UNIVERSITY OF THESSALY

SCHOOL OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Dynamic Maneuvering of Swarming UAVs Through Neural Network-Driven Control

Diploma Thesis

Ioannis Kalamakis

Supervisor: Dimitrios Katsaros

September 2025

UNIVERSITY OF THESSALY

SCHOOL OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Dynamic Maneuvering of Swarming UAVs Through Neural Network-Driven Control

Diploma Thesis

Ioannis Kalamakis

Supervisor: Dimitrios Katsaros

September 2025

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Δυναμικοί Ελιγμοί Σμηνών UAV Μέσω Ελέγχου Βασισμένου σε Νευρωνικά Δίκτυα

Διπλωματική Εργασία

Ιωάννης Καλαμάκης

Επιβλέπων/πουσα: Δημήτριος Κατσαρός

Σεπτέμβριος 2025

Approved by the Examination Committee:

Supervisor **Dimitrios Katsaros**
Associate Professor, Department of Electrical and Computer Engineering, University of Thessaly

Member **Rafailidis Dimitrios**
Associate Professor, Department of Electrical and Computer Engineering, University of Thessaly

Member **Flegkas Parisis**
Assistant Professor, Department of Electrical and Computer Engineering, University of Thessaly

DISCLAIMER ON ACADEMIC ETHICS AND INTELLECTUAL PROPERTY RIGHTS

«Being fully aware of the implications of copyright laws, I expressly state that this diploma thesis, as well as the electronic files and source codes developed or modified in the course of this thesis, are solely the product of my personal work and do not infringe any rights of intellectual property, personality and personal data of third parties, do not contain work / contributions of third parties for which the permission of the authors / beneficiaries is required and are not a product of partial or complete plagiarism, while the sources used are limited to the bibliographic references only and meet the rules of scientific citing. The points where I have used ideas, text, files and / or sources of other authors are clearly mentioned in the text with the appropriate citation and the relevant complete reference is included in the bibliographic references section. I also declare that the results of the work have not been used to obtain another degree. I fully, individually and personally undertake all legal and administrative consequences that may arise in the event that it is proven, in the course of time, that this thesis or part of it does not belong to me because it is a product of plagiarism».

The declarant

Ioannis Kalamakis

Diploma Thesis

Dynamic Maneuvering of Swarming UAVs Through Neural Network-Driven Control

Ioannis Kalamakis

Abstract

Unmanned Aerial Vehicles (UAVs) are increasingly deployed across defense, humanitarian aid, and search-and-rescue missions, yet coordinating many vehicles to cover dynamic tasks efficiently remains challenging. This thesis addresses that gap by designing, implementing, and evaluating a **centralized, reinforcement learning (RL) controller** for a swarm of multirotor UAVs.

We train a single policy with Proximal Policy Optimization (PPO) that consumes a joint observation of the swarm and outputs a joint action vector partitioned to per-drone commands. Training and evaluation are conducted in a custom environment built on `gym-pybullet-drones` (PyBullet-based), extended with (i) a shared waypoint pool constrained to 3D workspace bounds, (ii) waypoint life-cycle and ownership management, and (iii) two task-assignment strategies: a simple sequential rule and a global, distance-optimal matching via the Hungarian method.

Across compact and large workspaces and for increasing task loads, distance-aware assignment (Hungarian) consistently outperforms sequential ordering, yielding shorter total flight distance and mission time, higher completion rates, and fewer reassignments. The gains widen as task density increases, indicating that global, geometry-aware task allocation combined with a centralized PPO controller scales effectively for multi-UAV coverage.

Keywords:
multi-UAV coordination; reinforcement learning; PPO; task allocation; Hungarian algorithm; PyBullet; waypoint coverage.

Διπλωματική Εργασία

Δυναμικοί Ελιγμοί Σμηνών UAV Μέσω Ελέγχου Βασισμένου σε Νευρωνικά Δίκτυα

Ιωάννης Καλαμάκης

Περίληψη

Τα Μη Επανδρωμένα Αεροχήματα (UAVs) αξιοποιούνται ολοένα και περισσότερο σε αμυντικές επιχειρήσεις, ανθρωπιστικές βοήθειες και αποστολές έρευνας και διάσωσης, ωστόσο ο αποτελεσματικός συντονισμός πολλών οχημάτων για την κάλυψη δυναμικών εργασιών παραμένει πρόκληση. Η παρούσα εργασία αντιμετωπίζει αυτό το κενό με τον σχεδιασμό, την υλοποίηση και την αξιολόγηση ενός *κεντρικοποιημένου χειριστή ενισχυτικής μάθησης (RL)* για σμήνος πολυρότορων UAVs.

Εκπαιδεύουμε μία ενιαία πολιτική με Proximal Policy Optimization (PPO), η οποία λαμβάνει μια κοινή παρατήρηση του σμήνους και παράγει ένα ενιαίο διάνυσμα ενεργειών που κατανέμεται σε επιμέρους εντολές ανά UAV. Η εκπαίδευση και η αξιολόγηση πραγματοποιούνται σε προσαρμοσμένο περιβάλλον βασισμένο στο `gym-pybullet-drones` (PyBullet), το οποίο επεκτείνουμε με: (i) κοινή «δεξαμενή» σημείων πορείας (waypoints) περιορισμένη εντός τρισδιάστατων ορίων, (ii) διαχείριση κύκλου ζωής και «ιδιοκτησίας» των waypoints, και (iii) δύο στρατηγικές ανάθεσης εργασιών: μια απλή σειριακή μέθοδο και μια παγκόσμια, βέλτιστη ως προς την απόσταση αντιστοίχιση μέσω της μεθόδου Hungarian.

Σε συμπαγείς και εκτεταμένους χώρους εργασίας και για αυξανόμενα φορτία εργασιών, η ανάθεση με επίγνωση απόστασης (Hungarian) υπερέχει σταθερά της σειριακής: επιτυγχάνει μικρότερη συνολική απόσταση πτήσης και χρόνο αποστολής, υψηλότερα ποσοστά ολοκλήρωσης και λιγότερες επανααναθέσεις. Τα κέρδη εντείνονται με την πυκνότητα εργασιών, υποδεικνύοντας ότι η παγκόσμια, γεωμετρικά ενημερωμένη κατανομή εργασιών σε συνδυασμό με έναν κεντρικοποιημένο ελεγκτή PPO κλιμακώνεται αποτελεσματικά για πολυ-UAV κάλυψη.

Λέξεις-κλειδιά:
συντονισμός πολλαπλών UAVs· ενισχυτική μάθηση· PPO· κατανομή εργασιών· αλγόριθμος Hungarian· PyBullet· κάλυψη σημείων πορείας (waypoints).

Table of contents

Abstract x

Περίληψη xi

Table of contents xiii

List of figures xvii

List of tables xix

1 Introduction 1

1.1 Objective of this Thesis . . . 2

1.1.1 Contribution . . . 3

1.2 Organization of the Thesis . . . 3

2 General background 5

2.1 Machine Learning . . . 5

2.1.1 Supervised Learning . . . 6

2.1.2 Unsupervised Learning . . . 7

2.2 Deep Learning . . . 8

2.2.1 Artificial Neural Networks and Depth . . . 8

2.2.2 Training: Activation Functions and Optimization . . . 9

2.2.3 Regularization and Generalization . . . 10

2.2.4 Other Architectures: CNNs, RNNs, LSTMs . . . 10

2.3 Reinforcement Learning . . . 11

2.3.1 Fundamentals of Reinforcement Learning . . . 12

2.3.2 Value-Based Methods . . . 14

2.3.3 Model-Based Methods 18
2.3.4 Advanced Deep Reinforcement Learning Algorithms 19
2.3.5 Remarks for this thesis. 19

3 Related Work **21**

4 Problem and Proposed Solution **23**
4.1 Scenarios 23
4.1.1 Scenario 1: Defending a base under attack 23
4.1.2 Scenario 2: Humanitarian Delivery in Blocked Zones. 24
4.1.3 Scenario 3: Search and Rescue in Hazardous Environments 24
4.2 Proposed Solution 25
4.2.1 From targets to waypoints 25
4.2.2 Objective formulations and routing criteria 25
4.2.3 Path planning and execution 26

5 Experimental Setup & Evaluation **27**
5.1 Frameworks Used 27
5.1.1 Learning To Fly (gym-pybullet-drones) 27
5.1.2 Bullet Physics / PyBullet 27
5.1.3 Stable Baselines3 28
5.2 The Custom Environment: Marl_dynamic_waypoints 29
5.2.1 Features of the custom environment 29
5.3 Training 31
5.4 Metrics & Evaluation 35
5.4.1 Metrics 35
5.4.2 Evaluation Setup 35

6 Conclusion and Future Work **39**
6.1 Conclusions 39
6.2 Future Work 40

Bibliography **43**

APPENDICES **53**

A Environment Initialization Parameters **55**

B Observation Specification **59**

C Reward Function **61**

D Training Hyperparameters **63**

List of figures

2.1 Relationship among AI, ML, DL, and RL. Reinforcement Learning is a sub-field of ML; its overlap with DL corresponds to Deep RL[1]. 6
2.2 Supervised vs. Unsupervised learning. 8

5.1 PyBullet environment initial configuration: agents and waypoints placed within a small workspace 28
5.2 Sequential vs. Hungarian assignment schematic (illustrative). 30
5.3 Reward progression during training with 6 drones and 50 waypoints in a small workspace, using distance-based assignment. 32
5.4 Reward statistics during training with 6 drones and 50 waypoints in a small workspace, using distance-based assignment. 33
5.5 Waypoint metrics during training with 6 drones and 50 waypoints in a small workspace, using distance-based assignment. 34

List of tables

2.1 Conceptual comparison of Dynamic Programming (DP), Monte Carlo (MC), and Temporal-Difference (TD) learning. 13
2.2 Conceptual comparison among value-based methods. 15
2.3 Conceptual contrasts among core policy gradient methods included in this thesis. 17
2.4 Core advanced deep RL algorithms with their policy types and advantages. 20

5.1 Mean metrics of 10 runs per waypoint count and strategy for the small bound. 36
5.2 Mean metrics of 10 runs per waypoint count and strategy for the large bound. 37

C.1 Per-term reward values. 62

D.1 Hardware and training configuration. 63
D.2 PPO hyperparameters (defaults close to widely used SB3 practice). 64

Chapter 1

Introduction

Unmanned Aerial Vehicles (UAVs) have rapidly evolved from niche tools to versatile platforms used in inspection, mapping, emergency response, precision agriculture, and environmental monitoring. Their ability to quickly traverse large or hazardous areas, gather data, and act with fine-grained agility has made them a cornerstone of modern field robotics. As operational requirements expand—from simple data collection to time-critical missions—there is growing demand for **learning-based control** that adapts to uncertainty, disturbances, and task variability in real time.

Reinforcement Learning (RL), when combined with deep neural networks, enables UAVs to learn control policies directly from interaction, rather than relying solely on hand-tuned controllers. In single-UAV settings, Deep RL (DRL) has shown the capacity to stabilize attitude, track trajectories, and navigate toward goals under noise and partial observability. These learning-based policies can complement traditional control stacks by injecting adaptive behaviors in situations where analytic models are incomplete or where environment conditions vary.

Scaling from one UAV to many introduces additional coordination demands: agents must **divide work, avoid conflicts, and maintain coverage** as tasks and goals evolve. A practical path—followed in this thesis—uses a centralized multi-agent controller trained with PPO: a single policy network observes the combined state of all UAVs and produces coordinated actions for each of them. Training and execution are both centralized, so every decision is based on the global view of the swarm. Coordination emerges implicitly from this shared control signal and from the task-level waypoint-assignment mechanism, without requiring explicit communication among drones. This approach balances learning-based adaptability

with simple, robust task assignment rules.

1.1 Objective of this Thesis

The objective of this thesis, entitled **Dynamic Maneuvering of Swarming UAVs Through Neural Network-Driven Control**, is to design, implement, and evaluate a reinforcement learning framework for the coordinated control of multiple UAVs in dynamic waypoint coverage tasks.

One of the challenges addressed in this work is the **coordination of multiple UAVs** in coverage and navigation tasks. Conventional methods often rely on explicit inter-drone communication or rigid, pre-programmed strategies, which tend to be inefficient and difficult to scale to larger teams. In this thesis, coordination is achieved through a **centralized deep reinforcement learning controller trained with PPO**. A single policy network processes the joint observation of all UAVs and produces a combined action vector, which is then divided into individual commands for each drone. This design centralizes both training and execution, enabling adaptive and scalable coordination to emerge directly from shared global state information and task-level waypoint assignment, without requiring explicit communication protocols.

