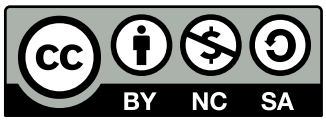
Aalto University
School of Science

Master's Programme in Mathematics and Operations Research

Multi-Agent Reinforcement Learning for Autonomous Unmanned Surface Vehicle Hunting Missions

Niki Leskinen

Master's Thesis
2026



Aalto University
School of Science

Author Niki Leskinen

Title Multi-Agent Reinforcement Learning for Autonomous Unmanned Surface Vehicle Hunting Missions

Degree programme Mathematics and Operations Research

Major Systems and Operations Research

Supervisor Prof. Kai Virtanen

Advisor Lauri Vasankari

Date April 23, 2026 **Number of pages** 78 **Language** English

Abstract

Unmanned surface vehicles (USVs) have become prominent in naval warfare during the Russo-Ukrainian war. USVs can be operated remotely or with different levels of autonomy, either individually or as part of a swarm. A swarm can be described as a decentralized group that acts in coordination. Advances in machine learning (ML) and more powerful onboard computing for ML inference enable greater autonomy and more advanced swarming behavior for USVs. One approach for enabling autonomous swarming behavior is multi-agent reinforcement learning (MARL). MARL is a sub-area of ML in which multiple agents learn through trial and error to choose actions that maximize rewards provided by the environment. It has been widely applied in robotics and has been explored for the control of USVs. This thesis investigates the applicability of MARL for controlling a swarm of USVs in a challenging two-phase mission in which the swarm must first locate and then destroy an adversarial surface combatant capable of hard-kill measures and jamming. The thesis reviews the relevant literature on MARL, swarming, naval warfare, and USVs. It then designs and implements a training framework based on multi-agent proximal policy optimization (MAPPO) for the task. Finally, the resulting controller is evaluated against an idealized rule-based reference controller, given full knowledge of the target. The aim is to assess whether MARL can provide a feasible and robust solution to the task, how its performance compares with that of the reference controller, and whether it can discover behaviors beyond those built into the rule-based approach.

The resulting MARL-based controller was able to locate and destroy the target with a success rate of 88%, while the ideal reference controller achieved 100%. The difference can be attributed to the agents not learning an effective search pattern and thus occasionally failing to locate the target. Under jamming, the MARL-based controller maintained a success rate of 78%, demonstrating robustness. Although trained only against a static target, the controller retained a 5% success rate against a moving target. The training did not yield novel tactics, but it did produce emergent evasive behavior. The results indicate that MAPPO can successfully train a large number of agents for a challenging, two-phase long-horizon task. At the same time, they reveal clear limitations in tactical refinement and search behavior.

Keywords Multi-agent reinforcement learning, MAPPO, naval warfare, autonomous vehicle, unmanned surface vehicle, swarming

Aalto-yliopisto
Perustieteiden
korkeakoulu

Tekijä Niki Leskinen

Työn nimi Moniagenttinen vahvistusoppiminen autonomisten miehittämättömien pinta-alusten kohteen etsintä- ja torjuntatehtävissä

Koulutusohjelma Matematiikka ja operaatiotutkimus

Pääaine Systeemi- ja operaatiotutkimus

Työn valvoja Prof. Kai Virtanen

Työn ohjaaja Lauri Vasankari

Päivämäärä 23.4.2026 **Sivumäärä** 78 **Kieli** englanti

Tiivistelmä

Miehittämättömien pinta-alusten (USV) merkittävyys merisodankäynnissä on kasvanut Ukrainan sodassa. Aluksia voidaan ohjata etäyhteydellä tai ne voivat toimia eri autonomiatasoilla yksin tai osana parvea. Parvi voidaan määritellä koordinoidusti toimivana joukkona, jonka päätöksenteko on hajautettu. Koneoppimisen kehittyminen sekä aluksiin integroidun laskentatehon kasvu mahdollistavat korkeamman autonomian ja kehittyneemmän parveilun USV-aluksille. Yksi tapa saavuttaa autonominen parveilu on moniagenttinen vahvistusoppiminen (MARL). MARL on koneoppimisen osa-alue, jossa joukko agentteja oppii yrityksen ja erehdyksen kautta valitsemaan toimintoja, jotka maksimoivat ympäristön antaman palkinnon. Sitä on sovellettu robotiikassa ja tutkittu USV-alusten ohjaamisessa. Työssä tarkastellaan MARL-menetelmän soveltamista USV-parven ohjaamiseen haastavassa kaksivaiheisessa tehtävässä, jossa parven tavoitteena on löytää ja tuhota tuhoaviin torjuntakeinoihin sekä signaalihäirintään kykenevä vastapuolen pinta-alus. Työn kirjallisuuskatsaus käsittelee MARL-menetelmää, parveilua, merisodankäyntiä ja USV-aluksia. Tämän jälkeen toteutetaan MAPPO-algoritmiin (multi-agent proximal policy optimization) perustuva koulutuskehys. Tuloksena saatua ohjainta arvioidaan vertaamalla sitä ideaaliin sääntöpohjaiseen vertailuohjaimeen, jolla on täysi informaatio kohteesta. Päämääränä on arvioida, voiko MARL tarjota tehtävään toimivan ja robustin ratkaisun, miten sen suorituskyky vertautuu vertailuohjaimeen, sekä oppiiko se taktiikoita, joita ei ole rakennettu sisään sääntöpohjaiseen ohjaimeen. MARL-pohjainen ohjain onnistui löytämään ja tuhoamaan kohteen 88 % onnistumisprosentilla, kun vertailuohjaimella vastaava luku oli 100 %. Ero selittyy pitkälti sillä, etteivät agentit oppineet tehokasta etsintätapaa, jolloin kohde jäi välillä löytymättä. Signaalihäirinnän alaisena MARL-pohjaisen ohjaimen onnistumisprosentti säilyi 78 %:ssa, osoittaen robustisuutta. Ohjain koulutettiin vain staattista kohdetta vastaan, mutta saavutti liikkuvaa kohdetta vastaan 5 %:n onnistumisprosentin. Koulutus ei tuottanut uusia taktiikoita, mutta synnytti emergenttiä väistöliikehdintää. Tulokset osoittavat MAPPO:n pystyvän kouluttamaan suuren parven toteuttamaan haastava kaksivaiheinen pitkän aikahorisontin tehtävä, mutta osoittaen rajoitteita agenttien taktisessa toiminnassa ja etsinnän oppimisessa.

Avainsanat moniagenttinen vahvistusoppiminen, MAPPO, merisodankäynti, autonominen pinta-alus, parveilu

Use of AI

Large language models (ChatGPT 5.2-5.4) were used to assist in the preparation of this thesis. These tools were used for some text rephrasing, drafting LaTeX tables, and supporting the development of the simulator code.

Preface

I wish to thank Professor Kai Virtanen and my advisor, Lauri Vasankari, for the opportunity to work on a challenging yet exceptionally interesting and meaningful topic. I am also grateful to Aalto University and to all its people for creating such a vibrant academic community. Most importantly, I want to thank Anniina for all the love, support, and patience throughout this process. Finally, I wish to thank Steely Dan for the music.

Helsinki, April 23, 2026

Niki Leskinen

Contents

Abstract 3

Abstract (in Finnish) 4

Use of AI 5

Preface 6

Contents 7

Notation and abbreviations 9

1 Introduction 11

2 Multi-Agent Reinforcement Learning 13
2.1 Reinforcement Learning . . . 13
2.1.1 Optimal Value Functions . . . 14
2.1.2 Characteristics of Reinforcement Learning . . . 16
2.1.3 Dynamic Programming . . . 16
2.1.4 Monte Carlo Methods . . . 17
2.1.5 Temporal-Difference Methods . . . 18
2.1.6 Partial Observability . . . 20
2.1.7 Function Approximation and Deep Reinforcement Learning 20
2.2 Multi-Agent Reinforcement Learning . . . 22
2.2.1 Stochastic Games . . . 23
2.2.2 New Challenges for Learning in Multi-Agent Systems . . . 24
2.2.3 Centralized Training with Decentralized Execution . . . 24
2.2.4 Multi-Agent Deep Reinforcement Learning . . . 25
2.2.5 Multi-Agent Proximal Policy Optimization . . . 27

3 Swarming and Naval Warfare 29
3.1 Swarms and Hunting Algorithms . . . 29
3.1.1 Theoretical Background . . . 29
3.1.2 Engineered Swarms . . . 30
3.2 Hunting Algorithms . . . 32
3.3 Naval Warfare and Autonomous USVs . . . 33
3.3.1 Naval Warfare . . . 33
3.3.2 Autonomous USVs . . . 33
3.3.3 Countermeasures Against USVs . . . 35
3.4 MARL for Autonomous Drones . . . 35

4 Reinforcement Learning Environment **38**
4.1 Simulation Environment 38
4.1.1 Environment and Episode Structure 38
4.1.2 USV Agents 39
4.1.3 Target Ship 41
4.2 MARL Formulation 42
4.2.1 Dec-POMDP 42
4.2.2 MAPPO Algorithm 43
4.2.3 Action, Observation, and State Spaces 45
4.2.4 Action Repeat 46
4.2.5 Reward Function 47
4.2.6 Agent Inactivity and Death Masking 51
4.2.7 Neural Network Architecture 52
4.2.8 Hyperparameter Search and Training 55

5 Results **58**
5.1 Training 58
5.2 Evaluation 59
5.3 Discussion 63

6 Summary **65**

References **67**

Notation and abbreviations

Notation

$\doteq$	equal by definition
$\approx$	approximately equal
$:=$	definition
$\leftarrow$	assignment
$\propto$	proportional to
$\odot$	element-wise multiplication
π	policy
π_*	optimal policy
$v(s)$	state-value function
v_*	optimal state-value function
$q(s,a)$	action-value function
q_*	optimal action-value function
V	estimate of state-value function v
Q	estimate of action-value function q
r	reward
R_t	reward at time t
a	action
A_t	action at time t
s	state
S_t	state at time t
t	discrete time step
T	terminal time step of an episode
$T(t)$	terminal time step of the episode that includes time step t
G_t	return at time t

Abbreviations

A2C	advantage actor critic
ACO	ant colony optimization
APF	artificial potential field
CIWS	close-in weapon system
CTDE	centralized training with decentralized execution
Dec-POMDP	decentralized partially observable Markov decision process
DP	dynamic programming
DRL	deep reinforcement learning
EW	electronic warfare
GRU	gated recurrent unit
IGM	individual-global-max
MAA2C	multi-agent advantage actor critic
MAPPO	multi-agent proximal policy optimization
MARL	multi-agent reinforcement learning
MADRL	multi-agent deep reinforcement learning
MC	Monte Carlo
MDP	Markov decision process
ML	machine learning
MLP	multilayer perceptron
PBRS	potential-based reward shaping
POMDP	partially observable Markov decision process
PPO	proximal policy optimization
POSG	partially observable stochastic game
RL	reinforcement learning
TD	temporal difference
USV	unmanned surface vehicle
VDN	value decomposition network

1 Introduction

The military significance of unmanned vehicles, sometimes referred to as drones, has increased substantially during the Russo-Ukrainian war and subsequent conflicts. Ukraine's military leadership stated already in May 2024 that "drones kill more soldiers on both sides than anything else" [1]. Three prominent categories of unmanned vehicles are unmanned aerial vehicles (UAVs), unmanned ground vehicles (UGVs), and unmanned surface vehicles (USVs), which operate on the surface of water. USVs and other unmanned vehicles can be operated remotely or with different levels of autonomy, either individually or as part of a swarm.

Traditional approaches to controlling groups of unmanned vehicles have often relied on predefined rules and engineered algorithms. Such methods can be effective in structured settings, but they are often rigid and may have difficulties in adapting to changing environments or mission objectives [2]–[5]. Recent advances in machine learning (ML) and onboard computing have made it increasingly feasible to equip unmanned vehicles with more autonomous capabilities. In this context, ML can be used for tasks such as computer vision, sensor fusion, target recognition, path planning, and collision avoidance by processing data from sensors such as cameras, lidar, radar, and sonar [6]. Beyond controlling individual vehicles, these capabilities can also support coordination within groups of vehicles, enabling more autonomous and decentralized swarm behavior [6].

One ML approach for such control problems is reinforcement learning (RL). RL is a sub-area of machine learning in which an agent learns through trial and error to choose actions that maximize rewards provided by the environment [7]. RL has many applications in robotics, ranging from helicopter flight controllers to humanoid robots [7], and it has also been explored for the control of USVs [8]. Multi-agent reinforcement learning (MARL) extends RL to settings with multiple agents that may cooperate or compete. In cooperative tasks, MARL offers a way to learn coordinated behavior directly from interaction with the environment rather than specifying all behavior in advance. A prominent example of RL discovering effective strategies is AlphaGo Zero, which learned to play Go through self-play and surpassed its predecessor AlphaGo [9]. This motivates the question of whether RL-based methods could also discover useful behaviors in complex multi-agent control tasks.

This thesis investigates the applicability of MARL to controlling a swarm of autonomous USVs in a naval hunting mission. The mission is a challenging two-phase mission in which the swarm must locate and destroy an adversarial target warship while facing defensive measures such as hard-kill systems and jamming. The thesis builds on earlier work: in [10], a naval warfare simulator was developed for applying MARL in the context of littoral warfare. Moreover, in [11] a naval warfare simulation environment was used for assessing the effectiveness of USVs against conventional crewed warships. In this thesis, the simulation environment is further developed and a training framework based on multi-agent proximal policy optimization (MAPPO) is designed and implemented for the task. The resulting controller is then evaluated against an idealized rule-based reference controller designed using domain knowledge, and given full knowledge of the target's position. The aim is to assess whether MARL

can provide a feasible and robust solution to the task, how its performance compares with that of the reference controller, and whether it can discover behaviors beyond those built into the rule-based approach.

The remainder of the thesis is organized as follows. Section 2 reviews the theoretical foundations of RL and MARL, including key concepts in value-based and policy-based learning, partial observability, centralized training with decentralized execution, and the MAPPO algorithm. Section 3 surveys other than MARL-based approaches to swarming and hunting algorithms. Additionally, it surveys naval warfare, USVs, and previous research on MARL for unmanned autonomous vehicles. Section 4 presents the RL environment used in this thesis and describes the MARL formulation in detail, including the simulation environment, the Dec-POMDP formulation, the MAPPO-based training setup, the action, observation, and state spaces, action repeat, the reward function, agent inactivity and death masking, the neural network architecture, and the hyperparameter search and training procedure. Section 5 presents the training and evaluation results of the base case scenario, jamming scenario, and moving target scenario, as well as comparisons against the rule-based reference controller. Finally, Section 6 summarizes the main findings and conclusions of the thesis.

2 Multi-Agent Reinforcement Learning

Multi-agent reinforcement learning (MARL) essentially applies reinforcement learning (RL) to multi-agent game models in order to learn optimal policies, i.e. decision-making strategies, for the agents [12]. RL is one of the main sub-areas of machine learning where an intelligent agent is taught by trial and error to act based on the environment state in order to maximize the expected long-term reward signal [13]. This section surveys the technical foundations and recent developments in MARL and places it in a wider context of machine learning, optimal control, and game theory.

2.1 Reinforcement Learning

Markov Decision Process (MDP) is a common framework for solving sequential decision-making problems and is often used as a mathematical form of RL tasks [13]. Within the framework, the learner and decision maker is called the agent, and everything the agent interacts with, is called the environment [14]. The agent and the environment interact continually with each other: the agent selects actions and the environment responds to these actions and presents new situations (states) to the agent [14]. The environment gives rewards to the agent, which the agent attempts to maximize over time by the choice of its actions [14]. The feedback loop of the agent-environment interaction is pictured in Figure 1: at each time step after observing the environment, the agent chooses an action with respect to its policy, and after the execution of the action, the environment gives a reward signal to the agent and updates to a new state [13]. The objective of the agent is to find an optimal policy that maximizes the expected cumulative reward over time.

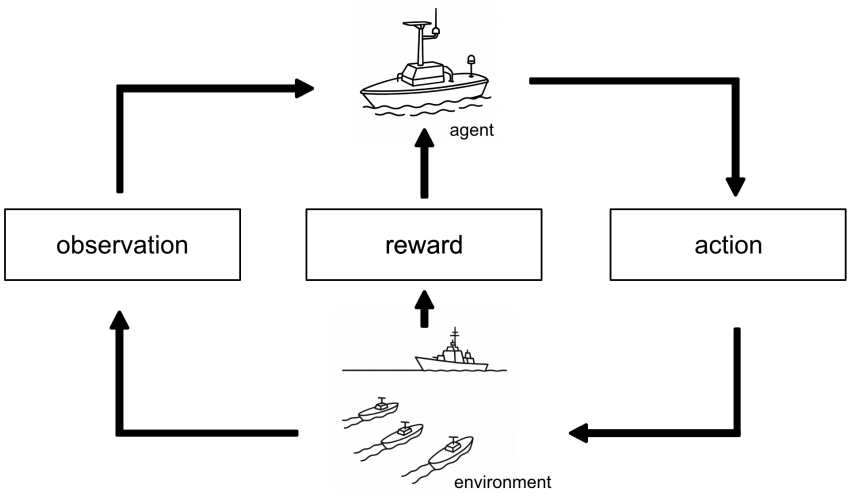


Figure 1: Agent-environment interaction loop in an MDP.

MDP assumes a Markovian environment, meaning the state at the next time step only depends on the current state, and that the environment is fully observable, meaning the agent can observe all information within the environment at any time step [13].

The latter assumption is often impossible to satisfy in many domains, hence there are variants of MDP such as partially observable MDP [15] [13].

MDP can be formulated as a quintuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$, where $\mathcal{S}$ is a set of all observable states, $\mathcal{A}$ is a set of discrete or continuous actions, $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the state transition function, $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ (or sometimes $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$) is the immediate scalar reward, and $\gamma \in [0, 1]$ is a discount factor of future rewards [14]. If a state s changes, an agent samples the next action from a probability distribution called the policy: $\pi(a \mid s)$ [14]. The state-action trajectory obtained by the interaction can be expressed as $\tau : (s_0, a_0, s_1, ..., a_{t-1}, s_t)$ [13].

2.1.1 Optimal Value Functions

RL tasks under an MDP seek policies that maximize the cumulative reward along trajectories. The discounted return from time t is

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}, \tag{1}$$

where R_{t+1} denotes the immediate reward at time $t+1$ and $\gamma \in [0, 1]$ is the discount factor [14]. Common problem classes are episodic tasks and continuing tasks. In episodic tasks the agent-environment interaction breaks into episodes that terminate in a predetermined terminal state for which an undiscounted finite-horizon return is a viable choice,

$$G_t = \sum_{k=0}^{T-t-1} R_{t+1+k}, \tag{2}$$

where T is the termination time [14]. In continuing tasks the interaction proceeds indefinitely without a natural terminal time, requiring the discounted return to be used in order to ensure finiteness and to formalize a preference for near-term rewards [14].

Discounting induces a present value for delayed outcomes: a reward k steps in the future contributes γ^k times what it would contribute if received immediately. When $\gamma = 0$, the agent is short-sighted and optimizes R_{t+1} only, but as $\gamma \to 1$, the agent becomes increasingly far-sighted and future rewards weigh more heavily [14]. For any bounded reward sequence, the infinite sum defining G_t is finite whenever $\gamma < 1$ [14]. In episodic tasks the return also satisfies the one-step recursion (with $G_T = 0$ at termination),

$$G_t = R_{t+1} + \gamma\, G_{t+1}, \tag{3}$$

and the same identity holds in continuing tasks when the discounted definition is used [14].

Calculating the return of a state is challenging due to the dependence on both the state-transition probabilities and the policy [13]. The weighted average over all trajectories can be expressed via the Bellman equations [13], [16], [17]. The expected return of a state s under a fixed policy π can be determined formally with the state-value

function for policy π:

$$v_\pi(s) \doteq \mathbb{E}_\pi\left[G_t \mid S_t = s\right] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+1+k} \,\middle|\, S_t = s\right]$$
$$= \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma v_\pi(s')\right], \quad \text{for all } s \in \mathcal{S}, \tag{4}$$

where $\mathbb{E}_\pi[\cdot]$ is the expected value of a random variable given that the agent chooses its actions accordingly to policy π, and t is any time step [14]. Equation (4) is the Bellman equation for v_π, which expresses a relationship between the value of a state s and the values of the successor states of s [14]. The expected return of taking action a in state s under π can be formally determined similarly with the action-value function for policy π:

$$q_\pi(s,a) \doteq \mathbb{E}_\pi\left[G_t \mid S_t = s, A_t = a\right] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s, A_t = a\right]$$
$$= \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma\, v_\pi(s')\right]$$
$$= \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma \sum_{a'} \pi(a' \mid s')\, q_\pi(s', a')\right], \text{ for all } (s,a) \in \mathcal{S} \times \mathcal{A}, \tag{5}$$

where equation (5) is the Bellman equation for q_π [14].

