176,748,000 4/7/25, 3:02 PM 1.237 day
april6\DroneAgent 49.85 41.76 21,644,000 4/6/25, 8:56 AM 3.915 hr
april_7\DroneAgent 63.15 64.45 19,972,000 4/8/25, 1:20 AM 4.251 hr
april_8-10\DroneAgent 1356 70 81,239,000 4/9/25, 10:22 PM 1.429 day
april_8-2\DroneAgent 77.59 87.92 6,423,000 4/8/25, 4:09 AM 1.677 hr
april_8-3\DroneAgent 67.18 20.33 35,733,000 4/8/25, 10:54 AM 6.734 hr
april_8-4\DroneAgent 95.46 73 93,000 4/8/25, 11:16 AM 1.106 min
april_8-5\DroneAgent 94.44 123 81,000 4/8/25, 11:26 AM 56.51 sec
april_8-6\DroneAgent 72.5 73 797,000 4/8/25, 11:44 AM 13.89 min
april_8-7\DroneAgent 270 133.7 123,000 4/8/25, 11:52 AM 6.261 min
april_8-8\DroneAgent 309.3 210.3 67,000 4/8/25, 11:58 AM 3.399 min
april_8-9\DroneAgent 262.3 623.5 51,000 4/8/25, 12:03 PM 2.57 min
april_8\DroneAgent 100.4 196 1,157,000 4/8/25, 2:09 AM 14.92 min
changed_code\DroneAgent 3000 3000 2,829,000 8/27/24, 7:46 PM 1.677 hr
curriculum_learning\DroneAgent 1205 57 240,004,000 4/10/25, 10:17 PM 23.73 hr
inf_sensing\DroneAgent 528.5 290.6 1,707,000 9/2/24, 9:07 PM 1.352 hr
inf_sensing_1\DroneAgent 1365 590 1,861,000 9/3/24, 12:50 AM 2.866 hr
multiple_areas\DroneAgent 1261 1666 10,937,000 9/3/24, 1:28 PM 56.83 min
multiple_instance\DroneAgent 794.4 1364 181,000 9/3/24, 4:17 AM 1.868 hr
my_run_x16\DroneAgent 216.9 71.1 114,000 4/2/25, 11:15 AM 2.098 day
my_run_x17\DroneAgent 17.76 16.05 29,200,000 4/2/25, 3:40 PM 4.361 hr
my_run_x18_neg_obs_dist\DroneAgent 22.06 22.37 193,322,000 4/4/25, 1:36 AM 1.413 day
one_hot1\DroneAgent 496.4 524.8 15,138,000 8/30/24, 8:57 AM 11.92 hr
one_hot\DroneAgent 571.2 418.3 532,000 8/29/24, 8:55 PM 24.81 min
one_hot_angles\DroneAgent 254.5 112.5 430,000 8/29/24, 8:00 PM 19.24 min
physicomimetics\DroneAgent 161 270 14,055,000 8/29/24, 9:56 AM 9.927 hr
physicomimetics_changed_controls\DroneAgent 164.6 185 936,000 8/29/24, 10:50 AM 39.44 min
ppo_train\DroneAgent 650.7 38.79 741,052,000 4/27/25, 5:02 AM 13.11 day
ppo_trial-a\DroneAgent 142.6 374 5,973,000 4/27/25, 6:12 AM 1.14 hr
presentation_eve_run_0\DroneAgent 125.1 158 2,236,000 11/29/24, 12:37 AM 39.42 min
presentation_eve_run_1\DroneAgent 160.1 42 49,999,000 11/29/24, 4:35 PM 14.98 hr
random_orientation1\DroneAgent 999.4 190.6 4,814,000 9/1/24, 11:07 PM 3.645 hr
random_orientation\DroneAgent 363.8 484.7 1,028,000 9/1/24, 6:45 PM 48.08 min
random_orientation_leader_path\DroneAgent 63.76 58 2,115,000 9/2/24, 10:06 AM 1.733 hr
random_orientation_reward_zero\DroneAgent 242.2 47 11,593,000 9/2/24, 7:55 AM 8.087 hr
red_ang_vel\DroneAgent 734.8 219 641,254,000 4/13/25, 10:53 PM 3.004 day
rev_rewards\DroneAgent 500 354 102,000 9/3/24, 1:20 AM 9.377 min
sac-trial1\DroneAgent 142.9 24 13,000 4/14/25, 1:55 AM 3.792 min
sac-trial2\DroneAgent 70.53 63.5 17,000 4/14/25, 2:01 AM 4.199 min
sac_training\DroneAgent 266.9 166.7 174,392,000 4/27/25, 1:12 AM 8.046 day
sac_trial-10\DroneAgent 140.2 207.3 27,000 4/18/25, 10:51 AM 21.03 min
sac_trial-11\DroneAgent 183.4 31.1 2,245,000 4/18/25, 10:18 PM 11.41 hr
sac_trial-12\DroneAgent 27.15 41 8,000 4/18/25, 10:22 PM 2.369 min
sac_trial-13\DroneAgent 40.14 34.59 13,000 4/18/25, 10:35 PM 9.59 min
sac_trial-14\DroneAgent 29.45 38.63 9,000 4/18/25, 10:39 PM 2.268 min
sac_trial-1\DroneAgent 10.4 13.15 4,000 4/16/25, 12:17 AM 9.211 sec
sac_trial-2\DroneAgent 5.68 5.68 1,000 4/16/25, 12:21 AM 0
sac_trial-3\DroneAgent 11.31 16.11 4,000 4/18/25, 4:28 AM 7.062 sec
sac_trial-5\DroneAgent 32.01 63 35,000 4/18/25, 4:48 AM 6.613 min
sac_trial-6\DroneAgent 35.08 60.5 38,000 4/18/25, 4:58 AM 7.129 min
sac_trial-7\DroneAgent 99.6 162 1,164,000 4/18/25, 8:44 AM 3.444 hr
sac_trial-8\DroneAgent 29.96 47.82 14,000 4/18/25, 9:20 AM 9.361 min
sac_trial-9\DroneAgent 67.81 70 50,000 4/18/25, 9:48 AM 23.84 min
stack10-trial1\DroneAgent 172.9 60.83 1,309,000 4/13/25, 11:38 PM 16.74 min
stack10-trial2\DroneAgent 145.3 44.25 427,000 4/13/25, 11:59 PM 5.445 min
stack10-trial3\DroneAgent 154.3 66 956,000 4/14/25, 12:45 AM 12.23 min
stack10-trial4\DroneAgent 159.4 166 423,000 4/14/25, 12:52 AM 5.377 min
stack10-trial5\DroneAgent 158.2 109 19,000 4/14/25, 12:53 AM 12.05 sec
stack10-trial6\DroneAgent 167.7 120.5 542,000 4/14/25, 1:19 AM 6.839 min
stack10-trial7\DroneAgent 170.1 150 289,000 4/14/25, 1:23 AM 3.588 min
stack10-trial8\DroneAgent 152.3 143.5 1,655,000 4/14/25, 1:48 AM 20.83 min
survival_reward\DroneAgent 810.3 448 154,000 9/3/24, 2:17 AM 19.55 min
survival_reward_1\DroneAgent 748.4 1413 1,820,000 9/3/24, 12:04 PM 1.551 hr
updatedCode1\DroneAgent 45.32 75.67 15,000 9/5/24, 3:09 PM 16.79 sec
updatedCode2\DroneAgent 96.7 64.5 18,977,000 9/5/24, 5:05 PM 1.722 hr
updatedCode4\DroneAgent 237.4 182 62,000 9/6/24, 6:13 PM 1.157 min