The second challenge is the **assignment of dynamic waypoints** to multiple UAVs in real time. A key challenge lies in distributing targets fairly and efficiently, while avoiding conflicts or idle drones. To address this, the environment developed in this thesis implements two assignment strategies: a sequential allocation method and an optimal distance-minimizing assignment using the Hungarian algorithm. These strategies provide flexible task allocation mechanisms that complement the centralized joint-policy control.

The third challenge concerns the **systematic evaluation of swarm behavior** within a realistic simulation environment. Achieving coordinated maneuvers in multi-UAV swarms is complicated by the non-stationary nature of agent interactions, the partial observability of the environment, and the demands of continuous control in large action spaces. To address these issues, a custom simulation platform was developed using **PyBullet and Gymnasium**, offering accurate physics, flexible swarm configurations, and integrated visualization. The evaluation framework incorporates detailed performance metrics—such as episodic rewards, waypoint completions per agent,failed attempts and overall success rates—enabling a com-

prehensive analysis of how neural network–based policies scale and adapt in multi-agent scenarios.

1.1.1 Contribution

The contribution of this thesis can be summarized as follows:

1. **Design of a custom UAV swarm environment:** A physics-based simulation environment was developed using PyBullet and Gymnasium, supporting multi-agent UAV navigation with dynamic waypoint generation, customizable swarm sizes, and visualization tools.

2. **Implementation of centralized deep reinforcement learning control:** A single PPO policy network was trained to process the combined state of all UAVs and to output an action for each drone (i.e., a joint action vector that is partitioned into per-UAV commands). Both training and execution are centralized, and coordination emerges from the shared global representation and the waypoint-assignment mechanism.

3. **Development of waypoint assignment strategies:** Multiple allocation mechanisms were integrated into the environment, including sequential assignment and an optimal distance-minimizing assignment based on the Hungarian algorithm, enabling systematic comparison of task distribution policies.

4. **Evaluation of neural network-driven swarm controller:** The proposed system was tested under different waypoint configurations. Metrics included waypoint completions, failure counts, total distance traveled, and mission completion time, providing a detailed assessment of learning progress and swarm coordination performance.

5. **Provision of a reproducible experimentation framework:** The entire pipeline—including environment, algorithms, training scripts, and visualization tools—was integrated into a modular framework, enabling future extensions and comparative studies in UAV swarm control.

1.2 Organization of the Thesis

- Chapter 2 provides the general background and theoretical foundations of this thesis.

- Chapter 3 reviews related work in the field of UAV swarm control and reinforcement learning.
- Chapter 4 introduces different scenarios for UAV swarm operations.
- Chapter 5 details the experimental setup, training procedures, and evaluation metrics used to assess the proposed approach.
- Chapter 6 concludes with a summary of contributions, discussion of limitations, and suggestions for future research directions.

Chapter 2

General background

2.1 Machine Learning

Machine Learning (ML) is a subfield of Artificial Intelligence (AI) concerned with algorithms that learn from data to improve their performance on a task without being explicitly programmed [2, 3]. The output of a learning algorithm can be expressed as a mapping

$$y(x) : \mathbb{R}^d \to \mathbb{R}^k, \qquad x \mapsto y(x),$$

where x is a new input vector and $y(x)$ the corresponding predicted output; the precise form of $y(x)$ is determined during a training (learning) phase using data [4].

In recent years, ML performance—especially on high-dimensional data such as time series, images, and video—has improved dramatically due to advances in Deep Learning (DL) [5, 6]. These gains are linked to (i) increased computational power and GPU adoption [7], (ii) methodological breakthroughs including regularization and optimization techniques such as Dropout, Batch Normalization, and residual learning [8, 9, 10], and (iii) the development of software ecosystems such as TensorFlow and PyTorch [11, 12].

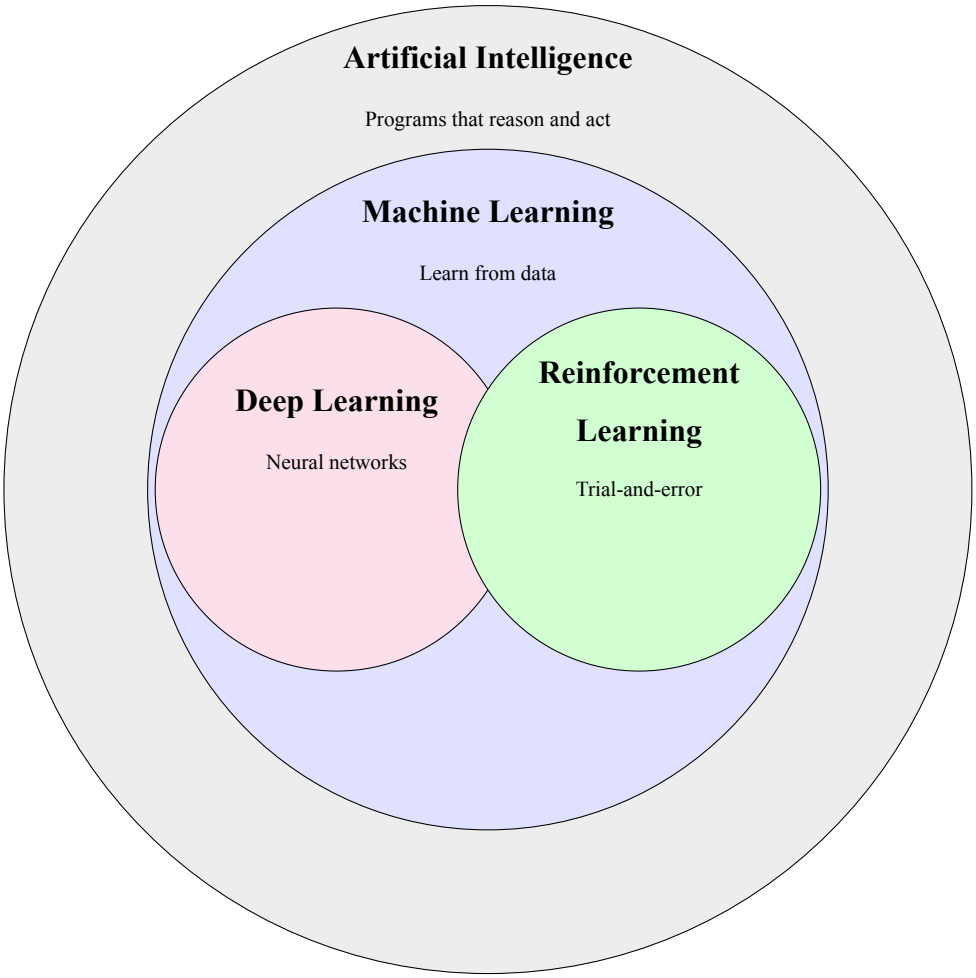

Figure 2.1: Relationship among AI, ML, DL, and RL. Reinforcement Learning is a subfield of ML; its overlap with DL corresponds to Deep RL[1].

2.1.1 Supervised Learning

Supervised learning is a paradigm in which an algorithm is trained on labeled data consisting of input–output pairs $[X, Y]$. The goal is to approximate an unknown function $f : X \mapsto Y$ that maps an input vector $x \in X$ to the corresponding output $y \in Y$:

$$y = f(x).$$

More formally, a supervised learning algorithm maps a dataset $D_{LS} = \{(x_i, y_i)\}_{i=1}^{N}$ into a predictive model by optimizing parameters θ to minimize an empirical risk over the training set:

$$I_S[f] = \frac{1}{N} \sum_{i=1}^{N} L(y_i, f_\theta(\mathbf{x}_i)),$$

where $L(\cdot)$ measures the discrepancy between the true label and the model prediction [4, 3, 13].

Tasks in supervised learning are commonly divided into two categories:

- **Classification**, where the output space consists of discrete labels (e.g., predicting whether an email is spam, or recognizing whether an image depicts a cat, a dog, or a car).
- **Regression**, where the objective is to predict continuous values (e.g., estimating a house price given features, or forecasting tomorrow's temperature from historical data).

A central challenge is **generalization**: models must perform well on unseen data. Issues such as overfitting and data imbalance require careful model selection, regularization, and evaluation [6, 13]. Supervised learning serves as the foundation for diverse applications, spanning computer vision (pattern and object recognition), speech and language processing, medical diagnostics, cheminformatics, information retrieval, and targeted marketing [13, 3].

2.1.2 Unsupervised Learning

Unsupervised learning considers datasets of inputs $\{x_1, x_2, \dots, x_N\}$ without associated labels, aiming to uncover latent structure such as clusters or low-dimensional representations. Formally, one objective is to model the underlying data distribution $p(x)$ or to learn a representation $\phi(\cdot)$ that captures the essential structure of the data [3, 6]. Two widely studied tasks are:

- **Clustering:** the task of grouping samples into k clusters $\{C_1, \dots, C_k\}$ such that points within the same cluster are more similar to each other than to those in different clusters. A classical approach is k-means, which partitions data by minimizing within-cluster variance [14, 13].
- **Dimensionality Reduction:** techniques that project data into a lower-dimensional subspace while preserving key variance or structural properties. A canonical method is Principal Component Analysis (PCA) [15].

Unsupervised methods have broad applications, including market segmentation, document organization and topic modeling, anomaly and fraud detection, biological data analysis, and serving as preprocessing for visualization or subsequent learning tasks [3]. However, they present unique challenges: the lack of ground-truth labels makes evaluation difficult (often relying on internal indices such as the silhouette score), and the high dimensionality

of modern datasets can degrade distance-based structure due to the **curse of dimensionality** [16, 17]

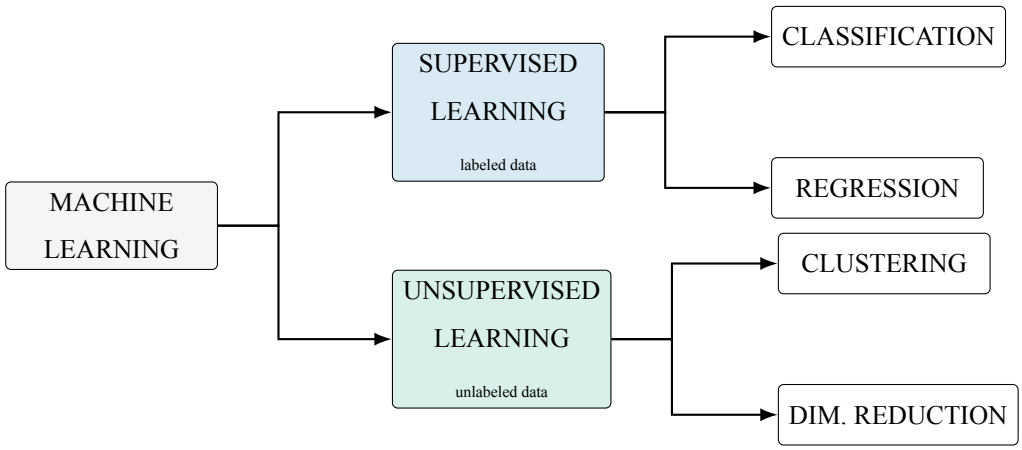

Figure 2.2: Supervised vs. Unsupervised learning.

2.2 Deep Learning

Deep Learning (DL) is a subfield of machine learning that learns hierarchical representations by composing many nonlinear transformations in deep neural networks [5, 6, 18]. By jointly learning both feature extractors and predictive mappings, DL reduces reliance on manual feature engineering and enables end-to-end modeling from raw inputs (e.g., images, signals, states) to outputs (e.g., labels, control actions) [19, 5]. Fueled by large datasets, increased compute, and effective optimization, deep models have achieved state-of-the-art performance across vision, language, speech, and control [5, 7]. At the same time, they are data and compute intensive and require careful regularization to generalize well [6]. In this thesis, deep neural networks serve as flexible function approximators for control policies within a reinforcement learning framework, enabling coordinated maneuvering in multi-UAV settings [1, 20].

2.2.1 Artificial Neural Networks and Depth

Artificial Neural Networks (ANNs) are computational models inspired by biological neural processing. An ANN comprises layers of interconnected units (**neurons**) that transform inputs into outputs through weighted connections and nonlinear activations [6, 4]. For a single neuron with inputs $\{x_i\}_{i=1}^{n}$, weights $\{w_i\}_{i=1}^{n}$, and bias b, the pre-activation is defined as

the weighted sum of inputs plus bias, and the activation as the nonlinear function applied to it:

$$z = \sum_{i=1}^{n} w_i x_i + b, \qquad a = f(z). \tag{2.1}$$

Here, $f(\cdot)$ denotes a nonlinear activation (e.g., sigmoid, tanh, ReLU). Stacking such units yields feedforward networks capable of approximating highly nonlinear mappings; under mild assumptions, multilayer feedforward networks are universal function approximators [21, 22].