Solving a reinforcement learning task means finding an optimal policy, which is a policy that yields an expected return that is higher or equal to any other possible policy [14]. There can be more than one optimal policies, but all optimal policies are denoted by π_*, and they share the same optimal state-value function v_*, defined as

$$v_*(s) \doteq \max_\pi v_\pi(s), \tag{6}$$

for all $s \in \mathcal{S}$ [14]. Optimal policies also share a common optimal action-value function q_*, which is defined as

$$q_*(s,a) \doteq \max_\pi q_\pi(s,a), \tag{7}$$

for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$ [14]. Since $q_*(s,a)$ gives the expected return for state-action pair (s,a) and thereafter follows the optimal policy, q_* can be written in terms of v_* [14]:

$$q_*(s,a) = \mathbb{E}\left[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a\right]. \tag{8}$$

The Bellman optimality equations (9), (10), and (11) allow v_* and q_* to be written

without reference to any specific policy [14]:

$$
\begin{aligned}
v_*(s) &= \max_{a \in \mathcal{A}(s)} q_{\pi_*}(s,a) \\
&= \max_a \mathbb{E}\left[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a\right] \qquad (9) \\
&= \max_a \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma v_*(s')\right] \qquad (10)
\end{aligned}
$$

$$
\begin{aligned}
q_*(s,a) &= \mathbb{E}\left[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \mid S_t = s, A_t = a\right] \\
&= \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma \max_{a'} q_*(s', a')\right]. \qquad (11)
\end{aligned}
$$

In essence, the Bellman optimality equation states that, under an optimal policy, a state's value equals the expected return obtained by taking the best available action in that state [14].

2.1.2 Characteristics of Reinforcement Learning

An unsolved dilemma in RL is the trade-off between exploration and exploitation: in order to obtain high rewards, an agent needs to exploit actions that are tried in the past and have yielded reward effectively, but to be able to find those actions in the first place, the agent must explore actions it hasn't performed before [14]. The balance between these two in changing the policy π can be determined with the exploration parameter $\epsilon \in [0, 1]$: for instance, in so-called ϵ-greedy action selection, the action with the greatest $q(s, a)$ is selected with probability $1 - \epsilon$, and other random actions are selected with the probability $\epsilon / N(a)$, where $N(a)$ is the number of possible actions [14].

The two prominent ways of learning are on-policy and off-policy methods [13]. On-policy methods aim to assess or improve the policy that is used for decision-making, hence they optimize the policy directly, while off-policy methods assess or improve a policy that is different than the one that generated the data, having higher data efficiency than on-policy methods [13], [14]. Depending on the environment, there are various methods that can be used to evaluate the optimal values. Dynamic programming (DP) methods can be used to solve the Bellman equations if the environment is fully known [17] [13], while Monte Carlo (MC) methods are useful when the environment cannot be fully modeled [13]. Temporal difference (TD) methods are similar to MC methods, and do not require a complete dynamics model of the environment, but update from sampled returns: TD bootstraps from one-step predictions, whereas MC waits for episode termination [14].

2.1.3 Dynamic Programming

Dynamic programming (DP) is a family of algorithms for computing value functions and optimal policies in MDPs by iteratively using Bellman equations as operators to estimate the value function for current policy v_π and optimal policy π_* [12], [14], [16],

[17]. DP assumes full knowledge of the MDP model, including the state transition probabilities and reward function [12]. Iterative policy evaluation is a DP method that first initializes a vector $v_0(s) = 0$ for all $s \in \mathcal{S}$ and iteratively performs update sweeps for all states $s \in \mathcal{S}$ using the Bellman equation

$$v_{k+1}(s) \leftarrow \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a) \left[r + \gamma v_k(s')\right] \tag{12}$$

until convergence to v_π [12]. This update uses current estimates of successor states s' to update s, which is a common property in many RL algorithms called bootstrapping [12]. Now that there is a method for policy evaluation, we need to consider a method for policy improvement. Greedy policy improvement selects for each s,

$$\pi'(s) \in \arg\max_{a \in \mathcal{A}} q_\pi(s, a), \tag{13}$$

and by the policy improvement theorem [14]

$$\sum_{a \in \mathcal{A}} \pi'(a \mid s) q_\pi(s, a) \geq \sum_{a \in \mathcal{A}} \pi(a \mid s) q_\pi(s, a) \tag{14}$$

$$= v_\pi(s) \tag{15}$$

we obtain $v_{\pi'}(s) \geq v_\pi(s)$ for all s [12]. Alternating evaluation and greedy improvement yields policy iteration, which reaches an optimal policy π_* and the associated v_* in finitely many iterations for finite MDPs [12]. An equivalent one-sweep scheme called value iteration combines evaluation and improvement by applying the Bellman optimality operator [12]:

$$v_{k+1}(s) \leftarrow \max_{a \in \mathcal{A}} \sum_{s', r} p(s', r \mid s, a) \left[r + \gamma\, v_k(s')\right]. \tag{16}$$

In practice, both policy iteration and value iteration are stopped when successive updates change v by less than a predetermined tolerance. Value iteration and policy iteration are the mathematical foundations that model-free methods emulate: iterative expected updates are replaced by sample-based bootstrapping in TD or episode-averaging in MC methods when state transition probabilities are unknown, while preserving the same fixed points and improvement principles [14].

2.1.4 Monte Carlo Methods

Monte Carlo (MC) methods estimate value functions directly from sampled experience without a model by averaging complete returns along episodes [14]. In their basic form, MC methods are defined for episodic tasks, so that returns are well-defined. A first-visit or every-visit estimator for v_π averages the empirical returns after visits to s under π:

$$V(s) \approx \frac{1}{N(s)} \sum_{i=1}^{N(s)} G^{(i)}(s), \quad G_t = \sum_{k=0}^{T-t-1} \gamma^k R_{t+1+k}, \tag{17}$$

where $N(s)$ is the visit count and T is the termination time [14]. MC estimates are unbiased and converge with increasing visits, but typically exhibit higher variance and are episode-by-episode incremental [14]. In addition, MC does not use bootstrapping, meaning the updates use sampled returns rather than other learned estimates [14].

When no model is available, action-value estimation $Q_\pi(s, a)$ is preferred so policies can be improved without one-step lookahead: first-visit and every-visit MC extends by averaging returns after visits to (s, a) [14]. MC control follows generalized policy iteration (GPI) [14]. In GPI, the action-value function q is constantly altered to better approximate the value function for the current policy q_π, and the policy π is constantly enhanced with respect to the current value function q [14]. With so called exploring starts, greedy improvement is guaranteed to improve or match the current policy, and without exploring starts, on-policy control uses ϵ-soft (for instance ϵ-greedy) policies to ensure all actions remain sampled [14].

To evaluate a fixed target policy π from episodes generated by a behavior policy b, coverage ($\pi(a \mid s) > 0 \Rightarrow b(a \mid s) > 0$) is assumed and returns are reweighted by the importance-sampling ratio [14]:

$$\rho_{t:T-1} \doteq \prod_{k=t}^{T-1} \frac{\pi(A_k \mid S_k)}{b(A_k \mid S_k)}. \tag{18}$$

For every-visit MC prediction, the ordinary estimator averages $\rho_{t:T-1} G_t$ over visits to s and is unbiased for $v_\pi(s)$, but can have very high or even unbounded variance [14]. The weighted importance sampling used to estimate $v_\pi(s)$, defined as

$$V(s) \doteq \frac{\sum_{t \in \mathcal{T}(s)} \rho_{t:T(t)-1} \, G_t}{\sum_{t \in \mathcal{T}(s)} \rho_{t:T(t)-1}} \tag{19}$$

is biased, yet the bias vanishes asymptotically and it typically has dramatically lower variance: under bounded returns its variance converges to zero even when the ratio's variance is infinite [14].

2.1.5 Temporal-Difference Methods

Temporal-Difference (TD) [18] methods solve the prediction problem from sampled experience by updating state values after one step of interaction, combining MC's sampling with DP's bootstrapping [14]. Given experience under a fixed policy π, TD(0) updates the current estimate immediately upon observing $(S_t, A_t, R_{t+1}, S_{t+1})$:

$$V(S_t) \leftarrow V(S_t) + \alpha \big[R_{t+1} + \gamma \, V(S_{t+1}) - V(S_t) \big], \tag{20}$$

where the term in brackets

$$\delta_t \doteq R_{t+1} + \gamma \, V(S_{t+1}) - V(S_t) \tag{21}$$

is the TD error, and α is a constant step-size parameter [14]. Unlike MC, which must wait until the episode return G_t is known, TD(0) uses the one-step target $R_{t+1} + \gamma V(S_{t+1})$

and updates online and incrementally at every transition [14]. Generally, MC targets correspond to the recursion $v_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s]$, while TD(0) mirrors the Bellman expectation equation $v_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s]$ but substitutes V for v_π [14]. This yields stepwise updates with lower variance suitable for long or continuing tasks where episode-end returns are delayed or undefined [14].

Under standard stochastic-approximation conditions, tabular TD(0) converges to v_π for a fixed policy π (with probability 1 for diminishing stepsizes, and in the mean for sufficiently small α) [14]. Moreover, when V is held fixed during an episode, the MC error $G_t - V(S_t)$ decomposes as a discounted sum of TD errors, $G_t - V(S_t) = \sum_{k=t}^{T-1} \gamma^{k-t}\delta_k$, linking TD's local one-step targets to global returns [14].

An early breakthrough in RL was the development of Q-learning, which is an off-policy TD control algorithm [14], [19], defined by

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\right] \tag{22}$$

where α is the learning rate. The learned action-value function Q directly approximates the optimal action-value function q_* independently of the policy being followed, simplifying the analysis of the algorithm and enabling early convergence proofs. However, Q-learning has its limitations: Thrun and Schwartz [20] raised the problem of overestimated Q value in Q-learning algorithm, and Deepmind [21] further proved that any kind of error, whether environmental noise, function approximation, non-stationarity, or any other reason, would lead into overestimation of the Q value [13]. Double Q-learning has been proposed as a solution to the issue of overestimation: it stores two Q functions, Q^A and Q^B and alternates which one it updates [22]. Instead of using the value $Q^A(s', a^*) = \max_a Q^A(s', a)$ to update Q^A as in Q-learning, the value $Q^B(s', a^*)$ is used [22]. Because Q^B is trained on the same task but from a different stream of experience, this cross-evaluation reduces the overestimation that arises when selection and evaluation use the same values [22]. Q^B is updated similarly using b^* and Q^A [22].

Sarsa is a prominent on-policy TD control method that applies generalized policy iteration using TD prediction for the current behavior policy [14]. It learns action values $q_\pi(s, a)$ for the policy π by viewing the process as a Markov chain over state-action pairs and updates after every transition from a nonterminal state S_t:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\right], \tag{23}$$

and if S_{t+1} is terminal, $Q(S_{t+1}, A_{t+1})$ is defined as zero [23] [14]. The update rule uses all elements in the $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$ quintuple of events, giving the name to the algorithm [14]. Sarsa converges to an optimal policy and action-value function with probability 1, given that all state-action pairs are visited infinitely often, and the exploration parameter ϵ decays so that π converges to the greedy policy, for instance by setting $\epsilon_t = 1/t$ [14].

2.1.6 Partial Observability

As mentioned earlier, an MDP assumes a Markovian environment and full observability: one can determine the next action by evaluating $\pi_*(s)$ at the current state s. In many real-world scenarios, however, the agent cannot directly observe the true state. In such cases, the problem is modeled as a partially observable Markov decision process (POMDP) [15]. In the classic formulation, a POMDP is a sextuple $(\mathcal{S}, \mathcal{A}, T, R, \Omega, O)$, where $\mathcal{S}, \mathcal{A}, T$, and R define the underlying MDP; Ω is a finite set of possible observations; and the observation function $O : \mathcal{S} \times \mathcal{A} \to \Pi(\Omega)$ maps each resulting state-action pair to a probability distribution over observations [15]. $O(s', a, o)$ denotes the probability of observing o after taking action a and arriving in state s', i.e. $O(s', a, o) = O(s', a)(o)$ [15]. Within an MDP, the agent's policy π selects actions as a function of the fully observed state. Under partial observability, the agent maintains an internal belief state b, which is a probability distribution over $\mathcal{S}$ that summarizes its history of actions and observations [15]. Thus, the controller is made of two components: a state estimator SE that updates b given the last action and current observation, and a policy π that maps the current belief b to an action [15]. The belief state is a sufficient statistic of the past in the sense that when correctly computed, no additional history improves decision quality beyond b [15]. This yields a Markov process over belief states and enables planning as if the belief was the state [15].

2.1.7 Function Approximation and Deep Reinforcement Learning

Tabular representations break down when state or action spaces are large or continuous [12], [13]. A standard approach in these cases is to represent the approximate value function as a parameterized functional form with weight vector $\mathbf{w} \in \mathbb{R}^d$, which can be written as $\hat{v}(s, \mathbf{w}) \approx v_\pi(s)$ [14]. For instance, $\hat{v}$ can be a linear function with $\mathbf{w}$ being the vector of feature weights, or $\hat{v}$ may be a function computed by a multi-layer neural network with $\mathbf{w}$ being the vector of connection weights in the layers [14]. Methods that use function approximation apply equally well to Markovian environments and partially observable environments [14]. It enables generalization across states and compact representations of complex policies, but introduces stability challenges when combined with bootstrapping and off-policy learning (the so-called "deadly triad") [14], [24].

Deep learning has become increasingly influential with higher computing power and the advances in deep neural network technologies such as convolutional neural network (CNN) and recurrent neural network (RNN), showcasing high fitting and representation capability when processing high-dimensional data [13]. Deep reinforcement learning (DRL) utilizes these qualities of neural networks to approximate values or policies and solve RL problems with a large state space and continuous action space [13]. Generally, deep learning can be used to map original states of the environment to features, and RL is then used to map the features to actions [13]. DRL can consider these two processes as a whole due to the ability to use the deep neural network as a black box [13].

There are two broad families of DRL methods: value-based and policy-based.

A standard foundation for DRL algorithms is a value-based method called deep Q-learning (DQN) which approximates $q(s, a)$ with Q-learning updates, but substitutes the tabular value function with a neural network [12], [13]. However, similar to Q-learning, DQN exhibits a tendency of overestimating the Q-value, to which double deep Q network (DDQN) has been proposed as a solution: it decomposes DQN's single value function used to select actions and calculate target values into two different value functions, i.e. networks, with the main network being used to select the optimal action and the target network being used to estimate the value function [12], [13], [21].

Policy-based methods are based on policy gradient algorithms, which are able to optimize the policy directly by computing gradients with respect to the parameters of their policy in order to update the learned policy [12], [13]. Gradient-based optimization requires a differentiable representation of the policy and a loss function to compute gradients for updating the policy's parameters [12]. Minimizing a loss function should correspond to improving the policy, which may be measured via $V_\pi(s)$ or $Q_\pi(s, a)$, but changing the policy to maximize such values alters both action choices and the state visitation distribution, coupling returns to unknown environment dynamics [12]. The policy gradient theorem [14] describes a solution to these challenges [12]. For episodic tasks policy gradient can be written as

$$\nabla_\phi J(\phi) \propto \sum_{s \in \mathcal{S}} \Pr(s \mid \pi) \sum_{a \in \mathcal{A}} q_\pi(s, a) \nabla_\phi \pi(a \mid s; \phi), \tag{24}$$

where J is the maximizable objective function, i.e. quality of the policy, over parameter ϕ of parameterized policy π, $\propto$ denotes "proportional to", and $Pr(s \mid \pi)$ is the on-policy state distribution of π [12]. The probability of being in state s at time t while following π obeys

$$\Pr(S^t = s \mid \pi) = \begin{cases} \mu(s), & t = 0, \\ \sum_{s'} \Pr(S^{t-1} = s' \mid \pi) \sum_a \pi(a \mid s') \mathcal{T}(s \mid s', a), & t > 0, \end{cases} \tag{25}$$

where $\mu(s)$ is the distribution over initial states and $\mathcal{T}(s' \mid s, a)$ is the environment transition kernel [12]. The above allows the definition of the discounted state-occupancy measure

$$\rho(s \mid \pi) = \sum_{t=0}^{\infty} \gamma^t \Pr(S^t = s \mid \pi), \tag{26}$$

which satisfies the recursion

$$\rho(s \mid \pi) = \mu(s) + \gamma \sum_{s'} \rho(s' \mid \pi) \sum_a \pi(a \mid s') \mathcal{T}(s \mid s', a) \tag{27}$$

[12]. Normalizing ρ over states yields the on-policy distribution used in (24) [12]:

$$\Pr(s \mid \pi) = \frac{\rho(s \mid \pi)}{\sum_{s'} \rho(s' \mid \pi)}. \tag{28}$$

Within policy gradient algorithms, actor-critic algorithms are a notable family of algorithms that train simultaneously a parameterized policy called the actor and a value

function called the critic [12]. Gradient estimates derived from the policy gradient theorem are used to optimize the actor, while the critic is used to form bootstrapped return estimates [12]. Bootstrapping enables estimations from the experience of a single step similarly to TD methods, yielding lower variance compared to MC estimates [12]. A notable actor-critic method is the advantage actor-critic (A2C) [25], which provides estimates of the advantage, i.e. how much better action a is than the policy's average choice in state s, given by $Adv_\pi(s,a) = q_\pi(s,a) - v_\pi(s)$, to guide the policy gradients [12]. With most policy gradient algorithms, there is a risk that any individual gradient update step can move a policy too far in the parameter space and reduce its expected performance [12]. Trust-region methods, such as trust region policy optimization, mitigate this risk by defining an area within the space of policy parameters in which the policy's updates are constrained [12]. Proximal policy optimization (PPO) [26] maintains the notion of trust regions while using a simple, efficient surrogate objective that enables multiple updates per data batch [12]. Given a behavior policy π_β and the current policy $\pi(\cdot \mid s;\phi)$ to be optimized, PPO forms an importance sampling weight

$$\rho(s,a) = \frac{\pi(a \mid s;\phi)}{\pi_\beta(a \mid s)}, \tag{29}$$

which adjusts the data distribution to enable the data generated by π_β appear on-policy for π [12]. PPO updates the actor by minimizing the clipped surrogate loss

$$\mathcal{L}(\phi) = -\min\left(\rho(s^t,a^t)\,Adv(s^t,a^t), \text{clip}(\rho(s^t,a^t),\ 1-\epsilon,\ 1+\epsilon)Adv(s^t,a^t)\right), \tag{30}$$

which limits the effective policy change by bounding ρ to $[1-\epsilon, 1+\epsilon]$ [12]. The critic is trained by regression to a bootstrapped target,

$$y^t = \begin{cases} r^t, & \text{if } s^{t+1} \text{ is terminal,} \\ r^t + \gamma V(s^{t+1};\theta), & \text{otherwise,} \end{cases} \tag{31}$$

with $\mathcal{L}(\theta) = \left(y^t - V_(s^t;\theta)\right)^2$ being the critic loss[12].

2.2 Multi-Agent Reinforcement Learning

MARL extends single-decision-maker RL to game models with multiple decision-making agents whose actions jointly influence the dynamics and rewards. The agents coordinate their actions with each other (often called a cooperative setting) or against each other (competitive setting), and sometimes the scenario is a combination of these two [12]. Due to the research setting of this thesis, we focus on literature that is relevant for cooperative scenarios. In this subsection we formalize the multi-agent generalization of an MDP, present an overview of the training and execution modes and a foundational multi-agent deep reinforcement learning algorithm, and finally survey the latest research in addressing common issues in complex MARL settings such as communication and credit assignment.

2.2.1 Stochastic Games

The central multi-agent generalization of an MDP is a stochastic game, also known as a Markov game [27]–[29]. Similarly to MDPs, stochastic games assume history-independence and full observability. A finite discounted stochastic game with n agents is a tuple

$$(I, \mathcal{S}, \{\mathcal{A}_i\}_{i\in I}, \mathcal{T}, \{R_i\}_{i\in I}, \gamma, \mu),$$

where $I = \{1, ..., n\}$ is a finite set of agents, $\mathcal{S}$ is the state set, $\mathcal{A}_i$ is the action set of agent i; $\mathcal{T} : \mathcal{S}\times\mathcal{A}\times\mathcal{S} \to [0, 1]$ such that $\forall s \in \mathcal{S}, a \in \mathcal{A} : \sum_{s'\in\mathcal{S}} \mathcal{T}(s, a, s') = 1$ is the state transition probability function, $R_i : \mathcal{S}\times\mathcal{A}\times\mathcal{S} \to \mathbb{R}$, where $\mathcal{A} = \mathcal{A}_1 \times ... \times \mathcal{A}_n$ is the reward to agent i, $\gamma \in [0, 1)$ is the discount factor, and $\mu : \mathcal{S} \to [0, 1]$ is the initial state distribution [12], [28]. At time t, each agent observes the current state $s^t \in \mathcal{S}$ and chooses an action $a_i^t \in \mathcal{A}_i$ with a probability according to its policy $\pi_i(a_i^t \mid h^t)$, where $h^t = (s^0, a^0, s^1, a^1, ..., s^t)$ is the state-action history which is observed by all agents, i.e. full observability [12]. With the joint action a^t, the game moves to the next state $s^{t+1} \in \mathcal{S}$ with a probability determined by $\mathcal{T}(s^t, a^t, s^{t+1})$ and all agents i receive a reward $r_i^t = R_i(s^t, a^t, s^{t+1})$ [12].

Realistic multi-agent systems are rarely fully observable. Similarly to POMDPs, partially observable stochastic game (POSG) [30] introduces individual observation processes to a stochastic game. A POSG is defined by the same elements as a stochastic game, but it additionally defines for each agent $i \in I$ an observation space O_i and an observation function $O_i : \mathcal{A} \times \mathcal{S} \times O_i \to [0, 1]$ such that $\forall a \in \mathcal{A}, s \in \mathcal{S} : \sum_{o_i\in O_i} O_i(a, s, o_i) = 1$ [12]. POSGs proceed in the same manner as stochastic games, except after the environment transitions to s^t under the previous joint action a^{t-1}, each agent receives a private observation o_i [12]. Agent i's policy is history dependent, $\pi_i(a_i^t \mid h_i^t)$ with $h_i^t = (o_i^0, \ldots, o_i^t)$, to address partial information [12]. Observation functions can be used widely to represent the observability conditions, such as ability to observe actions of other agents or the limits of the view region [12]. Communication between agents can also be modeled: sending a message can be embedded into the agent's action space as an action that doesn't affect the state of the environment but can be observed by other agents, and an incoming message from another agent may be plugged into the observation function along with added noise and communication breakdowns, or constraints in communication distance [12].