BIBLIOGRAPHY

[1] Zain Anwar Ali, Eman H Alkhammash, and Raza Hasan. State-of-the-art flocking strategies for the collective motion of multi-robots. *Machines*, 12(10):739, October 2024.

[2] Rana Azzam, Igor Boiko, and Yahya Zweiri. Swarm cooperative navigation using centralized training and decentralized execution. *Drones*, 7:193, 03 2023.

[3] Iain D Couzin, Jens Krause, Nigel R Franks, and Simon A Levin. Effective leadership and decision-making in animal groups on the move. *Nature*, 433(7025):513–516, 2005.

[4] Marco Dorigo, Guy Theraulaz, and Vito Trianni. Reflections on the future of swarm robotics. *Science Robotics*, 5(49):eabe4385, 2020.

[5] Jakob Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients, 2024.

[6] Maximilian Hüttenrauch, Adrian Šošić, and Gerhard Neumann. Deep reinforcement learning for swarm systems, 2019.

[7] Tanja Katharina Kaiser. *The Predicting Swarm: Evolving Collective Behaviors for Robot Swarms by Minimizing Surprise*. PhD thesis, Universität zu Lübeck, 2022.

[8] Landon Kraemer and Bikramjit Banerjee. Multi-agent reinforcement learning as a rehearsal for decentralized planning. *Neurocomputing*, 190:82–94, May 2016.

[9] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments, 2020.

[10] BL Partridge and TJ Pitcher. The use of lateral line information in the schooling behaviour of fish. *Animal Behaviour*, 28(3):679–686, 1980.

[11] M L Puterman. *Markov decision processes*. Wiley Series in Probability & Mathematical Statistics: Applied Probability & Statistics. John Wiley & Sons, Nashville, TN, May 1994.

[12] Wei Ren and Randal W Beard. *Distributed consensus in multi-vehicle cooperative control: theory and applications*. Springer, 2008.

[13] Craig W Reynolds. Flocks, herds and schools: A distributed behavioral model. In *ACM SIGGRAPH Computer Graphics*, volume 21, pages 25–34. ACM, 1987.

[14] Gal Sajko and Jan Babič. *Achieving Human-Inspired Drift Diffusion Consensus in Swarm Robotics*, pages 29–41. XYZ, 09 2024.

[15] Gábor Vásárhelyi, Csaba Virágh, Gergő Somorjai, Tamás Nepusz, Agoston E. Eiben, and Tamás Vicsek. Optimized flocking of autonomous drones in confined environments. *Science Robotics*, 3(20):eaat3536, 2018.