Depth in Neural Networks

A **Deep Neural Network (DNN)** is simply an ANN with **multiple hidden layers**. By composing multiple layers–each combining an affine transformation (linear mapping plus bias) with a nonlinear activation–deeper models learn **hierarchical representations** in which higher layers capture increasingly abstract features, enhancing their expressivity on complex tasks [5, 6]. Depth (number of hidden layers) and width (units per layer) together determine capacity; in practice, modern deep networks may include conveniences like normalization or skip connections, but the core idea remains layered composition.

2.2.2 Training: Activation Functions and Optimization

Training adjusts the weights and biases of a neural network to minimize a loss function L over data, typically using gradient-based learning (**backpropagation**) [23]. In gradient descent, a parameter w is updated according to

$$w \leftarrow w - \eta \frac{\partial L}{\partial w}, \tag{2.2}$$

where η denotes the learning rate. Modern optimizers such as Adam adapt learning rates on a per-parameter basis and often accelerate convergence [24].

Nonlinear activation functions are essential for preventing the network from collapsing into a purely linear model and for enhancing representational power. Common choices include

$$\text{sigmoid: } f(z) = \frac{1}{1+e^{-z}}, \qquad \text{tanh: } f(z) = \tanh(z), \qquad \text{ReLU: } f(z) = \max(0, z),$$

with rectified linear units (ReLU) widely adopted in deep architectures due to their favorable optimization behavior [25].

2.2.3 Regularization and Generalization

A central challenge in deep learning is **overfitting**, where a model fits the training data too closely and generalizes poorly to unseen inputs. Regularization aims to improve generalization through:

- **Weight decay (L2):** penalizing large weights by augmenting the objective with $\lambda\|W\|_2^2$ [26, 27],
- **Dropout:** randomly zeroing hidden units during training to reduce co-adaptation and prevent overfitting [8],
- **Batch Normalization:** normalizing intermediate activations to stabilize and accelerate training [9],
- **Early stopping:** halting training once validation performance ceases to improve [28].

Appropriate combinations of these methods are routinely used in modern deep learning systems [6]. Nevertheless, recent work shows that even heavily over-parameterized networks can generalize well without strong explicit regularization, motivating continued investigation into the underlying mechanisms of generalization [29]

2.2.4 Other Architectures: CNNs, RNNs, LSTMs

While this thesis employs fully connected feedforward networks as policy function approximators, other neural architectures are central in deep learning. **Convolutional Neural Networks (CNNs)** are designed for spatially structured data, achieving state-of-the-art performance in vision tasks by leveraging local connectivity, parameter sharing, and hierarchical feature extraction [30]. **Recurrent Neural Networks (RNNs)** introduce hidden-state recurrence to capture temporal dependencies in sequential data [31]. **Long Short-Term Memory (LSTM)** networks extend RNNs by incorporating gated memory cells that mitigate vanishing gradients and enable learning of long-range dependencies [32]. These families of models extend deep learning to domains such as images, speech, language, and time series. In the present work, a simpler feedforward ANN was deemed sufficient for representing the control mappings under study.

2.3 Reinforcement Learning

Reinforcement Learning (RL) studies how an **agent** interacts with an **environment** to achieve goals through interaction and feedback. At each decision step, the agent receives information from the environment (e.g., sensor readings, proprioceptive signals, or a processed state estimate), chooses an **action** according to its current **policy** (its control rule), and then the environment evolves and returns a scalar **reward** that reflects immediate progress toward the task objective [20, 1].

The environment. The environment comprises everything external to the policy's internal computation. In robotics, this typically encompasses the robot's body and dynamics (e.g., rigid bodies, joints, contact), actuators and their limits (motors, propellers), sensors (IMU, GPS, cameras, encoders), the task setup (track, obstacles, tools), and the surrounding world (airflow, terrain). The **agent** holds the decision rule (policy) that maps the available information to control commands.

Actions. Actions are the control inputs the agent can issue. In practice, these may be discrete (e.g., $\{-1, 0, 1\}$ thrust increments; gear selections) or continuous (e.g., voltages, PWM duty cycles, motor torques, rotor thrusts, joint velocities). In practice, actions are typically bounded by hardware limits and safety envelopes; the policy outputs are thus clipped or squashed (e.g., via tanh) to admissible ranges.

Observations. Observations or **states**, when a full state estimate is available. Can include kinematic variables (positions, attitudes, velocities), actuator states, and exteroceptive measurements. The **reward** is a scalar signal designed to reflect task progress (e.g., negative tracking error, energy penalties, smoothness costs, task completion bonuses). Over time, the agent aims to choose actions that maximize cumulative reward, thereby improving performance through experience [20, 1].

2.3.1 Fundamentals of Reinforcement Learning

Finite Markov Decision Processes (MDP). Formalizes sequential decision-making with finite state and action sets $\mathcal{S}$ and $\mathcal{A}$.It is defined by:

- a transition probability function $T(s, a, s') = \Pr(S_{t+1} = s' \mid S_t = s, A_t = a)$,
- a reward function $R(s, a, s')$, and
- a discount factor $\gamma \in [0, 1)$.

The **Markov property** assumes the next-state distribution depends only on the current state and action, not on past history. A policy π maps states to actions, and the goal in reinforcement learning is to find a policy that maximizes the expected discounted return [1, 20].

Dynamic Programming. When the MDP's model (T, R) is known and state/action spaces are small enough, **Dynamic Programming** (DP) can compute optimal policies. Key algorithms include:

- **Policy evaluation**, which estimates the value $V^\pi(s)$ of a given policy π,
- **Policy improvement**, which modifies the policy to be greedy with respect to the current value estimates,
- **Policy iteration**, alternating evaluation and improvement until convergence, and
- **Value iteration**, which applies Bellman backups repeatedly until the value function converges.

A major limitation of DP methods is that they suffer from the **curse of dimensionality**: as $|\mathcal{S}|$ and $|\mathcal{A}|$ grow, both computational and memory requirements grow rapidly, making DP methods impractical for large-scale problems[1].

Monte Carlo Methods (MC) . Monte Carlo methods are model-free and rely purely on sampled experience. They do not require knowledge of T (transition probabilities) or R (expected rewards) in closed form. MC methods estimate value functions by averaging full returns over complete episodes. Because they **do not** bootstrap (i.e. their targets are not based on existing estimates of future values), these estimates are asymptotically unbiased. However, they tend to have high variance and need full episodes to be collected, which can slow learning in tasks with long horizons or without natural episodes [1].

Temporal-Difference (TD). Temporal-Difference methods merge ideas from Monte Carlo methods and Dynamic Programming. Like MC, they are model-free, but unlike MC, they *bootstrap* — they update value estimates by combining observed rewards with existing estimates of future values. For example, the TD(0) update is:

$$V(S_t) \leftarrow V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}) - V(S_t)).$$

TD methods typically enable more efficient, online learning with lower variance than Monte Carlo, although bootstrapping introduces bias [1].

Method	Requires model	Update style	Online?	Key trade-offs
DP	Yes	Bootstrapped full sweeps	No	Exact with known model; scales poorly; unsuitable for unknown environments.
MC	No	Episodic returns (non-bootstrapped)	No (updates after episodes)	Model-free and unbiased; very high variance; needs full episodes; best for episodic tasks; weaker for continuing control.
TD	No	Stepwise bootstrapped updates	Yes (step-by-step)	Model-free and online; bootstrapping lowers variance but adds bias; scalable to continuing tasks; backbone of modern RL.

Table 2.1: Conceptual comparison of Dynamic Programming (DP), Monte Carlo (MC), and Temporal-Difference (TD) learning.

2.3.2 Value-Based Methods

Value-based methods aim to learn an action–value function $Q(s, a)$ and act (approximately) greedily with respect to it. Below we summarize the distinctive ideas behind the main approaches relevant to this thesis and related work.

Q-learning (tabular, off-policy). A classical model-free control method that estimates the optimal value of state–action pairs while behaving ε-greedily. It is **off-policy**—the agent learns from exploratory behavior while updating toward the greedy target—and converges in the tabular case under standard conditions [33].

Fitted Q Iteration (FQI) / Neural Fitted Q (NFQ) (batch RL). A **batch** method: collect a dataset of transitions, compute Bellman targets, and **fit** a regressor to these targets, iteratively. FQI reduces value learning to supervised regression and can exploit strong non-parametric models (e.g., tree ensembles) [34]. NFQ applies the same principle with multilayer perceptrons, enabling data-efficient policy learning from limited interactions [35].

Deep Q-Network (DQN). Learns $Q_\theta(s, a)$ with a deep neural network and stabilizes training via two mechanisms: **experience replay** (to decorrelate and reuse samples) and a **target network** (to provide more stable targets). DQN famously achieved human-level performance on Atari directly from raw pixels [36].

Double DQN (reducing overestimation). In standard DQN, the max over next-state actions can overestimate values. **Double DQN** decouples action selection (online network) from evaluation (target network), reducing positive bias and improving stability [37].

Dueling Network Architecture (value + advantage). A single Q-network whose last layer splits into **value** $V(s)$ and **advantage** $A(s, a)$ streams, which are then combined to produce $Q(s, a)$. This decomposition helps the agent identify valuable states even when actions have similar values, improving learning efficiency [38].

Distributional DQN (learning the value distribution). Rather than predicting only the **expected** return, **distributional** reinforcement learning estimates the full **distribution** of returns $Z(s, a)$. In this framework, one models how returns are distributed (taking into account

Method	Key idea / specialty	Benefit	Typical setting
Q-learning	Off-policy tabular estimation of $Q^*(s, a)$	Converges in finite MDPs; low complexity.	Small discrete state and action spaces.
FQI / NFQ	Batch / offline regression of Bellman targets	Efficient when environment interactions are limited.	Logged data; simulators; limited online interaction.
DQN	Deep neural network + replay buffer + target network	Handles high-dimensional observations; stabilizes training.	Online, raw observations; discrete action spaces.
Double DQN	Separate action selection vs evaluation in TD target	Reduces overestimation bias; more stable learning.	Discrete actions; when DQN shows overestimation.
Dueling Architecture	Split into state-value and action-advantage streams	Better discrimination when actions have similar value.	Often used together with DQN-type methods.
Distributional Methods	Model full return distribution via atoms or quantiles	Richer learning signals; captures variance / risk; improved empirical stability.	Discrete actions; deep value-based agents; often combined with other enhancements.
Multi-step Returns	Use n-step instead of 1-step TD targets	Faster reward propagation; better sample efficiency.	Deep RL agents; episodic tasks; many modern value-based methods.

Table 2.2: Conceptual comparison among value-based methods.

randomness in rewards, transitions, and future actions) rather than collapsing all that uncertainty into a single mean. This provides richer training signals, helps capture variance and multimodality in possible outcomes, and often leads to better empirical performance compared to expectation-based methods [39].

Multi-step learning (n-step returns). Using n-step returns trades variance for faster reward propagation, often boosting learning speed and stability. Multi-step targets are now a standard component of deep value-based agents (e.g., Rainbow) [1, 40].

Policy Gradient Methods. Unlike value-based methods, which learn $Q(s, a)$ and then act (approximately) greedily, **policy gradient** (PG) methods optimize the policy directly. A neural policy is updated so that actions yielding higher long-term returns become more likely (in stochastic policies), or so that the output action improves (in deterministic ones). PG is particularly well-suited for continuous or high-dimensional action spaces, and for settings where stochastic exploration is beneficial [41, 42, 43].

Stochastic Policy Gradient. A **stochastic** policy $\pi_\theta(a \mid s)$ outputs a distribution over actions from which the agent samples. Learning proceeds by increasing the advantage-weighted log-likelihood of sampled actions. The REINFORCE algorithm is the canonical starting point, and variance reduction via baselines or learned critics is crucial in practice [42, 41, 44].

Deterministic Policy Gradient (DPG). A **deterministic** policy $\mu_\theta(s)$ outputs a single action in continuous action spaces. DPG avoids integrating over the action distribution in the gradient, making it more efficient especially in high-dimensional continuous control. Deep variants such as DDPG incorporate replay buffers and target networks for stability and learn off-policy from past experience [43, 45].