The agents often share a common reward function in fully cooperative settings, i.e. $R_1 = R_2 = ... = R_n = R$, or alternatively, the agents have different reward functions but the goal is to maximize the average reward across the agents, i.e. $\bar{R}(s, a, s') := \frac{1}{n}\sum_{i\in I} R_i(s, a, s')$ for any $(s, a, s') \in \mathcal{S} \times \mathcal{A} \times \mathcal{S}$, also known as the team-average reward [28]. A central cooperative framework is the decentralized partially observable MDP (Dec-POMDP), obtained by imposing a common reward function, i.e. each agent chooses actions from $\mathcal{A}_i$ based only on its local history and seeks to maximize the team return. [12], [31].

2.2.2 New Challenges for Learning in Multi-Agent Systems

In multi-agent learning, each agent's policy evolves during training, making the environment non-stationary from any single learner's perspective [12]. This breaks assumptions behind convergence proofs for single-agent TD and may lead to cyclic dynamics where the agents adapt their policies to other agents' changing policies, which also adapt to other agents' changing policies [12]. Practical remedies that have been proposed include importance sampling of replay data, centralized training, and incorporating meta-RL considering updates from other agents [32].

Many real-world multi-agent systems involve a substantial number of agents. When the number of agents is scaled up, the number of their joint actions may grow exponentially in the number of agents: $|\mathcal{A}| = |\mathcal{A}_1| \cdot \ldots \cdot |\mathcal{A}_n|$ [12], [32]. In addition to computational challenges, increasing the number of agents may increase the level of non-stationarity of the environment due to the increase in "moving parts" in the system [12]. Some proposed solution to scalability issues include parameter sharing, where homogeneous agents share neural networks and heterogeneous agents are trained independently [33], and techniques such as transfer learning or curriculum learning [34] which initially train with fewer agents and progressively scale up to environments with a larger number of agents [32].

Credit assignment is another common issue. In both single-agent RL and MARL, there exists the problem of temporal credit assignment, i.e. determining which actions throughout the history contributed to the accumulated rewards, while in MARL, attributing rewards to specific agents or coalitions arises the non-trivial problem of multi-agent credit assignment [12]. Issues in multi-agent credit assignment in cooperative settings may lead to reinforcing actions of agents that made no contribution to the reward, since the agents do not have the ability to understand the effects of their own actions [12]. Value decomposition and attributing values to joint actions have been proposed as solutions to multi-agent credit assignment [12], as well as multi-level advantage credit assignment [35].

2.2.3 Centralized Training with Decentralized Execution

There are three main paradigms in training and execution modes [12]. Centralized training and execution is a design pattern where the information used for learning of agent policies as well as policies themselves are shared centrally between the agents, which is often not viable for real-life applications [12]. Decentralized training and execution paradigm does not use central sharing of any information or mechanics during the training or within the policies, which can be a natural choice in some domains, but lacks the ability to leverage information on other agents and may result in increased non-stationarity [12]. The third paradigm is centralized training with decentralized execution (CTDE) [12]. CTDE algorithms train agent policies centrally, but the policies are designed for decentralized execution, combining the benefits of both fully centralized and decentralized paradigms [12]. CTDE is prominently used in deep MARL because it lets value functions use privileged, joint information during training without breaking tractability [12]. A common pattern is a multi-agent

actor-critic with a centralized critic that is conditioned on joint observation histories, yielding more accurate value estimates, while decentralized actors condition only on their own histories [12]. At execution the critic isn't needed since the actions come from the local policies, thus enabling fully decentralized execution [12].

2.2.4 Multi-Agent Deep Reinforcement Learning

Tabular MARL algorithms face the same problems as tabular RL algorithms (which were discussed in section 2.1.7), hence deep learning is required, enabling training of neural networks as function approximators with the ability to generalize over large state or action spaces [12]. The notation for multi-agent deep reinforcement learning (MADRL) algorithms is from hereon simplified for the sake of clarity: $\pi(\cdot;\phi_i)$ denotes the policy of agent i with agent i's policy parameters ϕ_i, i.e. the subscript of π is left out. Similarly, $v(\cdot;\theta_i)$ is the value function for agent i with parameters θ_i, and $q(\cdot;\theta_i)$ is the action-value function of agent i with parameters θ_i.

As discussed in section 2.1.7, the policy gradient theorem is the foundation of policy gradient algorithms. Since expected returns of agents depend on the policies of all agents, the single-agent policy gradient theorem (24) can be extended to a multi-agent policy gradient theorem (for partially observable environments) [12]. For agent i, the multi-agent policy gradient theorem using histories of information can be written as [12]

$$\nabla_{\phi_i} J(\phi_i) \propto \mathbb{E}_{\substack{\hat{h}\sim \Pr(\hat{h}|\pi),\\ a_i\sim\pi_i,\ a_{-i}\sim\pi_{-i}}} \left[q_i^{\pi}\big(\hat{h}, \langle a_i, a_{-i}\rangle\big)\, \nabla_{\phi_i} \log \pi_i\big(a_i \mid h_i = \sigma_i(\hat{h}), \phi_i\big)\right]. \tag{32}$$

An actor-critic algorithm under CTDE needs to consider both the actor and the critic networks: the actor network $\pi(h_i^t;\phi_i)$ requires only the local observation history of agent i for the action selection, enabling decentralized execution [12]. Since the critic is not used at execution time, it can be centralized and conditioned on privileged information, such as the full environment state [12]. Let z denote a vector representation of this information. The centralized critic's value loss for agent i is then [12]:

$$\mathcal{L}(\theta_i) = \big(y_i - v(h_i^t, z^t;\theta_i)\big)^2 \quad \text{with} \quad y_i = r_i^t + \gamma\, v(h_i^{t+1}, z^{t+1};\theta_i). \tag{33}$$

The amount privileged information to be included in z with regard to variance and performance of learning has been discussed for instance in [12], [36], [37]: it has been argued that the critic should at minimum have access to the agent's observation history h_i^t but any other information would increase the variance in the policy gradient during training [37], however there may be a beneficial trade-off in introducing additional privileged information to help agents avoid a local optimum [12], [36].

Another variant of the critic is the centralized action-value critic, for which agent i trains an action-value function q conditioned on the agent's observation history h_i, centralized information z^t, and actions of all agents a^t [12]. The centralized action-value critic's loss is then

$$\mathcal{L}(\theta_i) = \big(y_i - q(h_i^t, z^t, a^t;\theta_i)\big)^2 \quad \text{with} \quad y_i = r_i^t + \gamma\, q(h_i^{t+1}, z^{t+1}, a^{t+1};\theta_i), \tag{34}$$

and using the critic, agent i's policy loss is

$$\mathcal{L}(\phi_i) = -q(h_i^t, z^t, a^t; \theta_i) \log \pi(a_i^t \mid h_i^t; \phi_i), \tag{35}$$

and the multi-agent policy gradient for agent i is

$$\nabla_{\phi_i} J(\phi_i) = \mathbb{E}_{a^t \sim \pi} \left[q(h_i^t, z^t, \langle a_i^t, a_{-i}^t \rangle; \theta_i) \nabla_{\phi_i} \log \pi_i(a_i^t \mid h_i^t; \phi_i) \right]. \tag{36}$$

Training and applying centralized value functions comes with a caveat: learning becomes difficult due to the joint-action space growing exponentially with the number of agents [12]. Furthermore, centralized value functions do not allow individual agents to select actions in a decentralized fashion [12]. An alternative approach in CTDE for cooperative common-reward tasks is value decomposition, which learns per-agent utility functions $q(h_i, a_i; \theta_i)$ that depend only on each agent's local observation history h_i and action a_i [12]. Utility functions can be used by individual agents to efficiently select greedy actions, and are trained jointly so that, when aggregated, they approximate the centralized action-value function [12]. A key requirement is the individual-global-max (IGM) [38] property: greedy joint actions with respect to the centralized action-value function coincide with the joint of individually greedy actions, enabling decentralized greedy execution while training toward centralized objectives [12]. Formally, let

$$\mathcal{A}^*(h, z; \theta) = \arg\max_{a \in \mathcal{A}} q(h, z, a; \theta) \tag{37}$$

denote the sets of greedy actions with respect to a decomposed centralized action-value function $q(h, z, a; \theta)$, and let

$$\mathcal{A}_i^* = \arg\max_{a \in \mathcal{A}_i} q(h_i, a_i; \theta_i) \tag{38}$$

denote set of greedy actions with respect to agent i's individual utility function $q(h_i, a_i; \theta_i)$ [12]. The IGM property is satisfied if

$$\forall a = (a_{,1}, ..., a_n) \in \mathcal{A} : a \in \mathcal{A}^*(h, z; \theta) \Leftrightarrow \forall i \in I : a_i \in \mathcal{A}_i^*(h_i, \theta_i) \tag{39}$$

holds for all complete histories $\hat{h}$ with joint-observation histories $h = \sigma(\hat{h})$, per-agent observation histories $h_i = \sigma_i(\hat{h})$, and central privileged information z [12]. The IGM property means two things: first, each agent can act greedily with regard to its own utility and these individually greedy actions coincide jointly with the greedy joint action of the decomposed centralized q [12]. Secondly, the greedy joint action needed for target computation during training can thus be obtained efficiently by taking greedy actions per agent [12]. When IGM holds, the per-agent utilities are said to factorize the centralized value function [12]. In addition to enabling decentralized execution and easier learning, these utilities also provide a natural credit-assignment signal: because each utility depends only on its agent's local history and action, but is trained to approximate the centralized value in aggregate, an agent that contributes to the common reward should see its utility increase [12].

2.2.5 Multi-Agent Proximal Policy Optimization

Multi-agent proximal policy optimization (MAPPO) was selected as the training algorithm for our setting because PPO-based methods have shown strong empirical performance in cooperative multi-agent benchmarks such as [36], [39], making MAPPO a strong baseline algorithm for cooperative MARL tasks. MAPPO follows the structure of single-agent PPO by learning a policy π_θ and a value function V_ϕ as two separate neural networks [40]. The value function is used only during training for variance reduction and it can use additional global information that is not available to the agents during execution, which allows MAPPO to follow the CTDE paradigm [40].

MAPPO is commonly implemented in settings with homogeneous agents with parameter sharing, meaning all agents share the same policy and value function parameters [40]. It uses generalized advantage estimation (GAE) to compute advantage targets [41]. For agent k at sample index i, the advantage estimate is

$$\hat{A}_i^{(k)} = \sum_{l=0}^{\infty} (\gamma\lambda)^l \, \delta_{i+l}^{(k)}, \qquad \delta_i^{(k)} = r_i^{(k)} + \gamma \, V_\phi\big(s_{i+1}^{(k)}\big) - V_\phi\big(s_i^{(k)}\big), \tag{40}$$

where $\gamma \in [0, 1]$ is the discount factor, λ controls the bias-variance tradeoff, and $s_i^{(k)}$ is the critic input for agent k [40].

The actor network π_θ maps the local observation $o_i^{(k)}$ to a categorical distribution over actions in discrete action spaces, or to the parameters of a multivariate Gaussian distribution in continuous action spaces, from which the action is sampled [40]. The actor is trained by maximizing the PPO clipped surrogate objective

$$\begin{aligned} \mathcal{L}_\pi(\theta) = \frac{1}{Bn} \sum_{i=1}^{B} \sum_{k=1}^{n} \min\Big(r_{\theta,i}^{(k)} \, \hat{A}_i^{(k)}, \, \mathrm{clip}\big(r_{\theta,i}^{(k)}, \, 1-\epsilon, \, 1+\epsilon\big) \hat{A}_i^{(k)} \Big) \\ + \sigma \, \frac{1}{Bn} \sum_{i=1}^{B} \sum_{k=1}^{n} S\Big[\pi_\theta\big(o_i^{(k)}\big)\Big], \end{aligned} \tag{41}$$

where

$$r_{\theta,i}^{(k)} = \frac{\pi_\theta\Big(a_i^{(k)} \mid o_i^{(k)}\Big)}{\pi_{\theta_{\text{old}}}\Big(a_i^{(k)} \mid o_i^{(k)}\Big)}, \tag{42}$$

B is the batch size, n is the number of agents, ϵ is the PPO clipping parameter, $S[\cdot]$ denotes policy entropy, and σ is the entropy coefficient [40].

The critic network maps the critic input to a scalar value estimate,

$$V_\phi : \mathcal{S} \to \mathbb{R}, \tag{43}$$

and is trained by minimizing the clipped value loss

$$\mathcal{L}_V(\phi) = \frac{1}{Bn} \sum_{i=1}^{B} \sum_{k=1}^{n} \max\Big[\big(V_\phi(s_i^{(k)}) - \hat{R}_i^{(k)}\big)^2, \; \big(\mathrm{clip}\big(V_\phi(s_i^{(k)}), V_{\phi_{\text{old}}}(s_i^{(k)}) - \epsilon, V_{\phi_{\text{old}}}(s_i^{(k)}) + \epsilon\big) - \hat{R}_i^{(k)}\big)^2 \Big], \tag{44}$$

where $\hat{R}_i^{(k)}$ is the discounted reward-to-go target [40]. When recurrent actor and critic networks are used, the loss functions additionally sum over time and the networks are trained using backpropagation through time [40]. The pseudocode for recurrent MAPPO is presented in Algorithm 1.

Algorithm 1 Recurrent-MAPPO [40]

Initialize θ, parameters for π and ϕ, and parameters for critic V using orthogonal initialization [42]; set learning rate α
while step $\leq$ *step_max* **do**
  set data buffer $\mathcal{D} \leftarrow \{\}$
  for $i = 1$ **to** *batch_size* **do**
    $\tau \leftarrow [\,]$ ▷ empty list
    initialize actor RNN states $h_{0,\pi}^{(1)}, \ldots, h_{0,\pi}^{(n)}$
    initialize critic RNN states $h_{0,V}^{(1)}, \ldots, h_{0,V}^{(n)}$
    for $t = 1$ **to** T **do**
      for all agents a **do**
        $p_t^{(a)},\ h_{t,\pi}^{(a)} \leftarrow \pi\big(o_t^{(a)},\ h_{t-1,\pi}^{(a)};\ \theta\big)$
        $u_t^{(a)} \sim p_t^{(a)}$
        $v_t^{(a)},\ h_{t,V}^{(a)} \leftarrow V\big(s_t^{(a)},\ h_{t-1,V}^{(a)};\ \phi\big)$
      end for
      Execute joint action $\mathbf{u_t}$, observe $r_t,\ s_{t+1},\ \mathbf{o_{t+1}}$
      $\tau \leftarrow \tau \cup [\, s_t,\ \mathbf{o_t},\ \mathbf{h_{t,\pi}},\ \mathbf{h_{t,V}},\ \mathbf{u_t},\ r_t,\ s_{t+1},\ \mathbf{o_{t+1}} \,]$
    end for
    Compute advantage $\hat{Adv}$ on τ via GAE; use PopArt [43]
    Compute reward-to-go $\hat{R}$ on τ; normalize with PopArt
    Split τ into chunks of length L
    for $l = 0, 1, \ldots, \lfloor T/L \rfloor$ **do**
      $\mathcal{D} \leftarrow \mathcal{D} \cup \big(\tau[l : l+L],\ \hat{Adv}[\, l : l+L \,],\ \hat{R}[\, l : l+L \,]\big)$
    end for
  end for
  for *mini-batch* $k = 1, \ldots, K$ **do**
    $b \leftarrow$ random mini-batch from $\mathcal{D}$ with all-agent data
    for each data chunk c in mini-batch b **do**
      Update RNN hidden states for π and V from first hidden state in c
    end for
    Adam [44] update θ on $L(\theta)$ with data b
    Adam update ϕ on $L(\phi)$ with data b
  end for
end while

3 Swarming and Naval Warfare

This section surveys the fundamental concepts of swarm intelligence, engineered swarms, hunting algorithms, and the domain context of naval USV operations. Lastly, it surveys the latest research on utilizing MARL in autonomous USVs.

3.1 Swarms and Hunting Algorithms

3.1.1 Theoretical Background

Collective motion in nature can be seen as a source of fascination among science. For instance, ant colonies, fish schools, and bee hives exhibit complex and coordinated behavior without centralized control or any leaders. Ants can find the shortest path to food, bees collectively decide on the placement of a new nest, and African termites build mounds that may reach a diameter of 30 m and a height of 6 m, all via local interactions and simple sets of rules followed by the individuals [45]. Such self-organized behavior provides flexibility and robustness: the group as a whole is able to achieve its goals even if numerous individuals fail. This inspired the concept of swarm intelligence as a scientific discipline including research fields such as swarm optimization and distributed control in collective robotics [45].

Swarm intelligence researchers have often drawn direct analogies from biology and used them to create metaheuristics, which are high-level problem solving frameworks that generalize simple heuristics [46]. These can be broadly categorized as bio-inspired, physics-inspired, geography-inspired, and human-inspired metaheuristics [47]. For instance, insect colonies exhibit mechanisms such as stigmergy, where individuals communicate directly via the environment, for example worker ants' pheromone trails. This leads to emergent problem solving methods such as shortest-path finding, which in turn inspired the popular swarm intelligence metaheuristic algorithm Ant Colony Optimization (ACO) [45] [46]. Fish schools and bird flocks similarly highlight the role of simple local motion cues: aligning with neighbors, exhibiting attraction to close neighbors at long distance and repulsion at close range, which produce synchronized group behavior [48]. Swarm intelligence algorithms are by far the largest of the metaheuristic procedures and biomimetic computation approaches in general, representing over 67 % of all bio-inspired algorithms [46].

An early mathematical model capturing swarm dynamics is the Boids simulation by Reynolds [48], which demonstrated that a realistic flock of "boids" (bird-oid objects) emerges when each agent follows three basic behavioral rules:

1. Collision Avoidance: avoid collisions with nearby flockmates.
2. Velocity Matching: attempt to match velocity with nearby flockmates.
3. Flock Centering: attempt to stay close to nearby flockmates.

When each boid follows these local rules, the boids fly as a cohesive group, avoiding collisions, forming a lifelike flock without any central coordinator [48]. Reynolds'

experiment modeled a swarm as a particle system with distributed behavioral rules and proved that complex collective motion can be an emergent property of simple agent-based rules [48].

Following Reynolds, Vicsek et al. [49] proposed a simple model of self-driven particles: each particle is driven with a constant absolute velocity and at each time step the particles assume the average direction of their neighboring particles with some random perturbation added. In spite of the model's simplicity, the model exhibits a phase transition: while the particles normally wander randomly, below a critical level of perturbation or above a critical level of neighboring particle density, the particles spontaneously align and move in one direction [49]. This finding demonstrated quantitatively how local alignment interactions may lead to global order. Further studies placed Vicsek's model in a theoretical framework. For instance, Jadbabaie et al. [50] reformulated Vicsek's model as a consensus problem on a dynamic graph and proved convergence of headings both when every neighbor graph is always connected and when connectivity only holds jointly over bounded time intervals. This provided a rigorous link between swarm behavior and graph theory, proving that under weak connectivity conditions, distributed local rules can make a flock reach consensus, i.e. agree on a common heading [50].

Building on the Vicsek model, Cucker and Smale [51] introduced a flocking model in terms of differential equations. In contrast to the Vicsek model, every pair interacts, but with a weight that decays polynomially with distance. Other differences are that the velocities evolve dynamically, the coupling is smooth, and the flocking guarantees rely solely on the conditions on the initial state. The Cucker-Smale model proves that if the interaction decays no faster than with rate parameter $\beta \leq \frac{1}{2}$, from any initial positions and velocities, all velocities converge to a common limit and inter-agent distances remain bounded [51]. For faster decay $\beta > \frac{1}{2}$, the same convergence outcome holds if the initial positions and velocities satisfy an explicit inequality [51]. The Cucker-Smale model was later revisited by Zhang et al. [52] who designed a nonlinear controller for the Cucker-Smale flocking system that guarantees fixed-time flocking: each agents' velocities become equal by a computable time bound that doesn't depend on initial conditions.

Another analytical approach involves artificial potential fields (APF) and physics-inspired forces. By analogy to molecules or charged particles, agents can be given attractive and repulsive forces to maintain formation structure. For instance, Olfati-Saber [53] models agents as point-mass particles q_i, p_i that sense neighbors within range r, inducing a proximity net $G(q)$ whose nodes are agents and edges connect agents closer than r. Desired shapes, i.e. idealized crystalline packings for the flock are called α-lattices, configurations where each agent is at distance d from all its neighbors. Near-lattice shapes are quasi-α-lattices, and the model involves a deviation energy $E(q)$ which is small when a configuration is close to such a shape.

3.1.2 Engineered Swarms

Early swarm robotics research in the early 2000's achieved only modest group sizes – for example the Swarm-bots project financed by the European Commission between

2001 and 2005 built swarms of up to 20 robots that were capable of self-assembly, i.e. connecting physically to each other to form cooperating structures [54]. Extending the algorithms and ideas developed in Swarm-bots, the Swarmanoid project between 2006 and 2010 built heterogenous (flying, climbing, and ground-based) robot swarms that were able to collaboratively carry out a search and retrieval task [54]. In parallel with the breakthroughs in robotic swarm collaboration, research on hardware miniaturization laid grounds for deployment of swarms composing of hundreds or thousands of robots. For instance, Kilobots, low-cost open source robots developed at the Harvard University, were capable of performing collective behaviors as a group of 100 robots such as ant-inspired foraging, formation control, phototaxis, and synchronization without central control [55]. In 2014, the research group behind Kilobots showcased a swarm of 1024 Kilobots autonomously forming prescribed formations [56].

Engineered swarms exist beyond ideal ground platforms, but moving from laboratory or simulator circumstances to real-world conditions poses challenges. An early real-world experiment in 2016 [57] involved a swarm of ten low-cost differential-drive surface boats equipped with GPS, compass, and ad-hoc Wi-Fi – capable of autonomous homing, dispersion, clustering, and area monitoring. The controllers of the swarm were evolved with NEAT genetic algorithm with added environment and sensor noise to improve the transfer from the training platform simulator to real-world conditions [57]. A 2015 experiment introduced a fully decentralized task-allocation algorithm for autonomous underwater vehicles (AUVs), allowing a swarm to self-partition into work groups whose sizes adjust to changes in task demand and team size, while minimizing task switching [58]. A research paper published in 2018 [59] set out to tackle the challenges that arise when taking flying drone swarms from simulations to real-world conditions, where bounded spaces, obstacles, noise, delays, and acceleration limits exist. The model introduced in the paper is based on classic flocking theories by for instance Reynolds [48], Vicsek [49], and Olfati-Saber [53], but made alignment distance-dependent via an acceleration-limited breaking curve. Allowing distant neighbors a bigger mismatch in velocity than close neighbors respects each drone's acceleration bounds and eliminates oscillations that were obstructing real flights [59]. Another key novelty the paper introduces is treating the controller as a parameterized instance and using covariance matrix adaptation evolution strategy (CMA-ES) to maximize a single fitness, essentially turning the finding of correct parameters in the real world into an optimization problem [59].

Recent developments in computation, sensing, and communication have brought UAVs into everyday life, being capable of performing missions with extraordinary versatility [60]. Zhou et al. [60] were able to demonstrate a ten-drone swarm autonomously navigating a dense bamboo forest, maintaining formation while dodging unknown obstacles, performing random individual tasks simultaneously while maintaining minimum separation. The behavior was achieved using a novel spatio-temporal trajectory planner, which optimizes path shape and timing in real time in order to allow drones to take turns through gaps without a central coordinator. Zhang et al. [61] proposed a single end-to-end neural controller with differentiable physics, yielding an efficient, vision-only policy that scales from one drone to a multi-agent setting without redesign. The controller enabled the drones to achieve up to 20 m/s flight speeds with

static and dynamic obstacles, and exhibit mutual collision avoidance and turn-taking within a six-drone swarm, all emerging from the same policy [61].

3.2 Hunting Algorithms

Hunting algorithms are essentially algorithms for identifying, locating, and capturing a target. The design of efficient search patterns is vital to quickly locate an unknown moving target while covering a large area with a limited amount of agents. Modern research utilizes a variety of frameworks for solving the problem of minimizing the search time. The classic framework lies on the grounds of search theory and Bayesian inference, which are formalized for example by Stone [62], who proposes a generalized Lagrangian-multiplier method capable of producing uniformly optimal search plans. More novel approaches to the minimizing problem include bio-inspired metaheuristics, surveyed for example in [63] and reinforcement learning based methods, proposed for example in [64].

Each framework has produced various, sometimes overlapping strategies for swarms of agents to maximize coverage and minimize search time. One class of methods uses systematic sweeping patterns: for instance, Francos and Bruckstein [65] propose a pincer movement search where pairs of agents sweep towards each other from opposite sides of the circular search area. The coordinated pincer/double envelopment movement ensures any search target hiding in between is eventually detected, and can guarantee detection of even smart evaders under velocity constraints [65]. Asymptotic analysis of number of agents, velocities and sweeping processes, such as circular and spiral, can be used to optimize the search times [65].

The study of cooperative target search missions with autonomous vehicles has produced a numerous trajectory planning methods and search algorithms. Within this field of study, targets are typically categorized as either static or dynamic targets [66]. For static targets, there are methods surveyed in [66] such as zigzag formations for regional scanning, parallel interval search utilizing Dubins curve [67], helix method, and spanning tree algorithms. However, the major limitations of these methods are that they do not take into account potential failures the agents may experience nor any unforeseen circumstances, and they do not consider performance constraints of the agents, which can further reduce the efficiency in a real-world setting [66].

According to [66], the focus of studies has recently shifted towards moving targets and changing environments, where researchers have proposed bio-inspired heuristics such as ACO in [68], genetic algorithms, and bird flocks, as well as methods from within the Stone-style probabilistic search theory framework such as [69] and [70]. However, probability-based methods do not address the false alarm probability of sensors and ACO-based approaches are challenging to implement in real world and are limited to smaller scale search areas [66]. To address the aforementioned issues, Yang et al. [66] propose DAPSO: distributed, adaptive, real-time planning method based on local particle swarm optimization (PSO), which combines Stone's probabilistic approach with bio-inspired metaheuristics. It adopts the classical optimal-search architecture with a probability map with Bayesian updating, which is passed as an objective for the PSO to optimizing the flocking trajectories of the agents, resulting

in higher area coverage and detection rates than baseline strategies, such as random search, standard PSO, and ACO [66]. Authors of [71] propose a pursuit-evasion framework for USVs which is based on APF-inspired threat pontential fields and PSO: by encoding threat, geometry, and formation into a single potential minimized via PSO, it avoids high-dimensional HJI-style game formulations while still producing realistic, cooperative pursuit behaviors that are feasible to run in real time and in simulation.

3.3 Naval Warfare and Autonomous USVs

3.3.1 Naval Warfare

Naval warfare may be defined as the conduct of armed conflict in and from the maritime domain on, under, and above the sea, by forces employing sea power to coerce adversaries and protect interests through the control, denial, and exploitation of maritime space and communications. Corbett [72] defined it as: "[c]ommand of the sea, therefore, means nothing but the control of maritime communications ... The object of naval warfare is the control of communications". Modern doctrine reframes this as sea control, specifically “the condition in which one has freedom of action to use the sea for one’s own purposes ... [and] to deny or limit its use to the enemy” [73]. Mahan’s [74] historical analysis places this objective as crucial in the grand strategy, and emphasizes that control (command) of the sea determines the ability to project power, protect trade, and limit an adversary’s options. Contemporary research combines these notions: sea control grants the ability to use the maritime commons while denying the use of from the opponent, enabling power projection from the sea, the protection of sea lines of communication, blockade, and support to joint operations ashore [75].

3.3.2 Autonomous USVs

Marine drones, meaning unmanned surface vehicles and unmanned underwater vehicles are a relatively mature invention: the first marine drones were developed already in the 1960’s and were used for data collection and transfer [76]. Tordpedes were the first true drones, developed in the 1970’s for testing purposes and laying the ground for most of the theoretical and experimental work on marine drones [76]. The 1980’s is considered as the golden age in the development of marine drones as most of the theoretical developments were implemented in practical areas with DARPA laboratory in the US developing most of the prototypes [76]. The 1990’s saw the emergence of the first generation of marine drones capable of performing tasks for the purpose of experiments [76]. Finally, in the first decade of the 21st century, the maritime drone research and technology shifted from the academia to industrial level, and the theoretical development of cooperative motion of multiple drones had been accomplished by the end of the decade [76].

Today’s unmanned maritime systems in modern warfare are routine tools across intelligence, surveillance, reconnaissance, and logistics tasks [76]. USVs’ agility

and resilience in harsh environments make them potent instruments for strategic and precision strikes and a core element of anti-access/area-denial (A2/AD) strategies [6]. Notable use cases in recent years have been in the Black Sea during the Ukraine war where Ukraine has imposed damages on Russia's fleet through long-range coordinated strikes. A major turning point was the Sevastopol raid on October 29, 2022 where eight UAVs and seven USVs traversed approximately 300 km undetected, penetrating to a defended port and attacking the Russian naval base [76], [77]. Following months saw reduced Russian surface activity and rapid fortification of naval bases, providing evidence of the operational shock such systems can be able to deliver [76], [77]. From a broader perspective, Boretti [6] frames these developments as a paradigmatic case of asymmetrical naval warfare: remote controlled, low-cost USVs carrying various payloads can pose a significant threat to traditional high-value warships, forcing navies to invest in layered counter-USV defenses, such as electronic warfare (EW), sensors, and close-in defenses.

The most basic and inexpensive USVs with offensive capabilities of today are so-called suicide surface attack drones, commonly equipped with remote control and explosive payloads [6]. More advanced USVs may carry EW systems for deceiving and disrupting enemy radar systems and communications as well as anti-ship or land-attack missiles [6]. The most prevalent method of control for USVs is remote operation, allowing operators to guide the drones at a distance minimizing risks to human personnel, but more expensive and complex USVs are being developed to become autonomous [6]. USVs used for surveillance and reconnaissance purposes are typically equipped with cameras and advanced sensors, including radar, LiDAR (light detection and ranging), sonar, and electro-optical/infrared (EO/IR) systems to provide situational awareness using real-time data and intelligence [6]. Sonar is used for subsurface detection and mapping while radar and LiDAR are essential for avoiding obstacles and surface navigation and EO/IR sensors provide high-resolution imagery for targeting and surveillance [6]. Multi-sensor fusion can be utilized to further enhance the USVs target detection and tracking [78]. USVs use a variety of propulsion systems depending on the operating circumstances, size of the vehicle, and its function [6]. Especially smaller USVs use electric propulsion known for its quietness and efficiency, while larger USVs may use a hybrid propulsion system, where a diesel engine generates electrical power to drive electric motors, providing more range than a fully electrical system [6]. USVs designed for high-speed operations that require high maneuverability often utilize a gas turbine or a jet propulsion system, sometimes in a hybrid setting, where an electrical engine system provides the propulsion for stealthy operations and another system, such as jet turbine is used for high-velocity transits [6]. Communication and control of the USVs currently used in Ukraine is mostly done via satellite connection or mesh radio with an aerial repeater [79], [80], but commercial USV manufacturers offer also other means of connectivity, such as cellular, radio, UHF, and VHF [81].

Ukrainians have developed a range of USVs in the recent years in a variety of form-factors and capabilities [76] [80]. Among the most notable models are the Sea Baby, a water jet driven 6 m long and 2 m wide vehicle capable of carrying a combat load of 850 kg [80] and the MAGURA V5 in a slightly smaller form-factor with a

length of 5.5 m and width of 1.5 m, carrying a payload of up to 320 kg [79]. The former notably carried a successful mission damaging the Crimean bridge in July 2023, while the latter is considered highly effective against enemy vessels, with a track record of damaging and sinking numerous large enemy vessels [80].

Cutting-edge developments in AI, communication systems, machine learning, and sensor technologies have a significant role in the development of USVs, enabling improved target identification, better navigation, and ultimately, autonomous decision-making [6]. For instance, reinforcement learning can be used for collision avoidance and path planning optimization, while neural networks can be used for processing sensor data for classification and target recognition [6].

3.3.3 Countermeasures Against USVs

Present-day countermeasures against USVs are prominently evident in the Black Sea's theater of operations, where the Russian Federation has employed a variety of counter-USV strategies, ranging from traditional methods to modern-day approaches. Aviation has been a primary countermeasure: Russian aircraft and helicopters seek to detect and neutralize USVs early in their approach, and Russian sources have released footage of first-person view UAVs striking them [80]. Aviation offers rapid response and substantial firepower, lowering risk to the fleet [80]. Another traditional, though accuracy- and coordination-intensive method, is the use of machine guns and automatic weapons by the crews of Russian warships to eliminate enemy USVs at close range [80]. A more novel approach seen on the Black Sea to counter USVs has been EW systems, which are used for signal jamming of the USVs' control signals, ultimately resulting in making the USVs uncontrollable or cutting the contact between the vehicle and the control center [80]. However, it has been argued [80] that Ukrainian defense forces might have found a solution to signal jamming, since successful strikes by USVs have increased despite the use of EW countermeasures. Another novel countermeasures being developed and used are special physical obstacles in the water, such as nets and barriers for protecting ports, military bases, and anchored ships from USV strikes [80].

3.4 MARL for Autonomous Drones

A variety of work has been published in the recent years applying MARL to USV, UAV, and AUV swarms for coordination and task execution. UAVs and AUVs can be modeled in a two- or three-dimensional environment, and perform similar tasks and face similar challenges as USV swarms such as partial observability, noise, scaling, and non-stationarity, hence in addition to USV swarm applications of MARL, we survey some studies applying MARL to UAV and AUV swarms.

The authors of [82] address cooperative search for a moving target with multiple simulated UAVs operating under partial observability, obstacles, noisy sensors, and inter-UAV collision constraints, which can make standard MARL struggle with few rewards and slow convergence. To address these issues, they propose AS-MAPPO, which builds on conventional MAPPO: the algorithm forms a new global state by

aggregating the local observation information from all agents, but to reduce the dimensionality of the input data to the critic, they employ attention mechanisms that prioritize critical data and filter out irrelevant observations [82]. In training, AS-MAPPO yielded faster convergence, higher rewards, fewer collisions and higher search efficiency than conventional MAPPO and multi-agent deep deterministic policy gradient (MADDPG) across swarms sizes of 4 to 8 UAVs [82].

Another article [83] studies multi-AUV cooperative area search in an unknown environment with environmental disturbances. The authors model the search task as a Dec-POMDP, simplify the search environment into a 2D plane, and model a probabilistic sonar detection model [83]. A search information map is maintained and shared among AUVs, consisting of a coverage map, uncertainty map, and target presence probability map updated via a Bayesian rule over the sonar field of view [83]. The reward function has four components, coverage, uncertainty, target discovery, and cooperation, with varying weights [83]. The policies are trained using the SAC-QMIX algorithm, based on QMIX algorithm with inspiration from the a soft actor-critic algorithm proposed in [84], incorporating a maximum entropy mechanism into the agent networks during training, encouraging exploration and introducing randomness to the policies [83]. Both the baseline QMIX algorithm and SAC-QMIX achieve high area coverage and locate the target in both scenarios [83]. SAC-QMIX converges faster, but the authors report the same final absolute performance as QMIX [83].

Collaborative multi-UAV target tracking with capture and collision-avoidance constraints is studied in [85], which models the task as a Dec-POMDP in a 2D environment with local observations, a discrete nine-direction acceleration action set, and a composite reward consisting of a distance-change tracking term, collision penalty, step penalty, and a sparse global success reward for collaborative encirclement. The authors pre-train the policy network with expert trajectories from artificial potential field, employ behavior cloning (common method in imitation learning) with temporal difference (BCTD) to synchronize the optimization of both the policy and critic networks, and utilize MAPPO to dynamically fine-tune the policy [85]. In a three-UAV setting, the proposed MAPPO+BCTD framework outperforms MAPPO, QMIX, and MADDPG with shortest episode length, highest average reward and fastest convergence [85]. A similar task with USVs is studied in [8] where a swarm of 6-24 USVs are tasked to hunt 3-12 targets by encirclement. The authors of [8] model the task as a Dec-POMDP and propose a MAPPO-based distributed, partially observable multi-target hunting PPO (DPOMH-PPO) under the centralized training and centralized execution framework. The reward function consists of capture and guiding rewards, and collision and step penalties [8]. The authors introduce a feature embedding block to handle variable-length observation inputs, which normalizes the inputs and passes them to either a column-wise max pooling (CMP) or column-wise average-pooling (CAP) feature compression methods that output the final embedded feature [8]. Based on the ablation experiments, the CAP layer has the largest effect to the performance of the algorithm [8]. The proposed DPOMH-PPO algorithm outperforms baseline independent-learning PPO, histogram-embedding PPO, and minimap-embedding PPO algorithms in highest average reward per step and shortest task completion time [8]. Another study [2] in a similar setting targets multi-USV navigation and capture in

complex urban-like waterways via a hybrid, human-guided MARL framework. Mission designers inject knowledge by generating navigation subgoals (waypoint-like targets) and selecting among them with an immune network heuristic [2]. The PPO algorithm then learns the execution policy that drives each USV toward its assigned subgoal while coordinating with other USVs and avoiding static and dynamic obstacles [2]. The experimental results show that guidance accelerates the emergence of coordinated capture behavior relative to purely self-supervised learners [2]. Authors of a paper focusing on the encirclement of an evader by a swarm of USVs [86] propose the use MADDPG algorithm under the CTDE framework with multi-agent policy-sharing, meaning the network parameters of the best-performing agents are periodically used to replace the parameters of weaker-performing agents when the performance gap is large enough.

4 Reinforcement Learning Environment

This section describes the simulation environment in which the MAPPO algorithm is applied for the purpose of training a swarm of autonomous USVs to locate and eliminate an adversarial warship by successfully hitting the target with three USVs equipped with explosive payloads. The simulation environment is based on earlier work by Vasankari et al. [11] where the authors utilize a simulated littoral warfare environment to assess and compare the effectiveness of anti-ship missiles and kamikaze USVs against a target ship capable of countermeasures with a main gun and two CIWS. Code for the refactored version of the simulator with added features used here can be accessed at https://github.com/leskine/usv. Some parts of the code has been generated with ChatGPT 5.2 and 5.3.

4.1 Simulation Environment

The simulator is formulated as a time stepped, continuous space multi-agent environment in which a swarm of USVs attempt to locate and destroy a single surface combatant (later referred to as the target) capable of defensive measures. The environment progresses in discrete simulation steps, but positions and headings are represented in a continuous two-dimensional plane. All dynamics (radar detection and weapon effects) are deterministic given the random draws for radar detection, meaning from the MARL perspective the environment is a partially observable, stochastic transition system driven mainly by detection uncertainty, and possible communication and Global Navigation Satellite System (GNSS) jamming.

4.1.1 Environment and Episode Structure

At the beginning of an episode, a single naval target vessel is spawned at a random point within a rectangular theater of 300 km by 300 km. A swarm of N USVs are initialized at a launch site at the edge of the theater in parallel with a 100 m spacing between each USV. Each USV is assigned a unique index but is otherwise identical in its physical parameters. All entities (target, USVs, and projectiles) share a common Euclidean coordinate system and simple point-mass kinematics. The simulation proceeds in 16.67 ms increments. At each step, the agents' control actions determine the movement of the USVs, and the environment then advances all entities according to their motion models, performs radar detection from the target, updates weapon engagements, and resolves hits and destruction. The episode terminates when one of three conditions is met:

(i) the target has been destroyed by accumulating a predetermined number of USV impacts, which is interpreted as mission kill,

(ii) all USVs have been destroyed or have become inactive, or

(iii) a maximum step limit is reached, which is treated as a timeout.

4.1.2 USV Agents

The USVs are originally modeled after the Magura V5 USV in [11] with simple kinematics and a blade-turn-based steering model. The USVs have a maximum range of 800 km, speed of 42 knots (≈ 21.61 m/s), a height of 0.6 m, and inter-USV communication, based on the actual Magura V5 specifications in [79]. Each USV can be hit multiple times by defensive fire. Ammunition that passes within a small (4 m for gun and 8 m for CIWS) radius of the USV increments its internal hit counter, and when the number of hits reaches 3, the USV is marked inactive and removed from further interaction. Alternatively, a USV is marked inactive when it successfully strikes the target or reaches its maximum distance traveled. The heading of a USV is controlled by a rudder command defined as a continuous value $[-1, 1]$. The rudder controls the angle of the blades which cause the vessel to turn at a turn rate capped at 5°/s. Each USV is equipped with an idealized onboard sensor that provides the true distance and relative bearing to the target whenever the target is within the radar horizon given by

$$d = \sqrt{2kR_E}\left(\sqrt{h_{\text{USV}}} + \sqrt{h_{\text{target}}}\right) \approx 16\,\text{km}, \tag{45}$$

where d is the radar horizon distance, R_E is the radius of Earth, $k = 4/3$ is the effective Earth radius factor, and h_{USV} and h_{target} are the heights of the USV and the target. Outside this horizon, the USV receives no information about the target from its own sensing capabilities.

The per-agent observation vector is $\mathbf{o}_i = [\mathbf{o}_i^{\text{local}}; \mathbf{m}_i]$ with dimensionality 22. The agents are able to observe their own location, heading, and difference between their desired heading and actual heading, distance to the swarm centroid and nearest neighbor, as well as the location and distance to the land barrier. Once the target has been detected, the agents are able to observe their distance and relative bearing to the target. Through the communications channel, the agents receive aggregated information of the swarm's distance and relative bearing to the target. The complete list of components of the observation vector is presented in Table 1.

Index	Symbol / Name	Definition
0	x_{norm}	Agent x normalized by world half-extent, clipped to $[-1, 1]$
1	y_{norm}	Agent y normalized by world half-extent, clipped to $[-1, 1]$
2	$\cos(\psi_i)$	Cosine of agent heading
3	$\sin(\psi_i)$	Sine of agent heading
4	vis_i	Target visibility flag (0/1)
5	$d_{\text{norm},i}$	Target distance normalized by radar horizon (0 if not visible)
6	$\cos(\beta_i)$	Cosine of relative bearing to target (0 if not visible)
7	$\sin(\beta_i)$	Sine of relative bearing to target (0 if not visible)
8	$d^{\text{land}}_{\text{norm},i}$	Normalized distance to closest land-barrier point, clipped to $[0, 1]$
9	$\cos(\beta^{\text{land}}_i)$	Cosine of relative bearing to closest land-barrier point
10	$\sin(\beta^{\text{land}}_i)$	Sine of relative bearing to closest land-barrier point
11	$\cos(\Delta\psi^{\text{des}}_i)$	Cosine of desired-heading error (desired minus current)
12	$\sin(\Delta\psi^{\text{des}}_i)$	Sine of desired-heading error
13	$c_{\text{centroid},i}$	Normalized distance to active-swarm centroid, clipped to $[0, 1]$
14	$c_{\text{neighbor},i}$	Normalized nearest-neighbor distance, clipped to $[0, 1]$
15	$\overline{\text{vis}}_i$	Mean visibility from in-range communication neighbors
16	$\overline{d_{\text{norm},i}}$	Mean normalized target distance from in-range communication neighbors
17	$\overline{\cos(\beta)}_i$	Mean cosine of relative bearing from in-range communication neighbors
18	$\overline{\sin(\beta)}_i$	Mean sine of relative bearing from in-range communication neighbors
19	n^{norm}_i	Normalized in-range communication-neighbor count
20	$\overline{\cos(\Delta\psi^{\text{des}})}_i$	Mean cosine desired-heading error from in-range communication neighbors
21	$\overline{\sin(\Delta\psi^{\text{des}})}_i$	Mean sine desired-heading error from in-range communication neighbors

Table 1: Per-agent observation vector. Indices 0-14 are local observations, and indices 15-21 are communication features.