Actor–Critic Methods. **Actor–critic** methods combine a policy (the actor) with a value or advantage estimator (the critic) to produce low-variance, bootstrapped learning signals. Instead of using full returns, the critic provides estimates (e.g. via temporal-difference or n-step returns) that reduce variance relative to pure policy gradient methods. Modern variants include synchronous (e.g. A2C) and asynchronous (e.g. A3C) implementations, and both on-policy and off-policy forms are widely used (e.g. PPO, SAC, DDPG) [46, 41, 47].

Natural Policy Gradient. Standard policy gradients depend on the choice of parameterization and can lead to unstable updates. **Natural Policy Gradient (NPG)** preconditions updates with the Fisher information matrix, which defines a natural geometry on the policy space. This yields updates that are more stable, better scaled, and often more data-efficient [48].

Trust Region Policy Optimization. **TRPO** constrains policy updates within a trust region, typically enforced through a KL-divergence bound between old and new policies. This ap-

proach yields more stable and monotonic improvement under certain approximations and builds directly on the ideas of the natural policy gradient [49].

Method	Policy type	On-/ off-policy	Replay buffer	Trust region?	Key benefit / typical use
Stochastic PG (REINFORCE)	Stoch.	On-policy	No	No	Simple, general; works in discrete/continuous actions; needs variance reduction.
Deterministic PG / DDPG	Determ.	Off-policy	Yes	No	Efficient in continuous spaces; DDPG adds replay + target networks for stability.
Actor–Critic (A2C/A3C)	Stoch.	On-policy	No	No	Lower-variance PG via critic; scalable with parallel actors.
Natural PG	Stoch.	Either	Optional	No	Geometry-aware (Fisher-preconditioned) updates; more stable learning.
TRPO	Stoch.	On-policy	No	Yes	Monotonic policy improvement via KL trust region; more stable than vanilla PG.

Table 2.3: Conceptual contrasts among core policy gradient methods included in this thesis.

2.3.3 Model-Based Methods

Model-based RL (MBRL) learns an environment model (either explicit dynamics / reward or latent dynamics) and uses it for planning or generating synthetic data. Compared to model-free methods, MBRL can offer much greater sample efficiency, though it faces challenges like model bias (errors compounding over rollouts) and uncertainty in predictions

Representative approaches:

- **PETS:** Use an ensemble of probabilistic dynamics models and trajectory sampling (MPC) to select actions while accounting for uncertainty.
- **MBPO:** Generate short model rollouts from real states and mix synthetic and real transitions for training a policy or value learner.
- **Dreamer (latent world models):** Learn compact latent representations (often from pixel inputs), then imagine trajectories in latent space to improve policy/value via these imagined experiences.

Limitations and trade-offs.

Key challenges include: avoiding compounding model error (especially for long horizons), ensuring uncertainty in dynamics models is well captured (e.g. via probabilistic/ensemble methods), and balancing computation and planning vs real interaction.

Other SOTA examples .

MuZero learns a value – equivalent model (policy, value, reward predictors) and uses tree search planning rather than modeling full observations.

2.3.4 Advanced Deep Reinforcement Learning Algorithms

Building on trust-region or proximal policy gradient ideas, modern deep RL methods often combine entropy regularization, off-policy learning, and multiple value estimators to improve stability, exploration, and sample efficiency.

Proximal Policy Optimization (PPO). PPO uses a clipped surrogate objective to bound how far the updated policy can deviate from its predecessor, avoiding the computational overhead of constrained optimization per step. By performing multiple minibatch epochs per set of trajectories, PPO strikes a good balance between stability and empirical performance across many continuous control and robotics tasks [47].

Soft Actor–Critic (SAC). SAC maximizes a combination of expected return and policy entropy. It is off-policy, uses replay buffers and target networks, and tends to achieve excellent sample efficiency and robust exploration, especially in continuous control settings [50].

TD3 (Twin Delayed DDPG). TD3 improves on DDPG by introducing three key modifications: (1) two critic networks ("twin critics") whose minimum value is used to reduce overestimation bias, (2) delayed update of the policy relative to critics, and (3) target policy smoothing noise. These adjustments yield more stable learning in continuous action-space environments [51].

Other Recent Variants and Directions. There are several newer algorithms that build on SAC / TD3 to address specific weaknesses. For example, discrete action variants of SAC adapt its maximum entropy framework to non-continuous actions [52], and algorithms like SDSAC improve stability in discrete settings [53]. Methods like OPAC combine ideas from SAC and TD3 to improve exploration or reduce variance. These are interesting directions but typically more complex and less standard in many application settings.

2.3.5 Remarks for this thesis.

In our UAV control experiments, we adopt PPO as the primary algorithm, due to its simplicity, good performance under partial observability, and robustness to noise. We observe that TD3 and SAC are strong alternatives when sample efficiency or exploration in contin-

uous control is especially challenging. More complex variants (e.g. for discrete actions or hybrid methods) are potential future extensions but are beyond the current scope.

Algorithm	On-/Off-Policy	Clipping / Trust / Constraints	Entropy Regularization	Key Benefit / Typical Use
PPO	On-policy	Approximate trust region via clipping	Optional / often with entropy bonus	Stable updates; good empirical performance with fewer hyperparameter issues; works well in robotics / continuous control.
SAC	Off-policy	No explicit trust region constraint; uses target networks	Yes, core to objective (max-entropy)	Excellent sample efficiency; strong exploration; robust even in challenging continuous domains.
TD3	Off-policy	No KL constraint; adds delay and smoothing	No (usually minimal / implicit)	Reduces overestimation bias; more stable than DDPG; good performance in continuous control.
Other Variants	Mixed	Variants of constraints or tailored adaptations	Some include entropy / some don't	Address specialization (discrete actions, better exploration, variance reduction), but often more complex / sensitive.

Table 2.4: Core advanced deep RL algorithms with their policy types and advantages.

Chapter 3

Related Work

Deep learning has enabled UAV autonomy to go deep beyond perception to closed-loop control and navigation. Recent surveys document the breadth of learning-enabled UAV capabilities, including end-to-end sensing–to–control pipelines and embodied policies that execute fully onboard [54, 55, 56]. Demonstrations on agile single-UAV flight provide concrete evidence that neural policies can regulate aggressive motion under strict real-time constraints and transfer to hardware: vision-based racing systems map raw images to compact motion goals for high-speed gate traversal [57], sensorimotor policies perform acrobatic maneuvers with only onboard sensing and zero-shot sim-to-real deployment [58], and recent champion-level results show competitive lap times against human experts using learning-driven controllers [59].

In swarm settings, many studies impose a leader–follower structure, learning policies that control relative pose to both achieve and preserve formation. Obradović *et al.* train separate learned controllers for phase-specific behaviors (reach vs. maintain) using discrete motion primitives, validating both in simulation and on physical robots [60]. Liu *et al.* develop a collaborative controller that stabilizes leader–follower UAV formations under dynamic and uncertain conditions by adjusting relative position and heading from a compact group state [61]. Earlier multi-robot studies similarly frame formation keeping as a cooperative control problem with decentralized observations and homogeneous agents [62].

Moving beyond fixed roles, recent work targets **dynamic swarms** where coordination emerges from local interactions, often in cluttered environments or close proximity. Batra *et al.* learn decentralized policies that yield flocking, tight formations, obstacle avoidance, and goal swapping, with zero-shot transfer to resource-constrained quadrotors [63]. Follow-

up work introduces attention over neighbors and obstacles plus curriculum/replay strategies to scale collision-free navigation to dense settings and real hardware [64]. Complementing policy-learning, Neural-Swarm2 [65] augments nominal dynamics of heterogeneous multi-rotors with learned interaction residuals to capture aerodynamic effects (e.g., downwash), enabling safe close-proximity flight across heterogeneous vehicles and reducing tracking error. Task-coupled motion has also been explored: Chen *et al.* propose SCT-DRL, a value-based swarm controller that balances wide-area coverage with multi-target tracking via pairwise interaction simplifications and prior-experience initialization to stabilize training [66].

Despite this progress, gaps remain. Many systems either rely on decentralized execution with purely local observations—scalable but sometimes yielding inconsistent global coordination—or enforce leader–follower roles that limit adaptability. Interaction modeling that supports tight proximity and multi-objective motion is promising but not yet mainstream in fully dynamic swarms. Our approach explores the following regime: centralized training *and* centralized execution using one shared policy that, at every time step, looks at the combined state of all drones and then decides the action for each one, so the team moves in a globally consistent and tightly coordinated way.

Chapter 4

Problem and Proposed Solution

4.1 Scenarios

This thesis addresses the design, training, and deployment of a multi agent UAV swarm controlled by a **single policy** (a base station). Two operational modes are considered: (i) a **tethered command node**—a drone or mast linked to the base via a cable for power/backhaul—acting as a high-reliability relay; and (ii) a **standoff leader**—a stationary, safely positioned drone hosting the agent and issuing commands wirelessly to the rest of the swarm. The objective is a fast, reliable, and *scalable* control approach that keeps training time low while retaining robust performance in contested or GPS-denied environments [67, 68, 69, 70, 71, 72].

4.1.1 Scenario 1: Defending a base under attack

An operational base—or any protected facility—in a contested zone may face threats from immovable or semi-stationary adversary nodes capable of launching electronic warfare attacks, such as portable jamming devices or EMP-like systems. These threats can disrupt conventional radio-based command and GNSS navigation, rendering ordinary drones ineffective and leaving the base with little visibility of UAV status except unreliable motion traces.

This thesis proposes deploying a swarm of highly efficient, **fiber-optic tethered drones**. These drones are effectively immune to electromagnetic interference—since they rely on a physical fiber link rather than wireless signals [73]—allowing them to operate reliably even under intense jamming conditions. Recent battlefield developments, particularly from Ukraine and Russia, underscore the growing battlefield deployment of such fiber-optic drones

to maintain operational control under electronic assault [74, 75]. All swarm activity is orchestrated by a single intelligent agent at the base, sequencing target waypoints and directing neutralization operations with resilience to EW (electronic warfare) interference.

4.1.2 Scenario 2: Humanitarian Delivery in Blocked Zones.

In the aftermath of a disaster—whether triggered by natural forces such as earthquakes, floods, or hurricanes, or through human conflict—communities can become completely isolated: roads are washed out, bridges collapse, and many areas become unreachable via traditional logistics. In such situations, delivering essential supplies like medications, food, or blood can mean the difference between life and death.

Drones provide a powerful, rapid alternative: they can **fly over obstructed routes**, reaching remote settlements within minutes instead of hours. During the critical first 72 hours of a crisis—commonly dubbed the "golden window"—drones have enabled medical teams to deliver life-saving aid when roads were entirely impassable [76]. A striking case comes from Rwanda: Zipline's autonomous delivery drones now handle approximately 75% of the nation's blood supply outside Kigali, delivering to rural health centers in 15 minutes—a task that would require several hours by road [77].

In this context, the current thesis proposes deploying a swarm of drones coordinated by a mission computer loaded in the brain drone with the centralized policy. The base station launches this "leader", which carries the drones (through waypoint assignment) to a strategically chosen standoff position. Once in place, the mission computer autonomously issues or updates preplanned waypoints to the surrounding drones, thereby orchestrating precise and resilient delivery operations in blocked or disaster-stricken zones.

4.1.3 Scenario 3: Search and Rescue in Hazardous Environments

Collapsed buildings, cave systems, and wildfire zones present extreme hazards to human search and rescue teams—including structural collapse, thick smoke, and unstable terrain. Drones provide a strategic advantage by rapidly accessing these dangerous environments and delivering actionable data back to command centers.

Drones in this architecture are designed to be outfitted from the outset with mission-critical sensing capabilities—including thermal cameras, LiDAR scanners, and optical payloads—to support search-and-rescue operations and environmental awareness. Although this thesis

emphasizes coordination and inference using kinematic motion data, the underlying framework fully anticipates the integration of such sensors in practical deployments [78, 79].

4.2 Proposed Solution

While we do not explicitly simulate these scenarios end-to-end, they inform the design requirements (resilience to communication loss, dynamic target generation, etc.) that our solution addresses. This thesis proposes a single, unified architecture to address the three problem scenarios described previously. The core idea is to convert every mission objective—whether neutralizing a hostile node, delivering supplies to a clinic, or thoroughly searching a collapsed structure—into a set of spatial tasks (waypoints). A centralized policy, hosted on a designated leader node (or at the base), computes, assigns and updates waypoint lists and issues action commands to the agents of the follower drones. Below we present the components of this architecture and justify the design choices.