USVs communicate through a radius-limited, instantaneous broadcast channel. Let communications range $R_{\text{com}} = 30000\,\text{m}$. For each alive receiver i, the alive in-range neighbors (including self) are

$$\mathcal{N}_i(t) = \{j \in I : \ a_{j,t} = 1, \ \|\mathbf{p}_{j,t} - \mathbf{p}_{i,t}\|_2 \le R_{\text{com}}\}. \tag{46}$$

Each sender j forms a 7-dimensional message:

$$m_{j,t} = \left[\text{vis}_{j,t}, \ d_{\text{norm},j,t}, \ \cos(\beta_{j,t}), \ \sin(\beta_{j,t}), \ 0, \ \cos(\Delta\psi^{\text{des}}_{j,t}), \ \sin(\Delta\psi^{\text{des}}_{j,t})\right], \tag{47}$$

where the fifth slot is a placeholder that is overwritten after aggregation. Receiver i aggregates by mean over available neighbor messages and replaces the fifth element with the normalized neighbor count:

$$\bar{m}_{i,t} = \begin{cases} \dfrac{1}{|\mathcal{N}_i(t)|} \displaystyle\sum_{j \in \mathcal{N}_i(t)} m_{j,t}, & |\mathcal{N}_i(t)| > 0, \\ \mathbf{0}, & \text{otherwise}, \end{cases} \qquad \bar{m}_{i,t}[4] = \frac{|\mathcal{N}_i(t)|}{|I|}. \tag{48}$$

Inactive USVs training contributions are masked via death masking (described in detail in Section 4.2.6) and do not send or receive messages and are excluded from

neighborhood aggregation. Communication and GNSS jamming are implemented as optional perturbations. When communication jamming is enabled, link-level packet loss is applied before aggregation, so each neighbor transmission is independently dropped with a configured probability. Jamming can be applied for USVs inside a jamming radius around the target. GNSS jamming is modeled at the observation level by corrupting location observations: if enabled, a USV inside the configured jamming radius has its absolute position components (x, y) zeroed in its local observation, while relative sensing quantities such as target visibility and bearing-related terms remain available.

4.1.3 Target Ship

The target surface combatant represents a large corvette that shares the same kinematic model as the USVs: its position is updated as a point mass using its velocity and heading, and turning is governed by a blade angle, turning-rate factor and maximum turn rate. The ship can reach a maximum velocity of 27 kn (13.9 m/s). At every simulation step, the target performs radar surveillance against all incoming objects. For each object, the simulator computes the Euclidean distance between target and object, radar horizon distance d using equation (45), and the received radar power given a transmitter power (400 W), antenna gain (30 dB) and the object's radar cross section (RCS). If the object lies within the horizon and the received power exceeds a fixed model threshold of 10^{-6} (in simulator received-power units), it is considered detectable. The detection of an object is stochastic: the simulator draws from a Poisson distribution with rate parameter $\lambda = 1$ for targets closer than 8 km and $\lambda = 8000/d$ for more distant targets. A positive draw (value > 0) results in the object being added to the set of incoming targets, which may be engaged in the same step.

The target vessel's evasive behavior is simulated with a simple APF controller loosely based on the threat potential field idea of Wang et al. [71] and the APF formulation by Khatib [87]. Each detected USV at position p_i generates a repulsive potential that depends only on its distance $r_i = \|p_T - p_i\|_2$ to the target ship p_T. The negative gradients of these potentials are summed to form a net threat field F, which points away from nearby USVs and increases as the USVs move closer. If $\|F\|$ is below a small threshold, the target ship is considered unthreatened and stays still. Otherwise, the target starts heading towards F by controlling its rudder and throttle in proportion to the magnitude of F. In addition, the closest point of the environment's land barrier to the target generates a similar repulsive potential when distance from the target is < 20 km.

The target ship is capable of hard-kill measures with one main gun and two close-in weapon systems (CIWS). In the code these are all treated as effectors with the same interface but different parameters. Each effector maintains their ammunition count, availability flag, and a reassignment timer. When at least one incoming object (USV) has been detected, the target's engagement logic computes a predicted interception point for each object, based on their relative motion and a default projectile flight speed of 1000 m/s. For each object and effector, the simulator checks whether engagement is feasible: there must be ammunition remaining, the effector must be available, the

distance to the interception point must fall within the effector's engagement range, and the reassignment timer must be zero. If an effector is allowed to engage, a salvo aimed at the projected interception point is fired. The fired projectile's 3D distance to its target is monitored and if it falls within a hit radius of 8 m for main gun shells or 4 m for CIWS projectiles, the shot is counted as a hit. Every CIWS hit contributes one damage unit to a USV and for the main gun, damage depends on the final miss distance: a hit radius r_{hit} of $4 < r_{\text{hit}} < 8$ causes one unit, $1 < r_{\text{hit}} < 4$ causes two, and $r_{\text{hit}} < 1$ causes three. Once a USV has accumulated more than two damage units, it is considered destroyed and flagged inactive. The target ship is additionally equipped with simulated EW capabilities in GNSS signal and communication interference, which are modeled by denying the USVs' capability to observe its location coordinates and message vectors from other USVs.

For the target itself, incoming USVs act as kamikaze vessels. A USV that comes within the target's hit radius of 80 m is considered as a successful strike and the target's hits counter is incremented. Once the counter reaches a predefined number, the target is considered destroyed and the episode ends with a successful outcome for the USV swarm.

4.2 MARL Formulation

4.2.1 Dec-POMDP

From the MARL perspective, the setting is a stochastic multi-agent system with partial observability, which can be expressed as a finite, cooperative, partially observable stochastic game with n agents. All USVs are homogeneous and act in cooperation. Only the USVs are given agency: the target is either static or it's movement is controlled by its APF-inspired evasion algorithm, and its detection and engagement logic are stochastic, hence it is treated as a stochastic part of the environment dynamics. Due to the cooperative nature of the task, the agents share a common reward function, which leads to a Dec-POMDP formulation

$$\mathcal{M} = \big(I,\, \mathcal{S},\, \{\mathcal{A}_i\}_{i\in I},\, \{O_i\}_{i\in I},\, \mathcal{T},\, \mathcal{Z},\, R,\, \gamma,\, \mu\big), \tag{49}$$

where

- $I = \{1, \ldots, n\}$ is the finite set of USVs,
- $\mathcal{S}$ is the state space,
- $\mathcal{A}_i$ is the action space of agent i,
- O_i is the observation space of agent i,
- $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ with $\mathcal{A} = \mathcal{A}_1 \times \cdots \times \mathcal{A}_n$ is the state transition kernel, such that $\sum_{s' \in \mathcal{S}} \mathcal{T}(s, a, s') = 1$ for all (s, a),
- $\mathcal{Z} : \mathcal{S} \times \mathcal{A} \times O_1 \times \cdots \times O_n \to [0, 1]$ is the joint observation kernel,

- $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the common reward function (i.e., $R_1 = \cdots = R_n = R$),
- $\gamma \in [0, 1)$ is the discount factor,
- $\mu : \mathcal{S} \to [0, 1]$ is the initial state distribution.

At each time step t, the environment is in a state $s_t \in \mathcal{S}$, each USV i receives a private observation $o_{i,t} \in \mathcal{O}_i$ and selects an action $a_{i,t} \in \mathcal{A}_i$. The joint action $a_t = (a_{1,t}, \ldots, a_{n,t})$ induces a transition to $s_{t+1} \sim \mathcal{T}(\cdot \mid s_t, a_t)$, a joint observation $(o_{1,t+1}, \ldots, o_{n,t+1}) \sim \mathcal{Z}(\cdot \mid s_{t+1}, a_t)$, and a shared reward $r_t = R(s_t, a_t, s_{t+1})$. The target ship's actions are embedded in $\mathcal{T}$ as part of the environment.

4.2.2 MAPPO Algorithm

CTDE is a natural choice for this task: centralized execution is infeasible because agents must act under communication constraints and partial observability, while fully decentralized training suffers from non-stationarity since each agent perceives other independently learning agents as part of a shifting environment, leading to poor convergence [12]. CTDE resolves this by giving the critic access to the global state during training while each agent's policy conditions only on its local observation history and received messages at execution time. MAPPO is chosen as the learning algorithm due to its strong empirical performance in cooperative multi-agent tasks [39]. Our variant of MAPPO is outlined in Algorithm 2.

Algorithm 2 MAPPO with action repeat

Initialize actor π_θ (GRU) and feed-forward critic V_ϕ with orthogonal initialization
Initialize Adam optimizers with learning rates α_π, α_V
Initialize PopArt running statistics $\hat{\mu}, \hat{\sigma}^2 \leftarrow 0, 1$
Derive effective discount $\gamma_{\text{eff}} \leftarrow \gamma^F$ and $\lambda_{\text{eff}} \leftarrow \min\big(\gamma\lambda/\gamma_{\text{eff}},\ 0.995\big)$ from action repeat F
Launch E parallel environments; initialize observations $\mathbf{o}_0^{(e)}$, states $s_0^{(e)}$, GRU states $h_0^{(a,e)}$
for update $= 1, 2, \ldots, U_{\max}$ **do**
    Update learning rates, entropy coefficient c_H, and optional curriculum schedules
    $\mathcal{D} \leftarrow \{\}$ ▷ rollout buffer
    for $t = 0$ **to** $T - 1$ **do** ▷ collect T transitions across all E envs
        for all agents a in **all** envs e **(vectorized) do**
            $\mu_t^{(a,e)}, \sigma_t^{(a,e)}, h_{t+1}^{(a,e)} \leftarrow \pi_\theta\big(o_t^{(a,e)}, h_t^{(a,e)}\big)$
            $u_t^{(a,e)} \sim \mathcal{N}\big(\mu_t^{(a,e)}, \operatorname{diag}(\sigma_t^{(a,e)})^2\big)$
        end for
        $v_t^{(e)} \leftarrow V_\phi(s_t^{(e)})$ ▷ shared scalar value per env
        for all envs e **(parallel) do**
            for $f = 1$ **to** F **do** ▷ action repeat
                Apply $\mathbf{u}_t^{(e)}$ in env e; accumulate γ-discounted reward $r_t^{(e)}$
                if env e terminates **then break**; auto-reset and re-initialize $h^{(\cdot,e)}$
                end if
            end for
        end for
        Store $\big(o_t^{(e)},\ s_t^{(e)},\ \mathbf{u}_t^{(e)},\ \log \pi_t^{(e)},\ r_t^{(e)},\ d_t^{(e)},\ \mathbf{a}_t^{\text{alive},(e)},\ v_t^{(e)},\ \mathbf{h}_t^{(e)}\big)$ in $\mathcal{D}$
    end for
    Bootstrap: $v_T^{(e)} \leftarrow V_\phi(s_T^{(e)})$
    Compute GAE advantages $\hat{A}_t$ and returns $\hat{R}_t$ using $\gamma_{\text{eff}}, \lambda_{\text{eff}}$
    Broadcast: $\hat{A}_{t,e}^{(a)} \leftarrow \hat{A}_{t,e} \cdot \mathbf{1}[a \text{ alive}]; \quad \hat{R}_{t,e}^{(a)} \leftarrow \hat{R}_{t,e}$
    Normalize $\hat{A}^{(a)}$ per agent index with alive masking
    for epoch $= 1$ **to** K **do**
        Draw random permutation of the $T \times E$ *(time, env)* indices
        for each mini-batch b of size M **do** ▷ no RNN chunking
            Re-evaluate $\log \pi_\theta(u \mid o, h)$ and entropy H on batch b (single-step GRU)
            Compute PPO surrogate: $L^\pi \leftarrow -\frac{1}{\|\mathbf{a}^{\text{alive}}\|} \sum \mathbf{a}^{\text{alive}} \cdot \min\big(r_\theta \hat{A},\ \text{clip}(r_\theta)\hat{A}\big)$
            $L^{\text{actor}} \leftarrow L^\pi - c_H \cdot \bar{H}_{\text{alive}}$
            Update PopArt: $\hat{\mu}, \hat{\sigma} \leftarrow \text{EMA}(\hat{R})$; rescale V_ϕ output layer
            $L^V \leftarrow \frac{c_V}{\|\mathbf{a}^{\text{alive}}\|} \sum \mathbf{a}^{\text{alive}} \cdot \big(\tilde{V}_\phi(s) - \tilde{R}\big)^2$ ▷ $\tilde{\cdot}$ = PopArt-normalized
            Adam step on θ with ∇L^{actor}, grad clip $\delta_{\max}$
            Adam step on ϕ with ∇L^V, grad clip $\delta_{\max}$
            if $\widehat{\text{KL}} > \text{KL}_{\text{stop}}$ **then** early-stop this epoch
            end if
        end for
    end for
end for

4.2.3 Action, Observation, and State Spaces

In the MAPPO setup, each USV $i \in I$ uses a heading-only continuous action. Throttle is fixed by the simulator, so policy actions do not control speed. The action space of agent i at time step t is:

$$\mathcal{A}_{i,t} = (c_{i,t},\ s_{i,t}) \in \mathbb{R}^2 \tag{50}$$

The pair is interpreted as a desired relative-heading vector and unit-normalized:

$$(\hat{c}_{i,t},\ \hat{s}_{i,t}) = \begin{cases} \dfrac{(c_{i,t},\ s_{i,t})}{\sqrt{c_{i,t}^2 + s_{i,t}^2}}, & \text{if } \sqrt{c_{i,t}^2 + s_{i,t}^2} \geq \varepsilon \\ (1,\ 0), & \text{otherwise} \end{cases} \tag{51}$$

where $\varepsilon = 10^{-6}$. The desired relative heading change is then extracted as $\Delta\psi^{\star}_{i,t} = \text{atan2}(\hat{s}_{i,t},\ \hat{c}_{i,t})$ and clipped to ± 60. This heading error is linearly mapped to a rudder command:

$$u_{i,t} = \text{clip}\left(\frac{\Delta\psi^{\star}_{i,t}}{\Delta\psi_{\max}},\ -1,\ 1\right) \tag{52}$$

where $\Delta\psi_{\max} = 60$. Thus, the effective control authority is heading change via rudder. Using a the heading-vector action mode over direct rudder control has many advantages: it avoids the angular discontinuity of a scalar heading, lets the policy reason about direction rather than low-level rudder authority, and naturally supports a relative framing that helps with generalization. The built-in normalization and $\pm 60^{\circ}$ clamp also results in smooth, bounded maneuvers without requiring the network to learn those constraints.

The MAPPO actor receives per-agent observations and the critic receives a centralized global state. The actor observation per USV is

$$\mathbf{o}_{i,t} = [\mathbf{o}^{\text{local}}_{i,t};\ \bar{\mathbf{m}}_{i,t}] \in \mathbb{R}^{19}, \tag{53}$$

composed of 15 local kinematic and visibility features and 7 communication features described in Table 1. The centralized critic state is:

$$s_t = \left[g_{1,t}, \ldots, g_{N,t},\ z_t\right], \tag{54}$$

where each per-USV block $g_{i,t} \in \mathbb{R}^{14}$ and the global block $z_t \in \mathbb{R}^K$ are defined in Table 2. With $N_{\text{eff}} = 3$ effectors, $K = 10 + 3N_{\text{eff}} = 19$ and the critic state dimension is $\dim(s_t) = 14N + 19$.

Per-USV block $g_{i,t} \in \mathbb{R}^{14}$ (repeated N times)		
Offset	**Symbol**	**Definition**
0	$x_{\text{norm},i}$	Agent x normalized by world half-extent, clipped to $[-1, 1]$
1	$y_{\text{norm},i}$	Agent y normalized by world half-extent, clipped to $[-1, 1]$
2	$\cos(\psi_i)$	Cosine of agent heading
3	$\sin(\psi_i)$	Sine of agent heading
4	$d_{\text{world},i}$	Target distance normalized by world half-extent
5	$d_{\text{norm},i}$	Target distance normalized by radar horizon (0 if not visible)
6	$\cos(\beta_i)$	Cosine of relative bearing to target (0 if not visible)
7	$\sin(\beta_i)$	Sine of relative bearing to target (0 if not visible)
8	$v_{\text{norm},i}$	Speed normalized by maximum speed
9	$c_{\text{centroid},i}$	Normalized distance to swarm centroid
10	$c_{\text{neighbor},i}$	Normalized nearest-neighbor distance
11	$\cos(\Delta\psi_i^{\text{des}})$	Cosine of desired-heading error
12	$\sin(\Delta\psi_i^{\text{des}})$	Sine of desired-heading error
13	a_i	Activity flag (1 if alive, 0 otherwise)
Global block $z_t \in \mathbb{R}^K$, $K = 19$		
0	x_{tgt}	Target x normalized by world half-extent
1	y_{tgt}	Target y normalized by world half-extent
2	$\cos(\psi_{\text{tgt}})$	Cosine of target heading
3	$\sin(\psi_{\text{tgt}})$	Sine of target heading
4	v_{tgt}	Target speed normalized by maximum target speed
5	n_{inc}	Incoming-target count normalized by N
— repeated for each effector $e = 1, \ldots, 3$ —		
$6+3(e-1)$	ammo_e	Normalized remaining ammunition
$7+3(e-1)$	avail_e	Availability flag $\in [0, 1]$
$8+3(e-1)$	reassign_e	Reassignment timer $\in [0, 1]$
15	n_{salvo}	Active salvo count normalized by N
16	h_{prog}	Hit count normalized by hits-to-destroy
17	f_{dest}	Target-destroyed flag $(0/1)$
18	f_{inact}	Fraction of inactive USVs

Table 2: Centralized critic state vector. The per-USV block is emitted for each of N agents. Inactive agents emit all zeros including the activity flag as described in Section 4.2.6. The global block appends target state, effector status, and episode progress indicators.

4.2.4 Action Repeat

State transitions in the simulator occur at the underlying physics timestep, but the policy is queried only every 40 simulator steps and the selected action is held constant in between. This procedure commonly referred to as frame-skipping or action repeat has been studied in prior work, for example in [88]–[90]. In our setting, frame-skipping is essential for learning performance because the simulation runs at 60 Hz and episodes can consist of over 10^5 environment steps during training, while the most consequential events (hits on the target and target destruction) typically occur late in the episode. Action repeat reduces the effective decision horizon by a factor of 40 and corresponds to

issuing a new control decision approximately every 0.67 seconds, which improves the tractability and stability of policy optimization. In addition, it improves computational efficiency by reducing the number of policy evaluations per simulated second.

Action repeat results in a mismatch between the simulator timestep at which discounting is conceptually applied and the coarser decision timestep at which the policy buffer records transitions. Hence, a per-action discount factor γ_a is introduced that represents the desired discounting between consecutive policy decisions. The per-simulator-step discount is $\gamma = \gamma_a^{1/F}$ where $F = 40$ is the action repeat factor. GAE is then computed at the decision-level with an effective discount $\gamma_{\text{eff}} = \gamma^F = \gamma_a$ and an adjusted trace-decay parameter

$$\lambda_{\text{eff}} = \min\left(\frac{\gamma\,\lambda}{\gamma_{\text{eff}}},\ 0.995\right), \tag{55}$$

where λ is the base GAE parameter. This preserves the intended trace-decay rate $\gamma\lambda$ per simulator step while operating at the coarser decision-level time resolution. The accumulated reward within each action repeat window is discounted at γ per simulator step and normalized by the geometric series sum, so the reward signal seen by the learner is consistent with the effective discount horizon.

4.2.5 Reward Function

Designing a reward function plays a critical role in learning and the resulting performance and stability of the learned model. In our setting, credit assignment is harder due to a high number of agents sharing a common reward function, learning dynamics are fragile due non-stationarity caused by a high number of simultaneously learning teammates, and the episode horizon is particularly long. The true objective of our setting is straight forward (locate and destroy a target vessel), but difficult to learn from directly since success is nearly impossible early in the training and occurs late in the episode. This makes learning from raw terminal reward sample-inefficient. Hence, the design of the reward function must balance between:

1. Correctness: optimizing the reward should correspond to solving the actual task, meaning reward hacking and local optima should be avoided.

2. Learnability: rewards must provide sufficient shaped signal from early on in the episode and frequently enough for gradient-based policy optimization to make progress.

The reward function used in this setting therefore combines extrinsic rewards that represent mission outcomes (hits, target destruction/mission success, loss penalties) with intrinsic rewards (i.e. reward shaping, discussed for example in [91]) that encourage behaviors that make mission success more likely.

The characteristics of the mission influence design of the reward function. Target destruction can occur far into an episode, which makes the terminal reward heavily discounted and difficult to credit back to the early actions that enabled success, especially during early training when the policy often fails to find or reach the target at

all. The setting amplifies multi-agent credit assignment issues because the environment returns one shared scalar reward per step that is broadcast to all agents, while the learner treats each agent's transition as a separate sample. This is a common pattern in CTDE, but it is not scale-invariant unless shared rewards are normalized. Additionally, the problem is partially observable and naturally two-phase: before contact, the policy must search to detect the target after which it must transition to interception behaviors such as closing distance, aligning heading, and generating hits until destruction. This means rewards that are helpful for exploration in the search phase can become harmful after detection. Hence, the reward function must fade certain intrinsic terms once detection occurs. Finally, the environment contains irreversible failure events such as USVs being destroyed by the target vessel's defenses or being lost by hitting land. Penalizing losses is desirable, but doing so may incentivize the policy to avoid engagement and stuck to a local maximum. At each environment step t, a shared reward is computed as

$$r_t = r_t^{\text{intrinsic}} + r_t^{\text{extrinsic}} \tag{56}$$

where $r_t^{\text{intrinsic}}$ aggregates the shaping rewards and penalties and $r_t^{\text{extrinsic}}$ aggregates hit rewards, loss penalties, and a mission success reward.

Intrinsic terms

The intrinsic term is computed at each environment step t as

$$r_t^{\text{intrinsic}} = r_t^{\text{coverage}} + r_t^{\text{detect}} + r_t^{\text{prog}} + r_t^{\text{align}} + r_t^{\text{time}} \tag{57}$$

where r_t^{coverage} is a reward for covering the search area, r_t^{detect} is a one-time reward for detecting the target vessel, r_t^{prog} is a progress reward for closing in on the target, r_t^{align} is a reward for aligning bearing towards the target, and r_t^{time} is a time step penalty. The coverage r_t^{coverage} and detection rewards r_t^{detect} are shared reward terms that are divided by number of active agents, while progress r^{prog}, align r^{align}, and time r_t^{time} are computed and summed over each active agent and then averaged over the active agents. This normalization makes the magnitude of the shaping signal approximately invariant to the number of alive agents, preventing the shaping reward from changing simply because agents have been lost, improving training stability across episodes.

The coverage reward term r_t^{coverage} is used to encourage the agents to explore the area in order to locate the target. To quantize area exploration, the world is divided into 1500 m by 1500 m cells. Each cell has its own age counter $a_t^{i,j}$: at episode reset, all ages start at 0, and the age of all cells increments by 1 every step. When a cell is visited in a step, its age is reset to 0. At each time step t, a novelty n sum is computed over unique visited cells:

$$\sum n_t = \sum \min\left(1, \frac{a_t^{i,j}}{T_{\text{reset}}}\right), \tag{58}$$

where T_{reset} is age counter reset interval. The coverage reward is

$$r_t^{\text{coverage}} = \frac{\lambda_{\text{coverage}} w_t^{\text{search}} \sum n_t}{N_t}, \tag{59}$$

where N_t is the number of alive agents, $\lambda_{\text{coverage}}$ is the coverage reward weight and w_{search} is a smooth reward gating

$$w_t^{\text{search}} = \begin{cases} 1 & \text{search mode} \\ \frac{t_{\text{fade}}}{T_{\text{fade}}} & \text{fade-out} \\ 0 & \text{post-fade} \end{cases} \tag{60}$$

which shifts from search mode to fade-out when the detection condition `has_info` = 1, and from fade-out to post-fade after the fade-out period. When the detection condition is fulfilled, a one-time detection reward is given:

$$r_t^{\text{detect}} = \frac{R_{\text{detect}}}{N_t}. \tag{61}$$

The purpose of the detection reward is to avoid a local maximum of searching a maximal area without detecting the target. Additionally, a fixed time step penalty r^{time} is applied to each active agent on every non-terminal step to encourage faster mission execution:

$$r_t^{\text{time}} = -\frac{\lambda_{\text{time}}}{N_t}. \tag{62}$$

Potential-based reward shaping (PBRS) [91] is used to design the progress and alignment rewards r_t^{prog} and r_t^{align}. PBRS computes shaping as a discounted difference of potentials:

$$r_t^{\text{PBRS}} = \lambda(\gamma\Phi(s_{t+1}) - \Phi(s_t)), \tag{63}$$

where λ is the reward weight, γ is the discount factor, Φ is the potential function, and s_t is the state at time step t. The terms become a telescoping series when added up, converging to a finite value due to $\lambda < 1$. The terms cancel out over time, with the exception of the first term. The full shaped return becomes $\sum_{t=0}^{\infty} \gamma^t r_t - \lambda\Phi(s_0)$, which is independent of the action. Hence, the best action in state s stays the same, meaning the shaping doesn't change the optimal policy, even though the reward signal is richer.

The progress and alignment rewards are gated by

$$g_t = \mathbb{I}\{\texttt{has\_info}(s_t)\}, \qquad g_t \in \{0, 1\}, \tag{64}$$

where `has_info` = 1 when 10% of the alive agents have maintained visibility to the target for 5 consecutive environment steps, and 0 otherwise. The gating prevents information leakage from the reward function when the target is not visible, and the 10% share and 5 consecutive environment step requirements enhance stability and swarm behavior. The progress reward r_t^{prog} is formulated as

$$r_t^{\text{prog}} = \lambda_{\text{prog}}\, g_t(\gamma\Phi_{\text{prog}}(s_{t+1}) - \Phi_{\text{prog}}(s_t)) \tag{65}$$

with potential

$$\Phi_{\text{prog}}(s_t) = \begin{cases} 0, & \text{if } \texttt{terminal}(s_t) = 1, \\ \dfrac{b^{\left(1-\frac{d_t}{d_0}\right)} - 1}{b - 1}, & \text{otherwise,} \end{cases} \tag{66}$$

where d_t is the distance from the USV to the target at time t, d_0 is the initial distance at the start of the episode, and b is the exponential base. d_t/d_0 is clamped to $[0, 1]$. The $\texttt{terminal}(s_t)$ condition is set to 1 at the terminal event which is the time step when the target is destroyed by the last required hit. The alignment reward is formulated as

$$r_t^{\text{align}} = \lambda_{\text{align}}\, g_t(\gamma\Phi_{\text{align}}(s_{t+1}) - \Phi_{\text{align}}(s_t)), \tag{67}$$

with potential

$$\Phi_{\text{align}}(s_t) = \begin{cases} 0, & \text{if } \texttt{terminal}(s_t) = 1, \\ \left(1 - \text{clip}\left(\dfrac{d_t}{d_0},\, 0,\, 1\right)\right) \cdot \dfrac{\cos(\theta_t) + 1}{2}, & \text{otherwise,} \end{cases} \tag{68}$$

where θ is the heading error. The align reward $\frac{\cos(\theta)+1}{2}$ is distance-weighted by $1 - \text{clip}(\frac{d}{d_0}, 0, 1)$, so it grows as the agent closes in and goes to 0 when far. The rationale behind the progress reward is to guide the agent towards the target when the agents have visibility to the target, and the alignment reward is used to enhance the agents' strike accuracy. The distance-weighting allows the agent to perform evasive maneuvers from the target's defenses when closing in.

PBRS has practical benefits here because it depends on progress via the temporal difference of a potential function whereas conventional dense distance rewards scale directly with absolute distance and can change reward magnitude and variance when initial distances (in this case the distances from the agents to the target when `has_info` is set to 1) vary. Therefore the shaping signal remains more consistent across different initial distances, which reduces sensitivity to distance variance and improves training stability.

Extrinsic terms

Let $H_t \in \mathbb{Z}_+$ denote the cumulative number of successful hits on the target up to and including time t, and define $\Delta H_t = H_t - H_{t-1} \in \{0, 1\}$ as the number of new strikes at step t. Each successful strike yields a positive reward r_t^{hit}, and destroying the target by achieving $H_t \geq \texttt{required\_hits}$ yields an additional bonus r_t^{destroy} at the terminal step. However, if agents are lost during the episode, a loss penalty r_t^{loss} proportionate to the cumulative number of destroyed agents D_t will be given on the terminal step. Every agent L lost by hitting the land barrier at time t produces an immediate penalty

r_t^{land}. Formally,

$$r_t^{\text{extrinsic}} = r_t^{\text{hit}} + r_t^{\text{destroy}} - r_t^{\text{loss}} - r_t^{\text{land}} \tag{69}$$

$$r_t^{\text{hit}} = \frac{\lambda_{\text{hit}} \Delta H_t}{N_t} \tag{70}$$

$$r_t^{\text{destroy}} = \frac{R_{\text{destroy}}}{N_t} \tag{71}$$

$$r_t^{\text{loss}} = \frac{R_{\text{death}} \cdot D_t}{N_t}, \tag{72}$$

$$r_t^{\text{land}} = \frac{R_{\text{land}} \cdot L_t}{N_t}. \tag{73}$$

The purpose of penalizing losses caused by the target on the terminal step is more complex in terms of reward assignment compared to immediate penalties, but it prevents a local maximum where agents avoid contact with the target vessel due to potential losses from its defenses.

4.2.6 Agent Inactivity and Death Masking

Agents can become inactive after successful target strikes, by being destroyed by the target's defenses, by hitting the land boundary, or by range exhaustion at different times across agents within one episode. For handling agents that are lost before episode termination, we employ a strategy called death masking [40] which has major advantages: it preserves fixed-size recurrent batching, and only the experience from active agents contributes to gradients, while transitions prior to the inactivity remain fully learnable. We keep a fixed agent index set

$$I = \{1, \ldots, n\}$$

throughout the rollout and use masking rather than removing indices. Let

$$d_{i,t} \in 0, 1$$

be the per-agent done flag provided by the environment after step t, and define the alive mask

$$m_{i,t} = 1 - d_{i,t} \in 0, 1. \tag{74}$$

Once an agent becomes inactive ($m_{i,t} = 0$), it remains in the tensor structure for bookkeeping and batching, but is excluded from physical updates and from learning terms via masking.

Per-agent observations retain a fixed shape for all $i \in I$. For inactive agents, the local observation is simply replaced by a zero vector:

$$o_{i,t} = \begin{cases} \mathbf{o}_{i,t}, & m_{i,t} = 1, \\ \mathbf{0}, & m_{i,t} = 0. \end{cases} \tag{75}$$

Inactive agents retain fixed slots in the centralized critic state s_t, but their feature blocks are zeroed with an explicit alive indicator set to 0. Action tensors also keep fixed shape, but simulator dynamics apply control only to agents with $m_{i,t} = 1$.

The actor is recurrent with a hidden state tensor H_t over environments and agents. After each environment step, hidden states of inactive agents are zeroed by

$$H_{t+1} \leftarrow H_{t+1} \odot M_t, \tag{76}$$

where M_t is the broadcasted alive mask. For environments that terminate, hidden states are reset to zero for that environment. This prevents inactive-agent memory from leaking into future decisions.

GAE is computed at environment level using shared reward and centralized values, producing scalar A_t and G_t per environment time step. Agent-level tensors are formed by broadcast:

$$A_{i,t} = m_{i,t} A_t, \qquad G_{i,t} = G_t.$$

Hence inactive agents have no contribution to the advantage, but returns remain shared and are filtered by masks inside loss terms.

PPO losses also utilize masking. Let $\rho_{i,t}$ be the PPO ratio and ϵ the clip parameter. The actor objective is

$$L_{\text{actor}}(\theta) = -\frac{1}{\sum_{i,t} m_{i,t}} \sum_t \sum_{i \in I} m_{i,t}, \min\Big(r_{i,t} A_t, , \text{clip}(\rho_{i,t}, 1-\epsilon, 1+\epsilon) A_t\Big), \tag{77}$$

equivalently using $A_{i,t}$ inside the minimum. Entropy regularization is masked the same way:

$$L_{\text{ent}} = -\frac{1}{\sum_{i,t} m_{i,t}} \sum_{t,i} m_{i,t}, \mathcal{H}_{i,t}. \tag{78}$$

With centralized scalar critic $V_\phi(x_t)$, the critic loss is computed per agent by expansion and masking:

$$L_{\text{critic}}(\phi) = \frac{1}{\sum_{i,t} m_{i,t}} \sum_t \sum_{i \in I} m_{i,t} \big(V_\phi(x_t) - G_t\big)^2. \tag{79}$$

4.2.7 Neural Network Architecture

The MAPPO implementation employs two neural networks: a recurrent actor for decentralized action selection and a feed-forward critic for centralized value estimation. Both networks use orthogonal weight initialization and zero bias initialization across all linear layers. All N agents share a single actor instance, i.e. employ parameter sharing, and the critic is shared as it estimates a single environment-level value.

Actor Network

The actor π_θ is a recurrent Gaussian policy that maps each agent's observation $\mathbf{o}_{i,t} \in \mathbb{R}^{22}$ and a carried hidden state $h_{i,t} \in \mathbb{R}^{128}$ to the continuous action distribution. The architecture consists of three stages:

1. **Observation encoder:** A two-layer multilayer perceptron (MLP) [92] with tanh activations embeds the observation into a fixed-size representation:

$$\mathbf{e}_{i,t} = \tanh\Big(W_2 \, \tanh\big(W_1 \, \mathbf{o}_{i,t} + b_1\big) + b_2\Big), \quad \mathbf{e}_{i,t} \in \mathbb{R}^{128}, \tag{80}$$

   where $W_1 \in \mathbb{R}^{128\times 22}$, $W_2 \in \mathbb{R}^{128\times 128}$, and $b_1, b_2 \in \mathbb{R}^{128}$.

2. **Temporal memory:** A single-layer gated recurrent unit (GRU) [93] processes the embedding and updates the agent's hidden state:

$$\mathbf{x}_{i,t}, \; h_{i,t+1} = \mathrm{GRU}\big(\mathbf{e}_{i,t}, \; h_{i,t}\big), \quad \mathbf{x}_{i,t}, \; h_{i,t+1} \in \mathbb{R}^{128}. \tag{81}$$

   The hidden state is the only per-agent persistent state and is carried across timesteps within an episode, enabling the policy to integrate partial observations over time. At episode boundaries the hidden state is reset to zero.

3. **Action head:** A linear projection maps the GRU output to the Gaussian mean:

$$\boldsymbol{\mu}_{i,t} = W_\mu \, \mathbf{x}_{i,t} + b_\mu, \quad \boldsymbol{\mu}_{i,t} \in \mathbb{R}^2, \tag{82}$$

   where $W_\mu \in \mathbb{R}^{2\times 128}$ and $b_\mu \in \mathbb{R}^2$. The log standard deviation $\log \boldsymbol{\sigma} \in \mathbb{R}^2$ is a learnable parameter vector that is independent of the state, meaning exploration noise is shared across all observations and agents. Actions are sampled from a diagonal Gaussian:

$$\mathbf{a}_{i,t} \sim \mathcal{N}\big(\boldsymbol{\mu}_{i,t}, \; \mathrm{diag}(\boldsymbol{\sigma}^2)\big), \tag{83}$$

   where $\boldsymbol{\sigma} = \exp(\log \boldsymbol{\sigma})$. The two action components are interpreted as a desired heading vector (c, s) as described in Section 4.2.3.

The recurrent architecture is essential since each agent observes only a partial, egocentric view of the environment. The GRU allows the policy to maintain beliefs about unobserved quantities, such as the target's state when outside visibility range or the intentions of distant agents by integrating information over the observation history.

Critic Network

The critic V_ϕ is a feed-forward network that maps the centralized global state $s_t \in \mathbb{R}^{D_s}$ to a scalar value estimate, where $D_s = 14N + 19$.

1. **Feature extraction:** A two-layer MLP with tanh activations maps the global state to a latent representation:

$$\mathbf{h}_t = \tanh\Big(W_4 \, \tanh\big(W_3 \, s_t + b_3\big) + b_4\Big), \quad \mathbf{h}_t \in \mathbb{R}^{256}, \tag{84}$$

   where $W_3 \in \mathbb{R}^{256\times 355}$, $W_4 \in \mathbb{R}^{256\times 256}$, and $b_3, b_4 \in \mathbb{R}^{256}$.

2. **Value head:** A linear layer produces the scalar value estimate:

$$\tilde{v}_t = W_v \, \mathbf{h}_t + b_v, \quad \tilde{v}_t \in \mathbb{R}, \tag{85}$$

where $W_v \in \mathbb{R}^{1\times 256}$ and $b_v \in \mathbb{R}$.

3. **Pop-Art normalization:** The value head output $\tilde{v}_t$ is in a normalized value space induced by Pop-Art [43]. Pop-Art tracks running estimates of the return mean $\hat{\mu}$ and variance $\hat{\sigma}^2$ via exponential moving averages with decay rate β:

$$\hat{\mu} \leftarrow (1-\beta)\,\hat{\mu} + \beta\,\bar{R}_{\text{batch}}, \tag{86}$$

$$\hat{\sigma}^2 \leftarrow (1-\beta)\,\hat{\sigma}^2 + \beta\ \text{Var}(R_{\text{batch}}) + (1-\beta)\,\beta\,(\bar{R}_{\text{batch}} - \hat{\mu}_{\text{old}})^2. \tag{87}$$

After each statistics update, the value head weights W_v and bias b_v are rescaled in place so that the denormalized output remains consistent:

$$W_v \leftarrow \frac{\hat{\sigma}_{\text{old}}}{\hat{\sigma}_{\text{new}}}\, W_v, \qquad b_v \leftarrow \frac{\hat{\sigma}_{\text{old}}}{\hat{\sigma}_{\text{new}}}\, b_v + \frac{\hat{\mu}_{\text{old}} - \hat{\mu}_{\text{new}}}{\hat{\sigma}_{\text{new}}}. \tag{88}$$

The final denormalized value is $V(s_t) = \tilde{v}_t \cdot \hat{\sigma} + \hat{\mu}$. Predictions and targets are both compared in normalized space during training, while GAE advantages are computed using denormalized values. This stabilizes learning when the return distribution shifts substantially over the course of training, which is helpful especially in this setting where the task horizon is long and terminal rewards (target hits and destruction) are sparse.

The critic was chosen not to use recurrence whereas the actor uses it. The critic observes the complete Markov state, so there is no partial observability that recurrence would need to handle. Transitions can be sampled as independent pairs and it reduces memory use since critic hidden states don't need to be stored in the rollout buffer. Table 3 summarizes the parameter counts for the actor and critic networks and Figure 2 provides the visual representation of both networks.

Network	Layer	Input	Output	Parameters
Actor	Linear + Tanh	22	128	2 944
Actor	Linear + Tanh	128	128	16 512
Actor	GRU (1 layer)	128	128	99 072
Actor	Linear (μ)	128	2	258
Actor	$\log\sigma$ param	—	2	2
			Actor total	**118 788**
Critic	Linear + Tanh	355	256	91 136
Critic	Linear + Tanh	256	256	65 792
Critic	Linear (value)	256	1	257
			Critic total	**157 185**

Table 3: Parameter counts for actor and critic networks when $N = 24$.

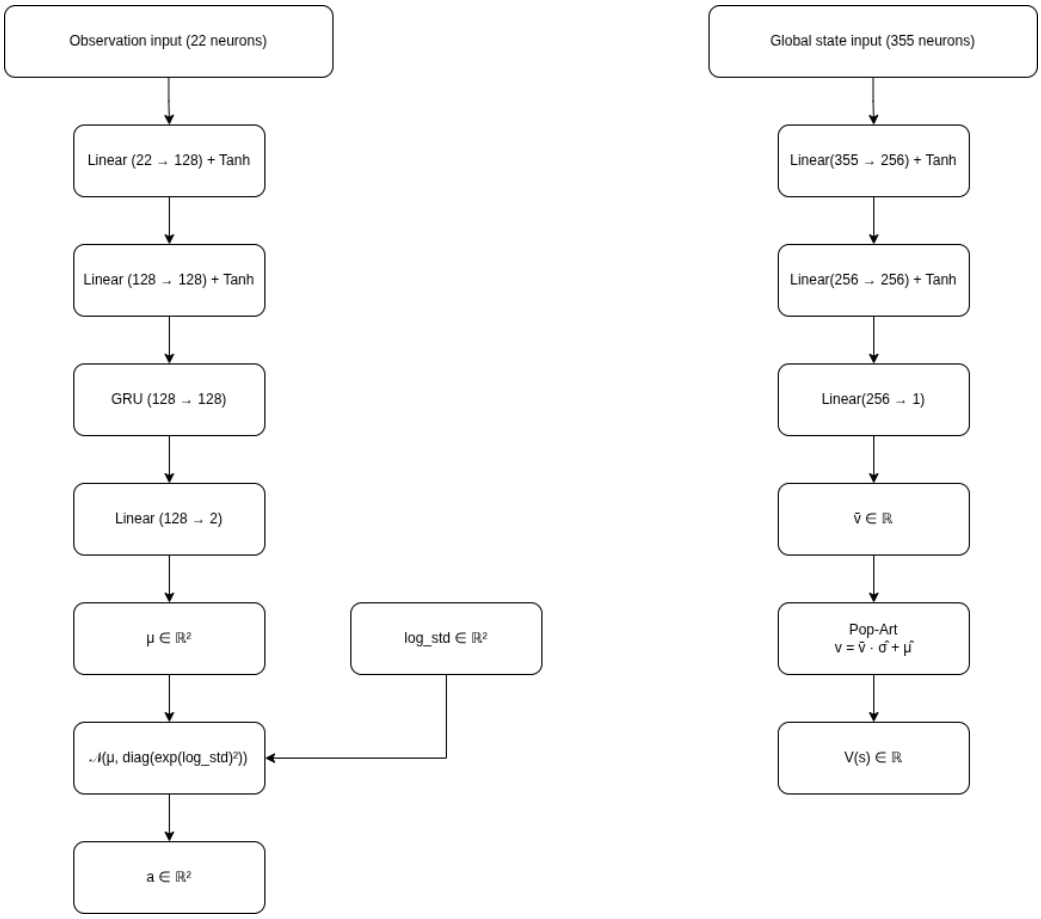


Figure 2: Actor and critic neural network layer specifications when $N = 24$.