4.2.1 From targets to waypoints

All mission goals are represented as **waypoints**. Converting complex objectives into waypoint sets reduces high-level goals to a common planning interface and allows the same allocation and routing machinery to be reused across defense, logistics and SAR tasks. Waypoint metadata enables encoding mission preferences (e.g., delivery deadlines, threat exposure limits, or search coverage density).

4.2.2 Objective formulations and routing criteria

Different missions require different objective functions. The centralized planner supports two basics criteria for waypoint assignment and routing:

- **Distance-minimizing routing:** minimize total traveled distance for each drone (battery/energy efficient); useful when endurance is primary.

- **Priority / sequential coverage:** enforce strict task ordering when certain waypoints are more urgent or must be visited in sequence (e.g., neutralization before resupply).

4.2.3 Path planning and execution

After waypoint assignment, each agent creates its own trajectory. Trajectory planning must account for obstacle avoidance, no-fly zones, and energy consumption models. Where appropriate, path planning and routing are coupled (route + trajectory co-optimization) to respect UAV dynamics and battery constraints.

Chapter 5

Experimental Setup & Evaluation

5.1 Frameworks Used

The experimental setup of this thesis relies on a number of established frameworks that provide physics simulation, reinforcement learning (RL) interfaces, and reproducible training pipelines.

5.1.1 Learning To Fly (gym-pybullet-drones)

Panerati et al. introduce **gym-pybullet-drones** [80], an open-source Gym-style environment for controlling one or more quadcopters via classical control (e.g. PID) or Reinforcement Learning. It supports multi-agent settings and implements realistic physics such as drag, ground effect, and downwash, uses URDF-based quadcopter models, and allows both CPU and GPU rendering. It also offers multiple observation modes (kinematic state; RGB, depth, segmentation imagery), multiple action modalities (motor RPMs or desired velocities), and is built to run many parallel environments in headless or GUI modes. The environment is compatible with RL libraries such as Stable Baselines3 and RLlib.

5.1.2 Bullet Physics / PyBullet

Bullet [81] is an open-source physics engine providing rigid-body dynamics, collision detection (discrete and continuous), constraint solving, URDF model support, etc. PyBullet [82], the Python binding, exposes these features with support for forward and inverse kinematics, ray intersection queries, and resetting / restarting simulations in a deterministic

way. It allows configuring physics timestep and solver parameters (to trade off realism vs computation), supports GUI as well as headless (non-visual) modes, which is beneficial for large-scale RL experiments.

5.1.3 Stable Baselines3

Stable Baselines3 (SB3) [83] is a library implementing state-of-the-art RL algorithms in PyTorch [84], providing clean, well-tested implementations of PPO, SAC, TD3, etc.Its unified API structure simplifies switching between algorithms and customizing policies (e.g. MLP vs CNN policies, dict observations) and input feature extraction. SB3 includes tooling for reproducibility (logging, evaluation, type hints, tests) and supports vectorized environments for parallel experience collection.

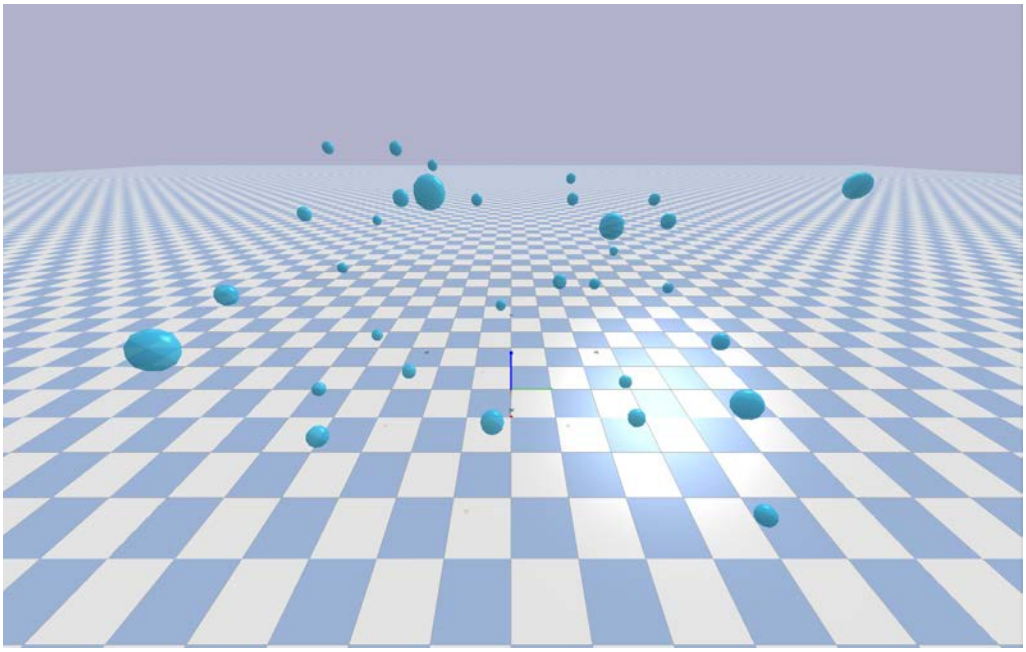

Figure 5.1: PyBullet environment initial configuration: agents and waypoints placed within a small workspace

5.2 The Custom Environment: Marl_dynamic_waypoints

While the Learning to Fly framework (gym-pybullet-drones) provides a robust foundation, this thesis required development of a custom environment that inherits core functionalities from **BaseAviary** and **BaseRLAviary** environments, and extends them with mechanisms tailored to multi-agent waypoint navigation under dynamic assignment and prioritization. The inherited base classes supply support for rigid-body simulation of multiple quadcopters (including URDF models, aerodynamic effects such as drag, ground effect, downwash, etc.), collision detection, and flexible and customizable observation and action modalities, as defined in gym-pybullet-drones. **Marl_dynamic_waypoints** introduces additional features: dynamic waypoint generation; strategies for assigning waypoints among agents; priority modes; customized observations per agent; a shaped reward structure encouraging efficient task division, fairness, and collision avoidance; and new termination conditions to better align with use-case requirements. For a detailed description of the initialization parameters, see Appendix A.

5.2.1 Features of the custom environment

Shared waypoint pool with spatial constraints and markers. At initialization and each reset, the environment generates a global pool of waypoints inside configurable workspace bounds, enforcing a minimum inter-waypoint separation, if the available space allows it.

Waypoint lifecycle and ownership. Waypoints progress through `AVAILABLE`→`CLAIMED`→`COMPLETED`, with per-waypoint owners and per-agent counters for assigned/failed/completed attempts; reaching a waypoint requires holding inside a radius for a fixed number of steps.

Assignment and re-assignment policies. Initial and ongoing assignment support **distance** and **sequential** modes; when distance mode is enabled, a Hungarian solver [85, 86] performs global distance-optimal matching of free agents to available waypoints.

Richer multi-agent observations. Each agent's kinematic base observation (**102** features for PID at `ctrl_freq`=60 Hz) is extended with **7** waypoint features (relative position, distance, "has-target" flag, available-waypoints count, agent's completion count) and **6**$(N-1)$

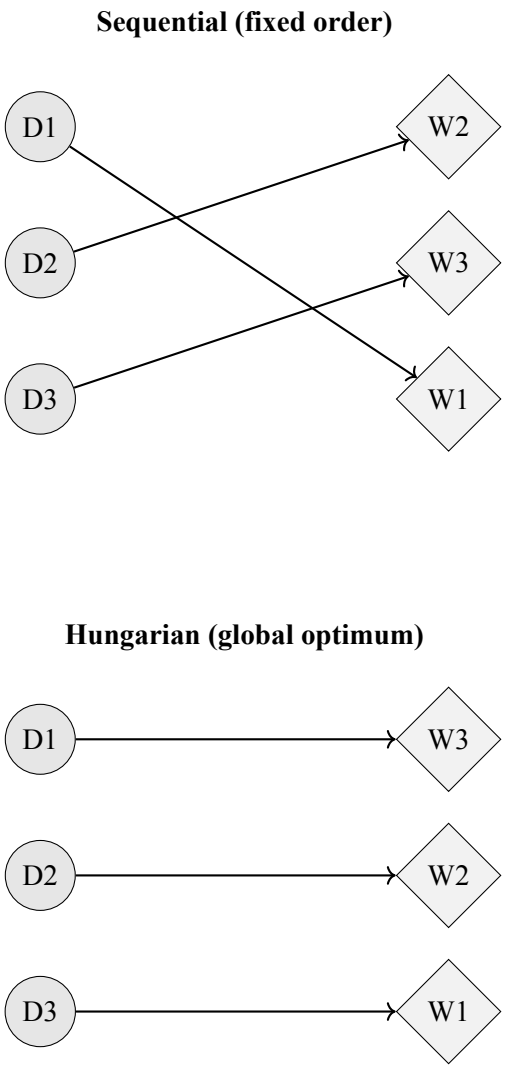

Figure 5.2: Sequential vs. Hungarian assignment schematic (illustrative).

neighbor features per other agent (relative position & velocity), with non-finite values sanitized before return. Overall per-agent dimension:

$$102 + 7 + 6(N - 1) = 6N + 103.$$

For example, with $N{=}5$ drones, this yields **133** features per agent and **665** when flattened across all agents. For more information, see Appendix B.

Custom shaped rewards. Rewards include idle penalties; "has-target" bonus; bounded distance shaping; progress shaping vs. last step; velocity-toward-target shaping with extra penalty when moving away; in-radius bonus / out-of-radius penalty; altitude constraints; proximity penalty and mid-range separation bonus; and a strong completion bonus with diminishing returns. For more details look Appendix.

Action space and control interface. The controller operates in two layers. At the high level, the reinforcement-learning policy produces a 3D motion intent for each agent—essentially how it should move in space at the next step. The environment combines this intent with the guidance vector that points to the agent's assigned waypoint, yielding the next target position. At the low level, a PID takes this input and translates it into motor RPMs, applying the usual safeguards (saturation limits, rate/jerk limiting, and anti-windup) so that the drone follows the command smoothly and remains stable.

Diagnostics for analysis. Per-agent `info` reports current target, available-waypoints, position, and assigned/failed/completed counts.

Reset hooks and reproducibility helpers. Reset reinitializes all counters, progress trackers, action-smoothing state, and (optionally) regenerates the waypoint pool.

5.3 Training

Algorithm Overview Proximal Policy Optimization (PPO) is an on-policy actor-critic algorithm. It optimizes a clipped surrogate objective, which stabilizes training by preventing excessively large policy updates. Generalized Advantage Estimation (GAE) improves variance reduction, and entropy regularization promotes exploration [87].

Choice of PPO over Other Algorithms PPO was selected for this study because it balances stability, ease of tuning, and strong empirical performance in continuous control and cooperative multi-agent tasks.

- **Stable policy updates via clipping.** PPO's clipped surrogate objective constrains destructive updates without TRPO's second-order machinery, yielding a simple, first-order method that is empirically stable across many benchmarks. [87]
- **Practical robustness and low tuning burden.** In practice, PPO requires fewer brittle design choices than many off-policy baselines, while still benefiting from minibatch updates and multiple epochs per rollout (good wall-clock performance with vectorized environments). [87]

- **Strong results in multi-agent cooperation.** PPO variants with centralized critics (e.g., MAPPO) achieve competitive or superior performance (and often competitive sample efficiency) on standard cooperative MARL suites with minimal hyperparameter tuning—an attractive property for our centralized controller. [88]

- **Mature, well-tested implementation.** The SB3 PPO implementation includes practical improvements (advantage normalization, optional value-function clipping) and integrates seamlessly with Gymnasium-style vectorized training and logging.

Trade-offs vs. Off-Policy Methods. Off-policy algorithms like DDPG/TD3 and SAC can be more sample-efficient, but introduce additional moving parts and tuning sensitivity. DDPG is known to suffer from overestimation bias and instability; TD3 mitigates this via twin critics, target-policy smoothing, and delayed updates at the cost of extra complexity. SAC's maximum-entropy objective is powerful and stable with good sample efficiency, but adds temperature tuning (often automated) and a heavier training loop. Given our dense-reward, fixed-horizon tasks and easy parallelization, PPO's on-policy stability and simplicity were prioritized. [89, 50, 90]

Figures 5.3 and 5.4 show the reward progression over training, and the distribution of episodic rewards (evaluated at regular checkpoints) for the MARL PPO setup.

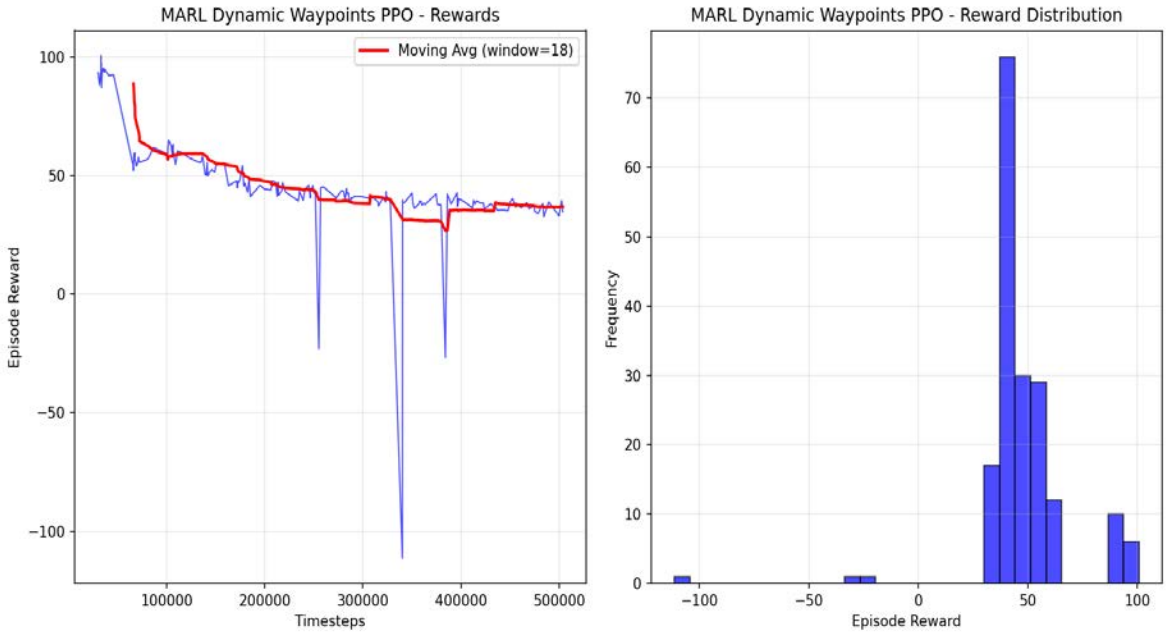

Figure 5.3: Reward progression during training with 6 drones and 50 waypoints in a small workspace, using distance-based assignment.

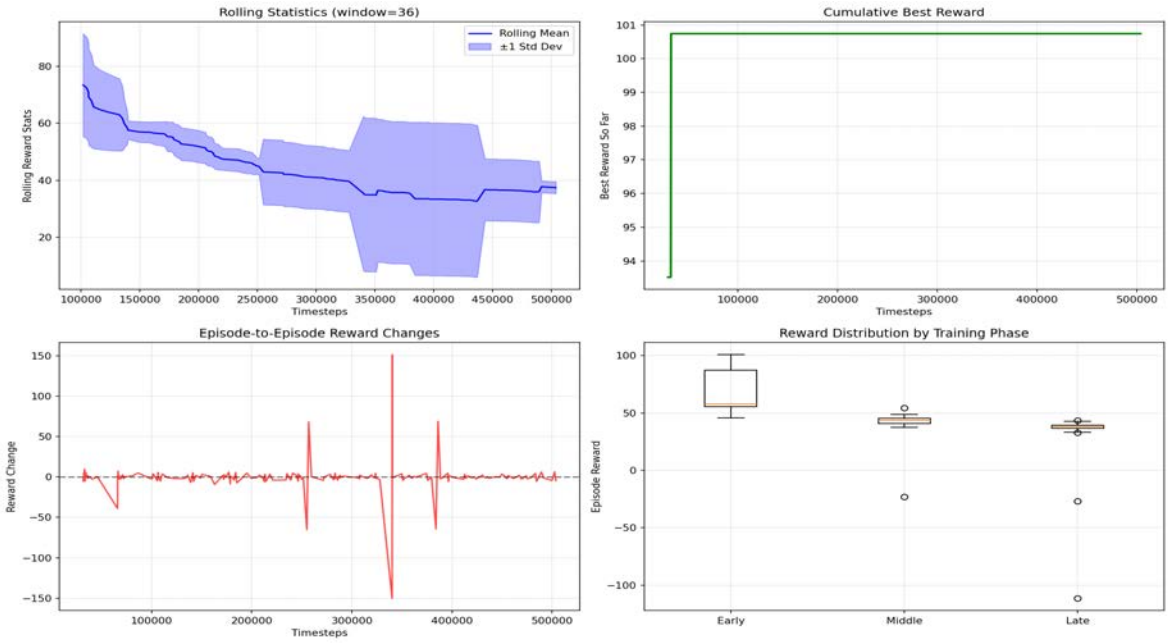

Figure 5.4: Reward statistics during training with 6 drones and 50 waypoints in a small workspace, using distance-based assignment.

Interpreting the curves. Because training uses observation *and* reward normalization (SB3 `VecNormalize`), the numeric scale of returns drifts as the running mean/variance are updated online. It is therefore common to see episodic rewards start high and then decline to a stable band even while behavior improves. [91, 92]

Reward vs. Timesteps. The smoothed curve stabilizes after the warm-up phase; large downward spikes correspond to rare penalty events rather than sustained collapse. In our runs, task-level metrics (success rate, waypoint completions) remain near the ceiling despite the numeric drift in normalized reward. [91]

Reward Distribution. Most episodes concentrate in a narrow reward band with few outliers, indicating increasing stability and consistency as training progresses. The narrowing spread is more informative than the absolute level when rewards are normalized. [92]

What the decline does *not* imply (and what it does). A downward trend in the *normalized* return does *not* mean degraded policy quality; it mainly reflects changing normalization statistics and occasional penalties. What matters is convergence of success/completion metrics and healthy PPO diagnostics (e.g., moderate KL and clip fraction), which we observe[91].

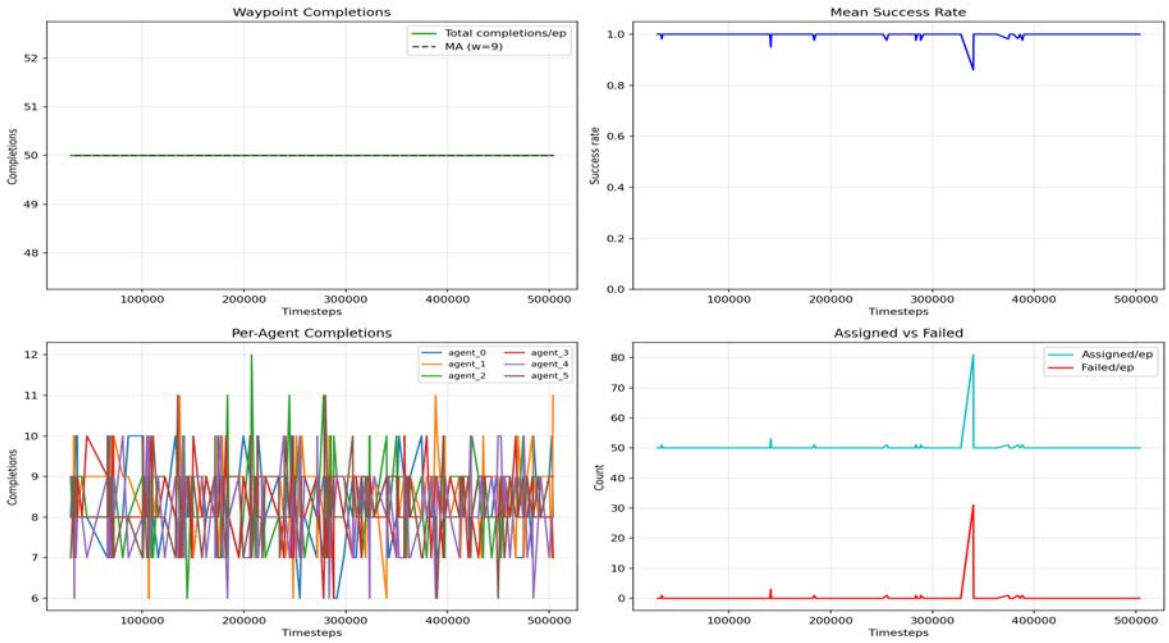

Figure 5.5: Waypoint metrics during training with 6 drones and 50 waypoints in a small workspace, using distance-based assignment.

Task-level metrics (success & completions). Figure 5.5 shows that task performance saturates: total waypoint completions per episode remain at the cap (50), the mean success rate stays near 1 with only brief dips, and per-agent completions fluctuate around 8–10 without persistent starvation. The "assigned vs. failed" panel confirms that failures are rare and align with the few large negative outliers seen in the reward curves (Fig. 5.3).

5.4 Metrics & Evaluation

To evaluate system performance, we design controlled scenarios that vary: (i) the **task load** (number of waypoints per episode), (ii) the **workspace bounds** (tight vs. wide), and (iii) the **assignment strategy** (**distance** vs. **sequential**). For each configuration, we report outcome and efficiency metrics that are standard in multi-robot coverage and task allocation: **success**, **failures**, **total path length** (as an energy proxy), and **elapsed time / makespan**.

5.4.1 Metrics

Let W denote the number of spawned waypoints, D the number of drones, C the number of waypoints successfully completed, and F the number of failed attempts. A failed attempt occurs whenever a claimed waypoint is released and reassigned due to timeout, stagnation, or inability to meet the hold requirement; since reassignment can occur multiple times per waypoint, F can exceed W.

In addition, we record:

- **Total distance:** cumulative distance travelled by all drones (proxy for energy consumption).

- **Time:** elapsed time until all waypoints are completed or the episode ends.

5.4.2 Evaluation Setup

We consider two workspace bounds: a **small bound** $((-5.0,\ -5.0,\ 0.5),\ (5.0,\ 5.0,\ 5.0))$ that creates a compact search volume, and a **large bound** $((-20.0,\ -20.0,\ 0.5),\ (20.0,\ 20.0,\ 10.0))$ that reduces congestion effects. Each experiment is averaged over 10 independent runs with randomized seeds to account for stochasticity in waypoint generation and agent behavior.

Results on Small Bound

Table 5.1: Mean metrics of 10 runs per waypoint count and strategy for the small bound.

Waypoints	Strategy	Completed	Failures	Total dist (m)	Time (s)
20	distance	20	0,00	76,63	22,89
20	sequential	20	0,00	191,14	52,74
60	distance	60	0,00	207,24	64,60
60	sequential	60	0,40	582,73	164,56
100	distance	100	0,20	321,73	105,41
100	sequential	100	0,33	810,30	236,85
200	distance	200	45,50	328,31	120,13
200	sequential	200	45,50	1 664,75	526,44
500	distance	500	0,00	527,40	210,27
500	sequential	500	545,44	6 255,10	1 600,10

The compact workspace magnifies the cost of inefficient assignments. As shown in Table 5.1, the **distance** strategy consistently produces much shorter paths and faster completion times than **sequential**. Quantitatively:

- At 20 WPs (waypoints): distance yields **60,0%** shorter path and **56,7%** lower time.
- At 60 WPs: path length is **64,4%** lower and time **60,7%** lower.
- At 100 WPs: **60,3%** shorter paths and **55,5%** lower time.
- At 200 WPs: **80,2%** shorter paths and **77.18%** lower time.
- At 500 WPs: the gap is largest, with **91,6%** shorter paths and **86,9%** lower time.

These trends highlight that distance-aware assignment scales far better with task load, while sequential ordering suffers from congestion and repeated reassignments. In dense workspaces, such differences directly translate into reduced battery drain and shorter mission durations.

Results on the bigger bound

Table 5.2: Mean metrics of 10 runs per waypoint count and strategy for the large bound.

Waypoints	Strategy	Completed	Failures	Total dist (m)	Time (s)
20	distance	20	0,40	354,61	77,78
20	sequential	20	1,90	773,94	161,17
60	distance	60	1,00	692,18	169,39
60	sequential	60	2,70	1 978,42	433,22
100	distance	100	0,40	886,10	238,82
100	sequential	100	80,30	2 887,73	698,43
200	distance	199	0,40	1 330,64	370,50
200	sequential	200	116,70	5 708,12	1 371,31
500	distance	500	574,00	2 248,70	661,00
500	sequential	442	12,00	13 065,40	3 000,00

With a larger workspace, absolute path lengths and times increase for both strategies, but the distance-aware assignment still dominates across waypoint counts. Quantitatively, distance vs sequential reduces:

- At 20 WPs: distance yields **54,2%** shorter paths and **51,7%** lower time.
- At 60 WPs: **65,0%** shorter paths and **60,9%** lower time.
- At 100 WPs: **69,3%** shorter paths and **65,8%** lower time.
- At 200 WPs: **76,7%** shorter paths and **73,0%** lower time.
- At 500 WPs: **82,8%** shorter paths and **77,9%** lower time.