4.2.8 Hyperparameter Search and Training

The learning performance of our MAPPO setup is sensitive to the hyperparameters used during training. To simplify hyperparameter search, we utilize a training curriculum. The initial training runs are for a simplified scenario with only two agents, a short initial distance to the target, the target's defensive systems disabled, and a fixed target spawn location. Once hyperparameters that produce satisfying learning outcomes are found, the complexity of the scenario is increased and the process is repeated. Reward function design is done in parallel with hyperparameter tuning, since changes in shaping terms and reward scales can change the effective optimization landscape and interact with the choice of hyperparameters.

Within a training run, the critic learning rate α_V and the actor learning rate α_π are scheduled to decrease by factors $\alpha_{V,f}$ and $\alpha_{\pi,f}$ every N_{half} learning updates to stabilize the learning process and prevent collapses in situations where the reward suddenly spikes or drops. A cosine-shaped entropy coefficient decay schedule is used to decrease exploration from $c_{\text{ent},0}$ to $c_{\text{ent},f}$ over time to increase the performance of the learned policy over training. To prevent overfitting to a single path, the spawn location and orientation of the agents is determined randomly in each episode. Table 4

outlines the training curriculum, and Table 5 lists the corresponding hyperparameter settings.

Stage	N (Agents)	Spawn distance (m)	Spawn cone	Defences	Target
1	2	1500	0°	Off	Static
2	2	5000	0°	Off	Static
3	2	5000-8000	0°	Off	Static
4	2	5000-8000	±10°	Off	Static
5	12	5000-8000	±10°	Off	Static
6	12	20000-40000	±10°	Off	Static
7	12	20000-40000	±22.5°	Off	Static
8	24	20000-40000	±22.5°	Off	Static
9	24	20000-40000	±22.5°	On	Static
10	24	25000-50000	±22.5°	On	Static

Table 4: Training curriculum used for hyperparameter tuning and reward design. Spawn cone refers to the initial bow bearing within which the target randomly spawns at the beginning of an episode.

Symbol	Value	Description
Simulation		
$T_{\max}$	$1.25 \cdot 10^5$	Maximum number of simulation steps per run.
k_{repeat}	40	Number of simulator steps executed per policy action.
Event and terminal rewards		
r_{hit}	30.0	Reward for a successful USV hit on the target.
r_{succ}	1440.0	Terminal reward granted when the episode ends in success.
r_{death}	60.0	Per-agent terminal penalty for agent losses at success.
r_{land}	0.5	Per-agent instantaneous penalty for agent losses by land barrier entry.
r_{detect}	2.5	Bonus when an agent first detects the target.
Coverage and search reward		
w_{explore}	0.07	Weight of the exploration or novelty term.
Δ_{cell}	1500 m	Side length of one spatial coverage cell.
N_{contact}	5	Consecutive contact steps required before state transition.
ρ_{vis}	0.1	Minimum fraction of alive USVs that must see the target.
N_{fade}	10	Number of steps over which the search reward fades out.
PBRS		
λ_{prog}	5.0	Weight of the progress-based potential term.
λ_{align}	3.0	Weight of the target-alignment potential term.
b_{exp}	32.0	Exponential base used in the shaping potential.
λ_{time}	5×10^{-4}	Per-step time penalty weight.
Rollout and discounting		
H	256	Rollout horizon in steps per environment before each PPO update.
N_{env}	32	Number of parallel environments used during training.
γ_{act}	0.99999	Discount factor at the policy-action timescale.
γ	auto	Per-step discount factor computed at runtime from $\gamma = \gamma_{\text{act}}^{1/k_{\text{repeat}}}$.
λ_{GAE}	0.95	Generalized Advantage Estimation smoothing parameter.
PPO optimization		
η_π	10^{-4}	Learning rate of the actor network.
η_V	8×10^{-4}	Learning rate of the critic network.
ϵ_{clip}	0.2	PPO clipping threshold for policy updates.
B	32768	Mini-batch size used during optimization.
K	3	Number of PPO epochs per update.
$\|g\|_{\max}$	0.5	Maximum gradient norm used for clipping.

Symbol	Value	Description
Policy distribution and entropy		
$c_{\text{ent},0}$	0.007	Initial entropy coefficient.
$c_{\text{ent},f}$	0.001	Final entropy coefficient.
s_{ent}	cosine	Entropy decay schedule.
ϕ_{hold}	0.1	Final training fraction with constant terminal entropy.
Value loss, LR schedule, and stabilization		
c_V	0.5	Weight of the value-loss term in the PPO objective.
s_{lr}	step_half	Learning-rate schedule used during training.
N_{half}	20	Number of updates between learning-rate halvings.
$\alpha_{\pi,f}$	0.95	Final multiplicative scaling of actor learning rate.
$\alpha_{V,f}$	0.98	Final multiplicative scaling of critic learning rate.
KL_{max}	0.003	KL-divergence threshold used for early stopping.
β_{PA}	3×10^{-4}	EMA rate for Pop-Art normalization statistics.
ε_{PA}	10^{-5}	Numerical stability constant for Pop-Art normalization.

Table 5: Simulation parameters and MAPPO hyperparameters.

The simulator and the MAPPO trainer are written in Python. NumPy is used for CPU-side environment data handling and auxiliary statistics in the simulator and the trainer, and PyTorch is used for neural network inference, tensor operations, automatic differentiation, and GPU-accelerated optimization. Multiple simulations can be run in parallel with multiprocessing. All training is performed in a containerized Debian 13 Linux environment on a Fedora 43 Linux host machine with a 4.80 GHz 22-core Intel Core Ultra 7 processor, 16 GiB of RAM, and an Nvidia GeForce RTX 4060 Max-Q mobile GPU with 8 GiB of VRAM. BF16 mixed-precision training is used to improve computational efficiency and reduce training time on the GPU. A 400-update training run with $N = 24$ and $T_{\text{max}} = 125000$ takes approximately 90 minutes.

5 Results

This section assesses the training metrics and the performance of the learned policy. The hyperparameters and detailed simulation parameters used in training are presented in Table 5.

5.1 Training

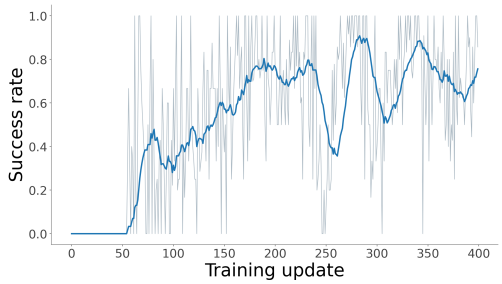
The policy is trained with 400 learning updates. During training, the model parameters are saved whenever the 15-update running average mission success rate (i.e. share of episodes that result in the target being destroyed) reaches a new high. Thus, the resulting policy corresponds to the best-performing checkpoint of the training run. The training scenario of the model is presented in Table 6.

Algorithm	N	Required hits	Spawn distance (km)	Spawn cone	Target	Defenses
MAPPO	24	3	25–50	±22.5°	Static	On

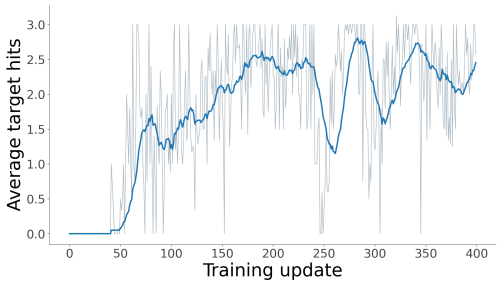
Table 6: Training scenario of the policy.

The policy was not trained to find and destroy a moving target because the reward function is incompatible with a moving target, resulting in inability to close in on the target during training. The progress and alignment rewards are formulated using the target's instantaneous position which is appropriate for a static target but problematic for a moving one, especially when using PBRS, as changes in the target's position can produce negative deltas even when the agents are behaving sensibly. Closing in on a moving target would likely require steering toward a future intercept point rather than directly toward the target's current location.

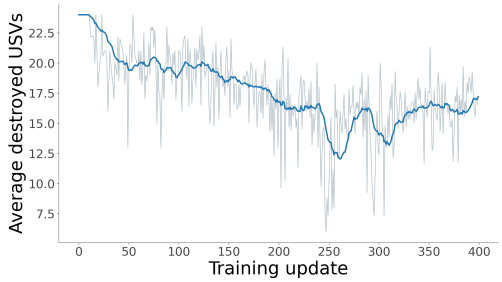
Figure 3 presents key performance metrics over the course of the policy's training run. The bright blue curve represents the running average and the light blue lines represent the raw metric values. The success rate in Figure 3a and the number of hits on target in Figure 3b increase over training, but begin to oscillate heavily after 250th update. The average number of USVs destroyed in Figure 3c decreases until the 250th update, and then levels off after the oscillations. At the same time, the agents learn to detect the target earlier (Figure 3d) and to produce the first hit in a shorter time (Figure 3e). The average total number of environment steps per episode shown in Figure 3f also decreases over the first 250 updates.



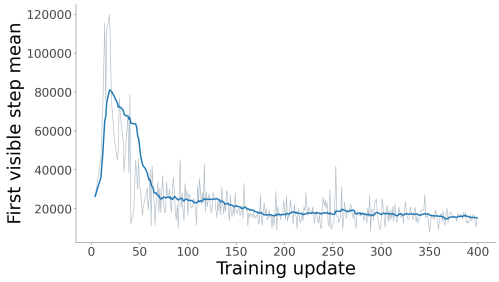
(a) Success rate.



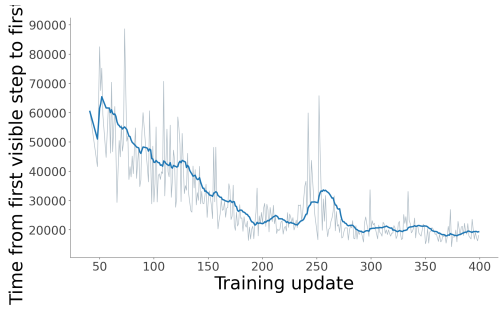
(b) Number of hits on target.



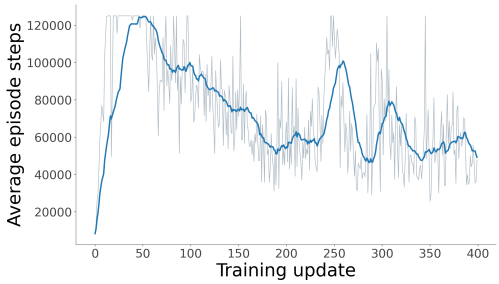
(c) USVs destroyed.



(d) Earliest target detection step.



(e) Steps from first detection to first hit.



(f) Environment steps of an episode.

Figure 3: Per-training-update average performance metrics of the policy over the training run.

5.2 Evaluation

The trained policy is evaluated by using it as the decentralized policy controller of the USV agents in the simulation environment. During evaluation, action selection is deterministic, meaning for each agent i, the learned policy π_θ maps the agent's local observation o_i to an action a_i, and actions are selected directly from the policy mean $a_{i,t} = \mu_\theta(o_{i,t}, h_{i,t})$, without stochastic sampling. Action repeat is disabled during evaluation to allow more fine-grained control, and the clip on the desired heading change is increased from $\pm 60^\circ$ used during training to $\pm 90^\circ$ to enable more agile maneuvers.

The policy is evaluated over 40 independent simulation episodes across different scenarios, and the reported performance metrics are computed as averages over these

evaluation episodes. The baseline evaluation scenario is based on the training setup. It features 24 agents and a static target with three functioning effectors (CIWS 1, CIWS 2, and a main gun) enabled. The target spawns at a random location 25-50 km from the center of the initial USV formation, within a ±22.5° relative angle from the bow bearing of the formation. The target requires three successful hits by the USVs to be destroyed. In the baseline scenario, the trained policy is able to find and destroy the target in 88% of the simulations. The further results of the evaluation runs are presented in Table 7.

To assess the robustness of the policy, communications and GNSS jamming (discussed in Section 4.1.2), which were not used in the training of the policy, are added to the baseline scenario. Communications jamming is modeled as a package loss with $P = 0.5$ within the visibility radius of the target, and GNSS jamming zeroes out global location values from the observation vector of each agent within the visibility radius of the target. As seen in Table 7, the policy is robust against jamming with only 0.10 lower success rate than in the baseline scenario. To further assess the robustness of the learned policy, a moving target is introduced to the baseline scenario. As evident in the results in Table 7, the agents are able to chase and accumulate some hits on the target despite being only trained to attack a static target, but as moving away from the USVs buys more time for the target to shoot at the USVs, the success rate is significantly lower and the number of destroyed USVs is higher. Designing a reward function that would enable chasing a moving target during training could potentially enable the USVs to learn more efficient tactics to destroy the target.

For comparison, Table 7 includes the evaluation results of an idealized rule-based reference controller. The controller uses distance-based mode switching between a long range formation approach, mid range zig-zag evasive steering, and close range terminal homing toward the target. Since the controller has complete information about the target's location throughout the simulation and the environment, it is able to steer toward the target continuously during each run. The rule-based controller does not use location information and the agents do not communicate, thus the controller was not evaluated in the jamming scenario. The rule-based reference controller outperformed the MAPPO-trained controller in every metric in baseline and moving target scenarios, although its performance also declined significantly in the moving target scenario. A visualization of the rule-based controller's strike path in the baseline scenario is presented in Figure 5.

Controller	Scenario	Mission success	Episode steps	Destroyed USVs	Hits on target	Timeout rate
MAPPO	Baseline	0.88	10466 ± 1935	11.39 ± 3.57	2.75 ± 0.89	0.10
Reference	Baseline	1.00	9783 ± 1735	7.88 ± 1.33	3.05 ± 0.22	0.00
MAPPO	Jamming	0.78	10544 ± 1982	10.94 ± 4.11	2.60 ± 0.86	0.18
Reference	Jamming	N/A	N/A	N/A	N/A	N/A
MAPPO	Moving target	0.05	16288 ± 3568	23.81 ± 0.80	0.18 ± 0.67	0.08
Reference	Moving target	0.20	14382 ± 2000	22.18 ± 3.22	0.93 ± 1.17	0.00

Table 7: Performance of the MAPPO and reference controllers under the baseline, jamming, and moving-target scenarios over 40 evaluation episodes per scenario. Values are reported as mean ± standard deviation. Episode steps and destroyed USV metrics only include episodes that were not terminated at timeout.

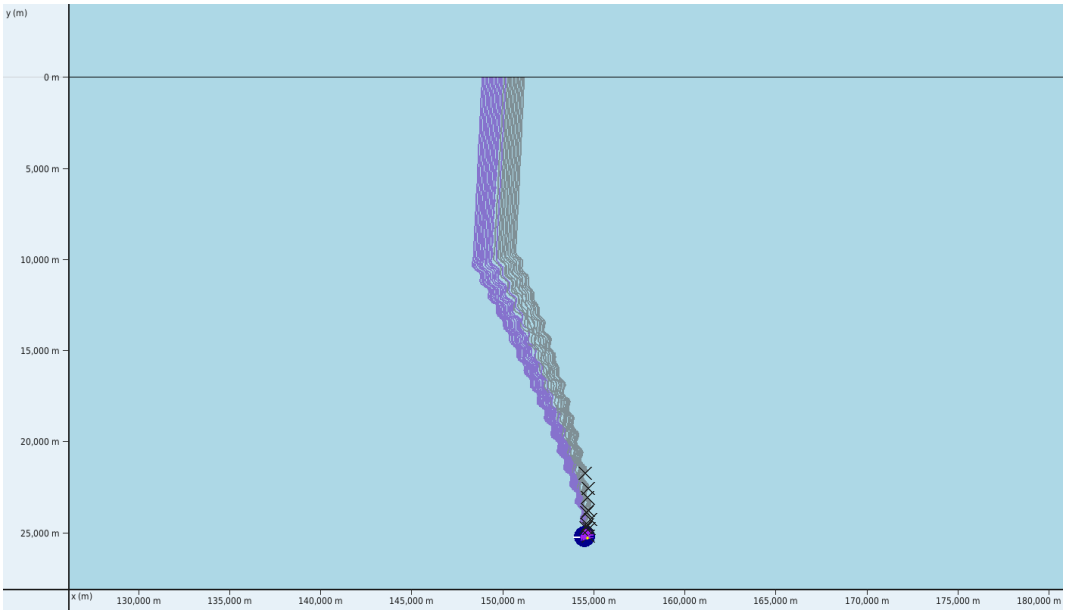


Figure 4: USVs striking a stationary target in the baseline scenario. The horizontal black line represents the land border, the blue circle represents the target, purple triangles represent alive USVs, and black crosses mark the locations of lost USVs. The paths of lost USVs are traced in grey and paths of alive USVs in purple. The graphical representations of the USVs and the target are not true to scale.

Despite the inclusion of the coverage-based search term in the reward function, the agents did not learn an effective search pattern. Instead, as shown in Figure 4, the agents travel in a straight line until the target becomes visible, after which they begin closing in on it. In the baseline evaluation scenario, 80% of the failed missions were timeouts, indicating that the agents failed to locate the target when it spawned outside their detection radius along this straight-line path. Similarly, in the jamming scenario, $\approx$ 78% of the failed missions ended in a timeout. In the moving target scenario, the target appears to have moved within the detection range of most agents, as the timeout rate was slightly lower (0.08) than in the baseline scenario (0.10).

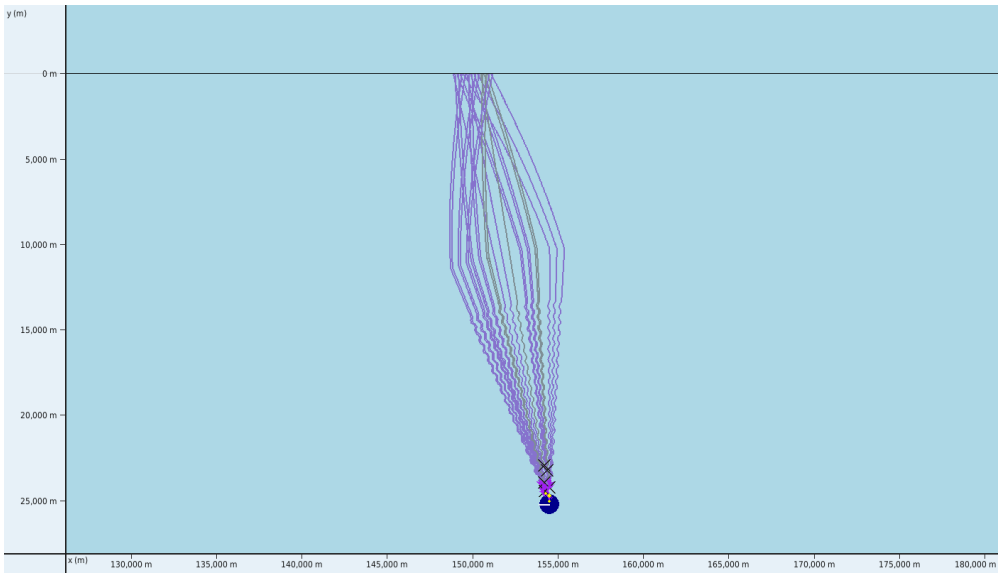


Figure 5: Rule-based reference controller guiding a swarm of USVs towards the target in the baseline scenario.

Interestingly, the agents learned a sine-wave-like zig-zag maneuver similar to that of the reference controller, which is activated once the target is detected. However, the wavelength and amplitude of the MAPPO controller's zig-zag are slightly larger: approximately 1000 m and 500 m, respectively, as shown in Figure 6. In contrast, the reference controller's zig-zag has a wavelength of about 750 m and an amplitude of about 250 m, as shown in Figure 7. Furthermore, the reference controller keeps the agents more spread out than the MAPPO controller, which may reduce their probability of being hit by the target's defenses. This may explain why the average number of destroyed USVs is significantly lower for the reference controller than for the MAPPO controller. The reason for the emergence of the zig-zag maneuver is not fully clear. The alignment reward r_t^{align} encourages movement toward the target and would favor a more direct approach. However, a straight approach makes the agents more predictable targets for the target's defenses, and the terminal loss penalty provides an incentive to reduce losses, which could explain learning the maneuver. Additionally, small repeated heading corrections can lead to oscillating movement patterns with continuous steering control, which may have contributed to the emergence of the zig-zag pattern. The less dispersed formation of the agents could possibly be explained by the lack of reward shaping in terms of the formation of the swarm.