In the 500-waypoint sequential setting, performance degrades significantly, with only 442 completions and 12 failures; this is mostly due to the 3000 s maximum simulation time cap.

Failures and success: for 20–200 WPs, distance shows fewer reassignments and higher success. This indicates that even with reduced congestion in a larger area, global distance-optimal matching avoids wasted travel. At 500 WPs, distance still achieves far shorter paths/times and a higher completed count (500 vs 442), but logs many more “Failures” .

Chapter 6

Conclusion and Future Work

6.1 Conclusions

This thesis presented a reinforcement-learning–driven approach for coordinated multi-UAV waypoint coverage using centralized policy with per-drone actions operating in a custom PyBullet–based environment. The system unifies (i) **a shared waypoint pool** with configurable bounds and lifecycle, (ii) **assignment strategies** (distance vs. sequential) including global re-matching, and (iii) **shaped rewards** that encourage efficient task division, progress toward targets, and collision avoidance. The framework is implemented on top of **gym-pybullet-drones PyBullet** and **Stable Baselines3** that exposes reproducible, modular training and visualization pipelines.

Summary of contributions.

- A physics-based, multi-agent environment (`Marl_dynamic_waypoints`) with dynamic waypoint generation, ownership, reassignment, rich observations, and explicit termination/diagnostics for coverage analysis.

- A centralized PPO policy for multi-drone control with flattened joint observations, with effective task-allocation modes (distance, sequential).

- A metrics suite and evaluation protocol across waypoint loads and workspace sizes, reporting success, failures, makespan, and total path length, with 10-run means per configuration.

Key findings. Under a compact "small-bound" workspace, *distance-based* assignment consistently dominated *sequential* ordering in efficiency: across 20–500 waypoints it achieved markedly lower total travel and elapsed time (e.g., 60–92% shorter paths and 56–87% lower time at representative loads), while maintaining near-perfect completion at moderate task sizes. These gaps widen as workload increases, indicating that global, geometry-aware matching avoids compounding detours that arise under rigid sequential visits.

Limitations and threats to validity.

- **Shared Policy** The environment could not handle different agents with different policies.
- **Physics and sensors.** Results are simulation-based with kinematic observations; no onboard perception (e.g., LiDAR/thermal) or wind/ground-effect variability beyond PyBullet's model were included.
- **Scalability metric** Although able to increase the swarm size, the computational power needed was not available for a larger swarm size.
- **Energy models.** Path length is used as an energy proxy; high-fidelity battery/weight models were not integrated.

6.2 Future Work

Building on the above, we outline concrete extensions:

1. **From centralized PPO (shared policy) to MAPPO (per-agent policies).** Replace the single shared PPO policy (controlling all agents) with a MAPPO formulation where each agent maintains its own policy.
2. **Sim-to-real transfer.** Introduce domain randomization (dynamics, delays, sensor noise), system identification, and bridge the gap to hardware. Evaluate transfer on micro-UAVs.
3. **Perception and multi-modal policies.** Integrate LiDAR/thermal/RGB sensing for search-and-rescue or EW-contested scenarios.

4. **Energy–and risk–aware planning.** Incorporate battery and mass models, wind fields, and risk maps into both routing and rewards; evaluate trade-offs between endurance and latency.

Closing remark. Overall, the results support a pragmatic recipe for swarm coverage: combine a stable on-policy learner with global, geometry-aware task assignment and minimalistic observations, then progressively layer safety, perception, and deployment tooling. This balances empirical performance with engineering simplicity and provides a clear runway toward field-ready autonomy.

Bibliography

[1] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 2 edition, 2018.

[2] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[3] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. MIT Press, 2012.

[4] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[5] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[6] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[7] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 1106–1114, 2012.

[8] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[9] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, 2015.

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016.

[11] Martín Abadi et al. Tensorflow: A system for large-scale machine learning. In *OSDI*, 2016.

[12] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems (NeurIPS) Systems Track*, 2019.

[13] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2 edition, 2009.

[14] J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability*, pages 281–297, 1967.

[15] I. T. Jolliffe. *Principal Component Analysis*. Springer, 2 edition, 2002.

[16] Peter J. Rousseeuw. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20(1):53–65, 1987.

[17] Michel Verleysen and Damien François. The curse of dimensionality in data mining and time series prediction. In *IWANN*, volume 3512 of *Lecture Notes in Computer Science*, pages 758–770, 2005.

[18] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015.

[19] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828, 2013.

[20] Vincent François-Lavet, Peter Henderson, Riashat Islam, Marc G. Bellemare, and Joelle Pineau. An introduction to deep reinforcement learning. *Foundations and Trends in Machine Learning*, 11(3–4):219–354, 2018.

[21] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, 1989.

[22] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257, 1991.

[23] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

[24] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015.

[25] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 315–323, 2011.

[26] Anders Krogh and John A. Hertz. A simple weight decay can improve generalization. In John E. Moody, Steve J. Hanson, and Richard P. Lippmann, editors, *Advances in Neural Information Processing Systems 4*, pages 950–957. Morgan Kaufmann, 1992.

[27] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations (ICLR)*, 2019. arXiv:1711.05101.

[28] Lutz Prechelt. Automatic early stopping using cross validation: Quantifying the criteria. *Neural Networks*, 11(4):761–767, 1998.

[29] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning (still) requires rethinking generalization. *Communications of the ACM*, 64(3):107–115, 2021.

[30] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[31] Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990.

[32] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[33] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3–4):279–292, 1992.

[34] Damien Ernst, Pierre Geurts, and Louis Wehenkel. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6:503–556, 2005.

[35] Martin Riedmiller. Neural fitted q iteration – first experiences with a data efficient neural reinforcement learning method. In *Proceedings of the 16th European Conference on Machine Learning (ECML 2005)*, volume 3720 of *Lecture Notes in Computer Science*, pages 317–328. Springer, 2005.

[36] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[37] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double Q-learning. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence (AAAI 2016)*, pages 2094–2100, 2016.

[38] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot, and Nando de Freitas. Dueling network architectures for deep reinforcement learning. In *Proceedings of the 33rd International Conference on Machine Learning (ICML 2016)*, pages 1995–2003, 2016.

[39] Marc G. Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning (ICML 2017)*, pages 449–458, 2017.

[40] Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI 2018)*, pages 3215–3222, 2018.

[41] Richard S. Sutton, David A. McAllester, Satinder P. Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 1057–1063, 1999.

[42] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3–4):229–256, 1992.

[43] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *Proceedings of the 31st International Conference on Machine Learning (ICML 2014)*, pages 387–395, 2014.

[44] Matthias Lehmann. The definitive guide to policy gradients in deep reinforcement learning: Theory, algorithms and implementations. arXiv preprint, 2024. arXiv:2401.13662.

[45] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *International Conference on Learning Representations (ICLR)*, 2016. arXiv:1509.02971.

[46] Volodymyr Mnih, Adria Puigdomènech Badia, Mehdi Mirza, Alex Graves, Tim Harley, Timothy P. Lillicrap, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *Proceedings of the 33rd International Conference on Machine Learning (ICML 2016)*, pages 1928–1937, 2016.

[47] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[48] Sham M. Kakade. A natural policy gradient. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 1531–1538, 2001.

[49] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization. In *Proceedings of the 32nd International Conference on Machine Learning (ICML 2015)*, pages 1889–1897, 2015.

[50] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

[51] Scott Fujimoto, Herke van Hoof, and David Meger. Twin delayed ddpg (td3). OpenAI Spinning Up documentation, 2018. Based on "Addressing Function Approximation Error in Actor-Critic Methods".

[52] Le Zhang, Yong Gu, Xin Zhao, Yanshuo Zhang, Shu Zhao, Yifei Jin, and Xinxin Wu. Generalizing soft actor-critic algorithms to discrete action spaces. *arXiv preprint*, 2024. arXiv:2407.11044.

[53] J. Zhou et al. Stable discrete sac (sdsac): Improving discrete soft actor-critic with stability enhancements. *arXiv preprint*, 2022. arXiv:2209.10081.

[54] Jiaping Xiao, Rangya Zhang, Yuhang Zhang, and Mir Feroskhan. Vision-based learning for drones: A survey. *arXiv preprint arXiv:2312.05019*, 2023.

[55] Li Zhou, Hao Yin, Haitao Zhao, Jibo Wei, Dewen Hu, and Victor C. M. Leung. A comprehensive survey of artificial intelligence applications in uav-enabled wireless networks. *Digital Communications and Networks*, 2024. Journal pre-proof.

[56] Chijioke C. Ekechi, Tarek Elfouly, Ali Alouani, and Tamer Khattab. A survey on uav control with multi-agent reinforcement learning. *Drones*, 9(7):484, 2025.

[57] Elia Kaufmann, Antonio Loquercio, René Ranftl, Alexey Dosovitskiy, Vladlen Koltun, and Davide Scaramuzza. Deep drone racing: Learning agile flight in dynamic environments. In *Proceedings of the 2nd Conference on Robot Learning (CoRL)*, volume 87 of *Proceedings of Machine Learning Research*. PMLR, 2018.

[58] Elia Kaufmann, Antonio Loquercio, René Ranftl, Matthias Müller, Vladlen Koltun, and Davide Scaramuzza. Deep drone acrobatics. In *Robotics: Science and Systems (RSS)*, 2020.

[59] Elia Kaufmann, Philipp Fässler, Marco Bauersfeld, Antonio Loquercio, and Davide Scaramuzza. Champion-level drone racing using deep reinforcement learning. *Nature*, 620, 2023.

[60] Juraj Obradović, Marko Križmančić, and Stjepan Bogdan. Decentralized multi-robot formation control using reinforcement learning. *arXiv preprint arXiv:2306.14489*, 2023.

[61] Zhijun Liu, Jie Li, Jian Shen, Xiaoguang Wang, and Pengyun Chen. Leader–follower uavs formation control based on a deep q-network collaborative framework. *Scientific Reports*, 14(4674), 2024.

[62] Abhay Rawat and Kamalakar Karlapalem. Multi-robot formation control using reinforcement learning. *arXiv preprint arXiv:2001.04527*, 2020.

[63] Sumeet Batra, Zhehui Huang, Aleksei Petrenko, Tushar Kumar, Artem Molchanov, and Gaurav S. Sukhatme. Decentralized control of quadrotor swarms with end-to-end deep reinforcement learning. In *Proceedings of the 5th Conference on Robot Learning (CoRL)*, volume 164 of *Proceedings of Machine Learning Research*, pages 576–586. PMLR, Nov 2022.

[64] Zhehui Huang, Zhaojing Yang, Rahul Krupani, Baskın Şenbaşlar, Sumeet Batra, and Gaurav S. Sukhatme. Collision avoidance and navigation for a quadrotor swarm using end-to-end deep reinforcement learning. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 300–306. IEEE, 2024.

[65] Guanya Shi, Wolfgang Hönig, Xichen Shi, Yisong Yue, and Soon-Jo Chung. Neural-swarm2: Planning and control of heterogeneous multirotor swarms using learned interactions. *IEEE Transactions on Robotics*, 38(2):1063–1079, 2022.

[66] Yiting Chen, Runfeng Chen, Yuchong Huang, Zehao Xiong, and Jie Li. Drl-based improved uav swarm control for simultaneous coverage and tracking with prior experience utilization. *Drones*, 8(12):784, 2024.

[67] Air Force Research Laboratory. Tactical high power operational responder (thor): Counter-swarm high power weapon, n.d. Official AFRL technology page.

[68] Air Force Research Laboratory. Afrl awards contract for drone killer, mjölnir; brings new drone ”hammer” to the fight, 02 2022.

[69] Financial Times. Russia’s reckless jamming of gps systems, 09 2025. Reports increased GPS jamming impacting European aviation.

[70] OPSGROUP Workgroup. Gps spoofing: Final report. Technical report, OPSGROUP, 09 2024.

[71] Defense Advanced Research Projects Agency. Subterranean challenge: Final event overview, 2021.

[72] Y. Cheng, X. Wang, S. Li, and H. Zhang. A review of UAV autonomous navigation in GPS-denied environments. *Robotics and Autonomous Systems*, 164:104533, 2023.

[73] F. Fattori, R. Fantacci, and F. Chiti. Tethered drones: A comprehensive review of systems and applications. *Drones*, 9(6):425, 2025.

[74] The Economist. How new drones are sneaking past jammers on ukraine's front lines. *The Economist*, May 2025. Reports on fibre-optic control links used to evade jamming; accessed 2025-09-17.

[75] Institute for the Study of War. Russian drone innovations are likely achieving effects of battlefield air interdiction in ukraine, August 2025. Includes reporting on Russian employment of fibre-optic UAVs; accessed 2025-09-17.

[76] World Food Programme. Delivering critical medical aid within the first 72 hours of a crisis, n.d.

[77] World Health Organization. Drones take rwanda's national blood service to new heights, June 2019.

[78] M. Lyu and colleagues. Unmanned aerial vehicles for search and rescue: A survey. *Remote Sensing*, 15(13):3266, 2023.

[79] J. A. H. Velasco. Development of uavs equipped with thermal sensors for disaster response in collapsed structures. In *ICAS 2024*, 2024.

[80] Jacopo Panerati, Hehui Zheng, SiQi Zhou, James Xu, Amanda Prorok, and Angela P. Schoellig. Learning to fly—a gym environment with pybullet physics for reinforcement learning of multi-agent quadcopter control. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7512–7519. IEEE, 2021.

[81] Erwin Coumans. Bullet physics simulation. In *ACM SIGGRAPH 2015 Courses*. ACM, 2015.

[82] Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. `http://pybullet.org`, 2016. Accessed: 2025-09-12.

[83] Antonin Raffin, Ashley Hill, Mark Plappert, et al. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.

[84] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. *arXiv preprint arXiv:1912.01703*, 2019.

[85] Harold W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1-2):83–97, 1955.

[86] James Munkres. Algorithms for the assignment and transportation problems. *Journal of the Society for Industrial and Applied Mathematics*, 5(1):32–38, 1957.

[87] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. In *arXiv preprint arXiv:1707.06347*, 2017.

[88] Chao Yu, Akash Velu, Eugene Vinitsky, Jiaxuan Gao, Yu Wang, Alexandre Bayen, and Yi Wu. The surprising effectiveness of ppo in cooperative, multi-agent games. *arXiv preprint arXiv:2103.01955*, 2021.

[89] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1587–1596. PMLR, 2018.

[90] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. *arXiv preprint arXiv:1709.06560*, 2018.

[91] Antonin Raffin and Stable-Baselines3 Contributors. Reinforcement learning tips and tricks, 2025.

[92] Stable-Baselines3. Vectorized environments (includes *VecNormalize* subsection), 2025.

APPENDICES

Appendix A

Environment Initialization Parameters

Below is a description of the inputs of the '__init__' function for the 'Marl_dynamic_waypoints' environment. These parameters control the setup of the environment before each experiment, defining drone behaviour, goal generation, termination criteria, and more.

`drone_model : DroneModel = DroneModel.CF2X` Specifies the quadcopter model used for each agent. The default is `CF2X` (Crazyflie 2.x nano-quadrotor). Different models may alter mass, inertia, arm length, or propeller geometry, which influences flight dynamics.

`num_drones : int = 2` The number of drones in the environment. Increasing this number increases the number of agents in the simulation.

`num_waypoints : int = 20` The total number of waypoints generated in the pool at the start of each episode. These are the targets agents must claim and visit, according to the assignment strategy.

`initial_xyzs, initial_rpys : optional` Initial positions (x, y, z) and orientations (roll, pitch, yaw) of the drones. If `None`, default or randomized initial placements/orientations are used.

`physics : Physics = Physics.PYB` The physics backend used. E.g. PyBullet. Other physics engines (if supported) or modes may be selectable, affecting realism, numerical stability, and computational cost.

`pyb_freq : int = 240` The physics simulation update frequency (in Hz) under PyBullet—how many physics steps per second are computed. Higher frequencies give finer phys-

ical fidelity but incur more computation.

`ctrl_freq : int = 60` The control frequency (in Hz)—how often control decisions/actions are sent (by the agent). It should divide the physics frequency to maintain alignment and stability.

`gui : bool = False` Whether to display a graphical user interface (visualization) of the simulation. Turning GUI off often increases speed, especially when running many parallel environments.

`record : bool = False` Whether to record video or logs of the simulation (e.g. flight paths, waypoints, agent behaviour).

`obs : ObservationType = ObservationType.KIN` Type of observation provided to agents. `KIN` means kinematic data (positions, velocities, orientation, etc.). Other types might include vision, depth, segmentation, hybrid, etc.

`act : ActionType = ActionType.PID` The action/control mode the agent uses. `PID` may mean using a built-in PID controller; other action types could include direct motor RPM commands, desired velocities, etc.

`episode_len_sec : int = 300` The maximum duration of an episode in seconds. If this time is exceeded (and not all waypoints completed, etc.), the episode terminates.

`verbose : bool = False` Whether to print or log additional debugging or status information during environment initialization and stepping.

`waypoint_radius : float = 0.5` Radius around a waypoint within which a drone is considered to have "arrived" at the waypoint.

`waypoint_hold_steps : int = 5` Number of consecutive control steps a drone must remain within the waypoint radius to count the waypoint as completed (i.e. to "hold" it).

`priority_mode : str = "distance"` Strategy for assigning waypoints to drones. Possible modes include "distance" (closest first), "sequential" (in fixed order).

`min_waypoint_separation : float = 2.0` Minimum allowed Euclidean distance between any two waypoints when they are generated—this avoids clustering and improves coverage.

`workspace_bounds : tuple = None` Defines bounds of the workspace as e.g. $((x_{\text{min}}, y_{\text{min}}, z_{\text{min}}), (x_{\text{max}}, y_{\text{max}}, z_{\text{max}}))$. If `None`, a default workspace is used.

`visualize_waypoints : bool = None` Whether the waypoint positions are visualized in the environment (e.g. markers) for debugging or analysis. If `None`, defaults may be inferred from GUI settings.

`waypoint_marker_radius : float = 0.1` The size/radius of the visual marker representing waypoints (if visualization is enabled).

`terminate_when_pool_exhausted : bool = True` Whether the episode should terminate once all waypoints in the pool have been claimed or completed. If `True`, you do not wait full episode length once tasks are done.

`early_termination_grace_steps : int = 1` Number of extra steps after the final waypoint is done (or pool exhausted) before the termination actually happens.

Appendix B

Observation Specification

Each agent i receives a stacked observation vector $\mathbf{o}_i \in \mathbb{R}^d$ composed of the base kinematic state, a sliding window of recent actions, waypoint-related features, and neighbor-relative features. The final dimension is

$$d = \underbrace{12 + 3\,\frac{f_{\text{ctrl}}}{2}}_{\text{BaseRLAviary (kin + action buffer)}} + \underbrace{7}_{\text{Waypoint features}} + \underbrace{6\,(N-1)}_{\text{Neighbor features}},$$

where f_{ctrl} is the controller frequency in Hz and N is the number of drones.

Base Kinematic State (12 dims)

- $p = (x, y, z)$: position in world coordinates.
- $\eta = (\phi, \theta, \psi)$: roll, pitch, and yaw (rad).
- $v = (\dot{x}, \dot{y}, \dot{z})$: linear velocity (m/s).
- $\omega = (\dot{\phi}, \dot{\theta}, \dot{\psi})$: body angular rates (rad/s).

Action History ($3 \times \frac{f_{\text{ctrl}}}{2}$ dims) A FIFO buffer stores the past $\frac{f_{\text{ctrl}}}{2}$ PID target vectors (0.5 s lookback). Each entry contributes three values, yielding:

$$\dim(\text{action buffer}) = 3 \times \frac{f_{\text{ctrl}}}{2}.$$

For example, $f_{\text{ctrl}} = 30\,\text{Hz} \Rightarrow 15$ entries $\Rightarrow 45$ dims; $f_{\text{ctrl}} = 60\,\text{Hz} \Rightarrow 30$ entries $\Rightarrow 90$ dims.

Waypoint Features (7 dims)

- $\Delta\mathbf{p}_{\text{wp}} = (x, y, z)_{\text{wp}} - \mathbf{p}$: relative position to the claimed waypoint.
- $\|\Delta\mathbf{p}_{\text{wp}}\|$: Euclidean distance to the waypoint.
- $\mathbb{1}_{\text{target}}$: binary indicator (1 if waypoint assigned, 0 otherwise).
- n_{avail}: count of waypoints still marked Available.
- $n_{\text{completed}}$: number of waypoints completed by agent i so far.

If no waypoint is assigned, the relative position, distance, and indicator entries are set to zero; the remaining statistics are still populated.

Neighbor Features ($6\,(N-1)$ **dims)** For every other agent $j \neq i$:

- $\Delta\mathbf{p}_{ij} = \mathbf{p}_j - \mathbf{p}_i$: relative position.
- $\Delta\mathbf{v}_{ij} = \mathbf{v}_j - \mathbf{v}_i$: relative velocity.

These six values are concatenated in a fixed agent ordering for all $j \neq i$.

Appendix C

Reward Function

- **Idle penalty:** if idle, penalize.
- **Target bias:** if a target is assigned, reward.
- **Distance shaping :** If being close to target, reward, (based on current distance)
- **Progress shaping:** If being closer to target than previous step, reward.
- **Velocity alignment:** If projection of velocity on target, reward, else penalize.
- **Waypoint radius:** If inside specified radius of waypoint, reward.
- **No target:** When no target is assigned, penalize.
- **Stability (attitude):** If tilted more than the allowed, penalize.
- **Altitude banding:** If height smaller than 0.5, penalize.
- **Inter-drone spacing:** Discourages near-collisions, soft preference for moderate separation.
- **Completion (diminishing):** Reward agent for waypoint completion. Early completions get larger rewards to bootstrap and reinforce the main task, later completions receive smaller rewards so as to not dominate the task

Table C.1: Per-term reward values.

Term	**Condition**	**Formula**
Idle penalty	$speed < 0.05$	-0.3
Target assigned bias	target exists	$+0.3$
Distance shaping	target exists	$0.8 \cdot \max(0,\, 1 - d_t/10)$
Progress shaping	target exists	$0.4 \cdot \text{clip}(d_{t-1} - d_t,\, -0.5,\, 0.5)$
Velocity alignment	$d_t > 10^{-3}$	$0.2\, v_{\parallel}$
Not-toward penalty	$v_{\parallel} \le 0$	-0.2
Waypoint in-radius bonus	$d_t < r_{\text{wp}}$	$+2.5$
Outside-radius tax	$d_t \ge r_{\text{wp}}$	-0.1
No-target penalty	no target	-0.2
Stability (attitude)	$\lvert\text{roll}\rvert > \pi/4$ or $\lvert\text{pitch}\rvert > \pi/4$	-0.1
Altitude low	$z < 0.5$	-0.5
Collision / near-miss	$d_{ij} < 1$ (each $j \neq i$)	$-\max(0,\, 1 - d_{ij})$
Separation bonus	$1.5 \le d_{ij} \le 4$	$0.02\, \dfrac{d_{ij} - 1.5}{2.5}$
Completion bonus (diminishing)	on new completion k	$20 \cdot \max(0.3,\, 1 - 0.2(k-1))$
Final clipping	—	$r \leftarrow \text{clip}(r,\, -3,\, 12)$

Appendix D

Training Hyperparameters

Table D.1: Hardware and training configuration.

Item	**Value / Notes**
CPU	AMD Ryzen 5 3600X
RAM	16 GB
GPU	None (CPU-only)
Vectorized envs	16
Agents per env N	4 - 6 - 8 (drones)
Waypoints per env W	50
Workspace (large)	$40 \times 40 \times 10.5$ m
Training timesteps	500 000
Evaluation every	1 000 timesteps
Training time	$\sim$ 2 hours

Parameter	**Value**	**Typical range**
Learning rate	3×10^{-4}	10^{-5}–3×10^{-4}
Discount γ	0.99	0.95–0.999
GAE λ	0.95	0.90–0.98
Clip range	0.10	0.10–0.30
Rollout steps n_{steps}	512	512 - 8192
Batch size	128	64–1024
Epochs	5	5–20
Entropy coef.	0.0	0.0–0.02
Value loss coef.	0.5	0.3–0.7
Max grad norm	0.5	0.3–1.0
Policy network (type)	MLP	MLP / CNN
Activation function	ReLU	Tanh, ReLU
Net architecture (pi)	[512, 512, 256]	2–4 layers, 64–1024
Net architecture (vf)	[512, 512, 256]	2–4 layers, 64–1024
Log std init	-0.8	-2.0–0.0

Table D.2: PPO hyperparameters (defaults close to widely used SB3 practice).