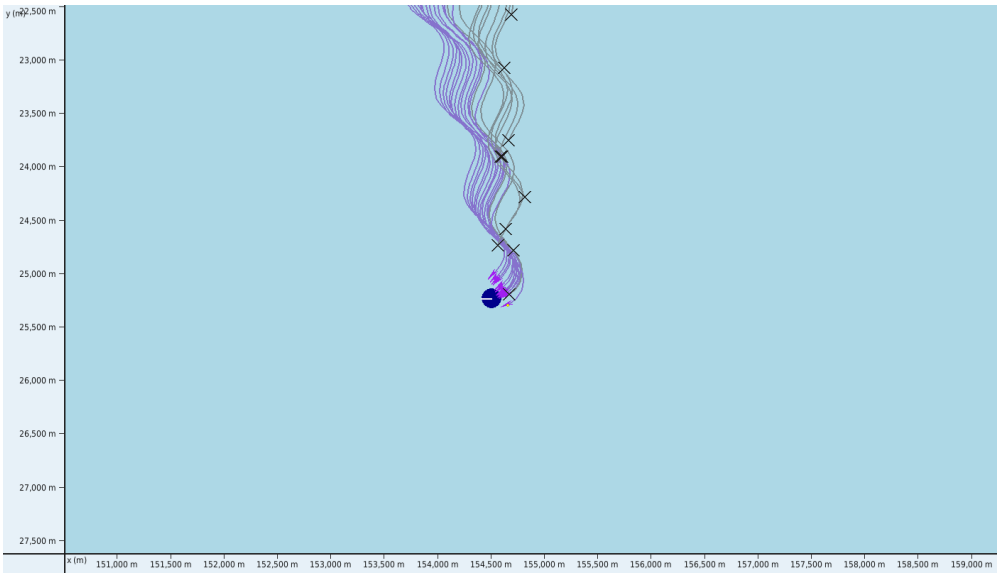


Figure 6: Close-up visualization of the MAPPO controller's final approach towards the target

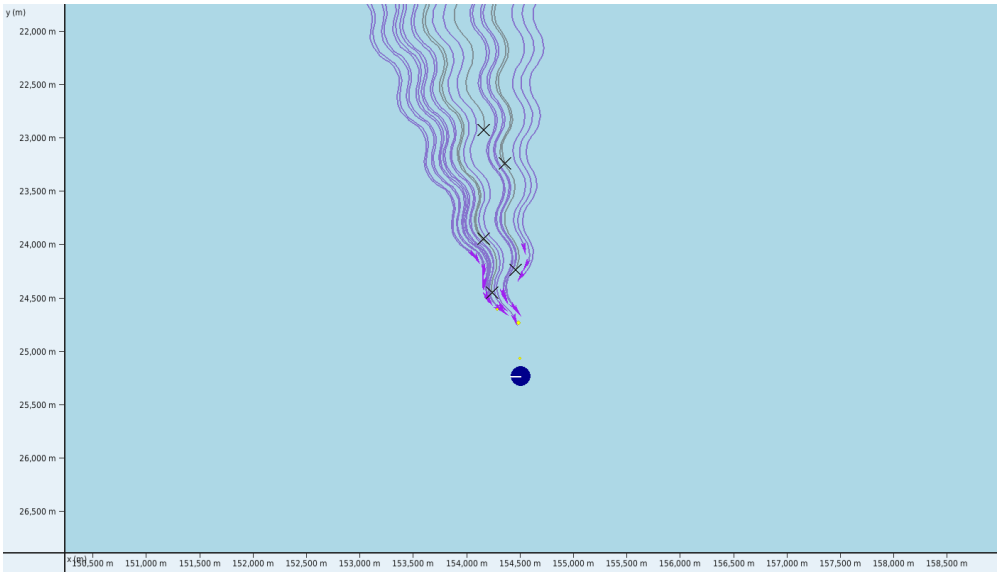


Figure 7: Close-up visualization of the reference controller's final approach towards the target

5.3 Discussion

The most important factor affecting the performance of the resulting controller is the design of the reward function, since MAPPO optimizes the expected return defined by that function. Another key factor is the training procedure. As shown in Figure 3, the training does not converge to a stable solution, but instead oscillates between average success rates of approximately 0.4 and 1.0. Because the training did not converge to

a stable 1.0 success rate, the trainer was not able to fully optimize the attack-phase behavior to minimize agent losses. The training procedure used a curriculum for hyperparameter search and reward function design, but not curriculum learning [94] in the actual training process, where the policy would be trained across a sequence of increasingly difficult scenarios.

Two opposing approaches could improve performance. The first is to reduce reward shaping. In the current setup, the shaping terms guide the agents toward behaviors such as approaching and aligning toward the target. With less shaping, the agents would have more freedom to discover their own tactics for locating and destroying the target. However, in this scenario, learning the task without shaping would likely be extremely difficult. One possible way to address this would be curriculum learning, in which training begins in a much simpler scenario, for example one where the target is placed directly in front of the agents. Once training converges in that setting, the learned policy could be used to initialize training in gradually more difficult scenarios until reaching the evaluation setting used in this thesis.

The opposite approach is to increase reward shaping. A search pattern could be encouraged by penalizing overly tight formations or by rewarding sweeping behavior during the search phase. Similarly, the reward function could encourage a more dispersed formation during the attack phase in order to reduce hits from the target, as seen in the reference controller. The downside is that stronger shaping may reduce the likelihood of discovering better tactics than the reference controller, as it steers the agents toward a solution defined more directly by the reward function rather than one discovered more freely through learning.

A third option is to seek a middle ground between these two extremes by redesigning the reward function more carefully. The objective would be to encourage search behavior and enable mission success in a curriculum setting, while keeping the shaping limited enough to preserve the agents' freedom to discover effective behavior. Training could initially focus on reaching a stable solution in which the target is destroyed consistently. After that, the objective could shift toward minimizing losses before gradually advancing to more difficult scenarios.

6 Summary

This thesis surveyed the fundamental concepts of MARL, swarming, and USVs, as well as recent developments in applying MARL to the control of USVs and other autonomous vehicles in various mission types. Building on these concepts and prior research, a MAPPO trainer was designed and built to allow a swarm of USVs to learn a policy for executing a hunting mission in which the swarm must locate and destroy a single surface combatant positioned at an unknown location.

The trained policy is able to locate and destroy the target with a success rate of 88%, which provides strong evidence that the MARL approach is feasible in this setting. For comparison, the rule-based reference controller achieved a 100% success rate with fewer destroyed USVs, but the controller had full knowledge of the target's position when majority of the MAPPO-trained controller's failed episodes were due to not locating the target. The MARL-based controller also demonstrated robustness by achieving a success rate of 78% under communications and GNSS jamming. Against a moving target, the success rate was significantly lower at 5%, indicating that the learned behavior did not generalize well to target motion. At the same time, the fact that the policy was still able to achieve some successes despite being trained only against a static target shows adaptability in the MAPPO-trained controller.

The training setup was not able to produce a policy that outperformed the rule-based reference controller, but the learned policy did exhibit non-trivial emergent behavior. While the agents did not learn an effective search pattern and instead moved approximately in a straight line until the target was detected, they did learn a sine-wave-like zig-zag attack maneuver after detection, similar to that of the reference controller. This suggests that the policy was able to discover a meaningful evasive strike behavior, even though its formation remained tighter and thus more prone to being hit by the target's defenses than that of the reference controller.

A plausible explanation for the lower performance than the reference controller is the combination of reward design and training dynamics. The training run did not converge to a stable 100% success-rate regime, but instead oscillated considerably, which likely limited the trainer's ability to refine the attack phase toward minimizing agent losses. In addition, the reward shaping during the search phase did not sufficiently incentivize broad exploration, which likely contributed to the absence of an efficient search pattern. The lack of shaping for swarm dispersion may likewise help explain why the learned attack formation remained more dense than that of the reference controller.

In future work, the reward terms could be reformulated using relative-motion quantities, such as closing speed, predicted miss distance, time to closest approach, or alignment with a predicted intercept point, in order to enable training against a moving target. To improve performance, curriculum learning could be incorporated into the training process so that the agents first learn to solve simpler versions of the task before progressing to the full two-stage search-and-destroy scenario. A more carefully redesigned reward function could also encourage more effective search behavior and greater dispersion during the strike phase while still leaving room for the discovery of novel tactics. For added realism, inter-USV collisions could be added to

the simulation together with a collision penalty in the reward function. Additionally, throttle could be included in the agent's action space to enable loitering. As a more substantial improvement, one could train two different policies for the search and strike phases, or utilize algorithms that allow individual policy and value functions for more heterogeneous agent actions, such as Independent PPO (IPPO) or MADDPG, although such approaches may be less stable than MAPPO in a 24-agent task.

Regardless of the limitations of the trained policy, the experiment demonstrated the potential of MARL, and especially MAPPO, for training a large number of agents in a challenging long-horizon two-phase task with sparse terminal rewards under partial observability. The thesis further contributes to the literature by studying USV swarm control in a setting that is more demanding than is typical in prior work, which often involves fewer agents, a single mission phase, shorter horizons, discrete environments, and targets without defensive capabilities.

References

[1] J. J. Andersson, S. Simon, and European Union Institute for Security Studies, Eds., *Minding the Drone Gap: Drone Warfare and the EU*. Luxembourg: Publications Office, 2024, 1 p., ISBN: 978-92-9462-266-2. DOI: 10.2815/578177.

[2] S. Nantogma, S. Zhang, X. Yu, X. An, and Y. Xu, "Multi-USV Dynamic Navigation and Target Capture: A Guided Multi-Agent Reinforcement Learning Approach", *Electronics*, vol. 12, no. 7, p. 1523, Mar. 23, 2023, ISSN: 2079-9292. DOI: 10.3390/electronics12071523.
URL: https://www.mdpi.com/2079-9292/12/7/1523 (visited on 08/24/2025).

[3] A. Howard, L. E. Parker, and G. S. Sukhatme, "The SDR Experience: Experiments with a Large-Scale Heterogeneous Mobile Robot Team", in *Experimental Robotics IX*, M. H. Ang and O. Khatib, Eds., red. by B. Siciliano, O. Khatib, and F. Groen, vol. 21, Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 121–130, ISBN: 978-3-540-28816-9 978-3-540-33014-1. DOI: 10.1007/11552246_12.
URL: http://link.springer.com/10.1007/11552246_12 (visited on 09/08/2025).

[4] Y. Liu, R. Song, R. Bucknall, and X. Zhang, "Intelligent Multi-Task Allocation and Planning for Multiple Unmanned Surface Vehicles (USVs) Using Self-Organising Maps and Fast Marching Method", *Information Sciences*, vol. 496, pp. 180–197, Sep. 2019, ISSN: 00200255. DOI: 10.1016/j.ins.2019.05.029.
URL: https://linkinghub.elsevier.com/retrieve/pii/S0020025519304323 (visited on 09/08/2025).

[5] K. Xue, Z. Huang, P. Wang, and Z. Xu, "An Exact Algorithm for Task Allocation of Multiple Unmanned Surface Vehicles with Minimum Task Time", *Journal of Marine Science and Engineering*, vol. 9, no. 8, p. 907, Aug. 22, 2021, ISSN: 2077-1312. DOI: 10.3390/jmse9080907.
URL: https://www.mdpi.com/2077-1312/9/8/907 (visited on 09/08/2025).

[6] A. Boretti, "Unmanned Surface Vehicles for Naval Warfare and Maritime Security", *The Journal of Defense Modeling and Simulation: Applications, Methodology, Technology*, vol. 23, no. 2, pp. 317–324, 2026. DOI: 10.1177/15485129241283056.
URL: https://doi.org/10.1177/15485129241283056 (visited on 04/26/2025).

[7] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement Learning in Robotics: A Survey", *The International Journal of Robotics Research*, vol. 32, pp. 1238–1274, Sep. 2013, ISSN: 978-3-642-27644-6. DOI: 10.1177/0278364913495721.

[8] J. Xia, Y. Luo, Z. Liu, Y. Zhang, H. Shi, and Z. Liu, "Cooperative Multi-Target Hunting by Unmanned Surface Vehicles Based on Multi-Agent Reinforcement Learning", *Defence Technology*, vol. 29, pp. 80–94, Nov. 2023, ISSN: 22149147. DOI: 10.1016/j.dt.2022.09.014.
URL: https://linkinghub.elsevier.com/retrieve/pii/S221491472200215X (visited on 04/26/2025).

[9] D. Silver, J. Schrittwieser, K. Simonyan, *et al.*, "Mastering the Game of Go Without Human Knowledge", *Nature*, vol. 550, no. 7676, pp. 354–359, Oct. 2017, ISSN: 0028-0836, 1476-4687. DOI: 10.1038/nature24270.
URL: https://www.nature.com/articles/nature24270 (visited on 03/17/2026).

[10] L. Vasankari, "Multi-Agent Reinforcement Learning for Littoral Naval Warfare", M.S. thesis, Aalto University School of Electrical Engineering, Espoo, Dec. 19, 2023, 92 pp.

[11] L. Vasankari, S. Rautio, and K. Virtanen. "Naval Strike Drone Effectiveness in Surface Warfare". (), pre-published.

[12] S. V. Albrecht, F. Christianos, and L. Schäfer, *Multi-Agent Reinforcement Learning: Foundations and Modern Approaches*. Cambridge, Massachusetts: The MIT Press, 2024, 366 pp., ISBN: 978-0-262-38051-5.

[13] X. Wang, S. Wang, X. Liang, *et al.*, "Deep Reinforcement Learning: A Survey", *IEEE Transactions on Neural Networks and Learning Systems*, vol. 35, no. 4, pp. 5064–5078, Apr. 2024, ISSN: 2162-237X, 2162-2388. DOI: 10.1109/TNNLS.2022.3207346.
URL: https://ieeexplore.ieee.org/document/9904958/ (visited on 08/24/2025).

[14] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction* (Adaptive Computation and Machine Learning Series), Second edition. Cambridge, Massachusetts: The MIT Press, 2018, 526 pp., ISBN: 978-0-262-03924-6.

[15] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and Acting in Partially Observable Stochastic Domains", *Artificial Intelligence*, vol. 101, no. 1–2, pp. 99–134, May 1998, ISSN: 00043702. DOI: 10.1016/S0004-3702(98)00023-X.
URL: https://linkinghub.elsevier.com/retrieve/pii/S000437029800023X (visited on 09/28/2025).

[16] R. Bellman, *Dynamic Programming*. Princeton, NJ: Princeton Univ. Pr, 1984, 339 pp., ISBN: 978-0-691-07951-6.

[17] R. A. Howard, *Dynamic Programming and Markov Processes*. Cambridge, Massachusetts: The MIT Press, 1960, 136 pp.

[18] R. S. Sutton, "Learning to Predict by the Methods of Temporal Differences", *Machine Learning*, vol. 3, no. 1, pp. 9–44, Aug. 1988, ISSN: 0885-6125, 1573-0565. DOI: 10.1007/BF00115009.
URL: http://link.springer.com/10.1007/BF00115009 (visited on 09/28/2025).

[19] C. Watkins, "Learning from Delayed Rewards", Ph.D. dissertation, University of Cambridge, England, 1989.

[20] S. Thrun and A. Schwartz, "Issues in Using Function Approximation for Reinforcement Learning", in *Proceedings of the 4th Connectionist Models Summer School*, Hillsdale, NJ, USA, 1993, pp. 255–263.

[21] H. van Hasselt, A. Guez, and D. Silver. "Deep Reinforcement Learning with Double Q-learning". arXiv: 1509.06461 [cs]. (Dec. 8, 2015),
URL: http://arxiv.org/abs/1509.06461 (visited on 09/28/2025), pre-published.

[22] H. V. Hasselt, "Double Q-learning", *Advances in Neural Information Processing Systems*, vol. 23, pp. 2613–2621, 2010.
URL: https://proceedings.neurips.cc/paper_files/paper/2010/file/091d584fced301b442654dd8c23b3fc9-Paper.pdf.

[23] G. Rummery and M. Niranjan, "On-Line Q-Learning Using Connectionist Systems", Cambridge University Engineering Department, England, CUED/F-INFENG/TR 166, 1994.

[24] H. van Hasselt, Y. Doron, F. Strub, M. Hessel, N. Sonnerat, and J. Modayil. "Deep Reinforcement Learning and the Deadly Triad". arXiv: 1812.02648 [cs]. (Dec. 6, 2018),
URL: http://arxiv.org/abs/1812.02648 (visited on 10/06/2025), pre-published.

[25] V. Mnih, A. P. Badia, M. Mirza, *et al.*, "Asynchronous Methods for Deep Reinforcement Learning", in *Proceedings of the 33rd International Conference on Machine Learning*, vol. 48, New York, NY, USA: JMLR: W&CP, 2016. DOI: 10.48550/arXiv.1602.01783.

[26] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. "Proximal Policy Optimization Algorithms". arXiv: 1707.06347 [cs]. (Aug. 28, 2017),
URL: http://arxiv.org/abs/1707.06347 (visited on 10/17/2025), pre-published.

[27] L. S. Shapley, "Stochastic Games", in *Proceedings of the National Academy of Sciences of the United States of America*, vol. 39, 1953, pp. 1095–1100. DOI: 10.1073/pnas.39.10.1095.

[28] K. Zhang, Z. Yang, and T. Başar. "Multi-Agent Reinforcement Learning: A Selective Overview of Theories and Algorithms". arXiv: 1911.10635 [cs]. (Apr. 28, 2021),
URL: http://arxiv.org/abs/1911.10635 (visited on 05/29/2025), pre-published.

[29] M. L. Littman, "Markov Games as a Framework for Multi-Agent Reinforcement Learning", in *Machine Learning Proceedings 1994*, Elsevier, 1994, pp. 157–163, ISBN: 978-1-55860-335-6. DOI: 10.1016/B978-1-55860-335-6.50027-1.
URL: https://linkinghub.elsevier.com/retrieve/pii/B9781558603356500271 (visited on 10/11/2025).

[30] E. A. Hansen, D. S. Bernstein, and S. Zilberstein, "Dynamic Programming for Partially Observable Stochastic Games", in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 4, 2004, pp. 709–715.

[31] F. A. Oliehoek and C. Amato, *A Concise Introduction to Decentralized POMDPs* (SpringerBriefs in Intelligent Systems). Berlin: Springer International Publishing, 2016, ISBN: 978-3-319-28927-4 978-3-319-28929-8. DOI: 10.1007/978-3-319-28929-8.
URL: http://link.springer.com/10.1007/978-3-319-28929-8 (visited on 10/04/2025).

[32] L. Yuan, Z. Zhang, L. Li, C. Guan, and Y. Yu. "A Survey of Progress on Cooperative Multi-agent Reinforcement Learning in Open Environment". arXiv: 2312.01058 [cs]. (Dec. 2, 2023),
URL: http://arxiv.org/abs/2312.01058 (visited on 05/29/2025), pre-published.

[33] F. Christianos, G. Papoudakis, A. Rahman, and S. V. Albrecht, "Scaling Multi-Agent Reinforcement Learning with Selective Parameter Sharing", in *Proceedings of the 38th International Conference on Machine Learning*, arXiv, Jun. 12, 2021. DOI: 10.48550/arXiv.2102.07475. arXiv: 2102.07475 [cs].
URL: http://arxiv.org/abs/2102.07475 (visited on 10/19/2025).

[34] F. L. D. Silva and A. H. R. Costa, "A Survey on Transfer Learning for Multiagent Reinforcement Learning Systems", *Journal of Artificial Intelligence Research*, vol. 64, pp. 645–703, Mar. 11, 2019, ISSN: 1076-9757. DOI: 10.1613/jair.1.11396.
URL: https://jair.org/index.php/jair/article/view/11396 (visited on 10/19/2025).

[35] X. Zhao and Y. Xie, "Multi-level Advantage Credit Assignment for Cooperative Multi-Agent Reinforcement Learning", in *Proceedings of the 8th International Conference on Artificial Intelligence and Statistics*, vol. 258, Mai Khao, Thailand: PMLR, 2025.

[36] G. Papoudakis and F. Christianos, "Benchmarking Multi-Agent Deep Reinforcement Learning Algorithms in Cooperative Tasks", in *Proceedings of the 35th Conference on Neural Information Processing Systems*, 2021.

[37] X. Lyu, A. Baisero, Y. Xiao, B. Daley, and C. Amato. "On Centralized Critics in Multi-Agent Reinforcement Learning". arXiv: 2408.14597 [cs]. (Aug. 26, 2024),
URL: http://arxiv.org/abs/2408.14597 (visited on 10/20/2025), pre-published.

[38] K. Son, D. Kim, W. J. Kang, D. Hostallero, and Y. Yi, "QTRAN: Learning to Factorize with Transformation for Cooperative Multi-Agent Reinforcement learning", in *Proceedings of the 36th International Conference on Machine Learning*, Long Beach, California, USA, 2019.

[39] G. Papadopoulos, A. Kontogiannis, F. Papadopoulou, C. Poulianou, I. Koumentis, and G. Vouros, "An Extended Benchmarking of Multi-Agent Reinforcement Learning Algorithms in Complex Fully Cooperative Tasks", in *Proceedings of the 24th International Conference on Autonomous Agents and Multiagent Systems*, Detroit, Michigan, USA: IFAAMAS, May 2025. DOI: 10.48550/arXiv.2502.04773. arXiv: 2502.04773 [cs].
URL: http://arxiv.org/abs/2502.04773 (visited on 10/20/2025).

[40] C. Yu, A. Velu, E. Vinitsky, *et al.*, "The Surprising Effectiveness of PPO in Cooperative Multi-Agent Games", in *Proceedings of the 36th International Conference on Neural Information Processing Systems*, 2022, pp. 24 611–24 624.

[41] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-Dimensional Continuous Control Using Generalized Advantage Estimation", in *ICLR 2016*, arXiv, Oct. 20, 2018. DOI: 10.48550/arXiv.1506.02438. arXiv: 1506.02438 [cs].
URL: http://arxiv.org/abs/1506.02438 (visited on 10/25/2025).

[42] W. Hu, L. Xiao, and J. Pennington, "Provable Benefit of Orthogonal Initialization in Optimizing Deep Linear Networks", in *ICLR 2020*, arXiv, Jan. 16, 2020. DOI: 10.48550/arXiv.2001.05992. arXiv: 2001.05992 [cs].
URL: http://arxiv.org/abs/2001.05992 (visited on 10/25/2025).

[43] H. van Hasselt, A. Guez, M. Hessel, V. Mnih, and D. Silver. "Learning Values Across Many Orders of Magnitude". arXiv: 1602.07714 [cs]. (Aug. 16, 2016),
URL: http://arxiv.org/abs/1602.07714 (visited on 03/02/2026), pre-published.

[44] D. P. Kingma and J. Lei, "Adam: A Method for Stochastic Optimization", presented at the 3rd International Conference for Learning Representations, San Diego, USA, 2015. DOI: 10.48550/arXiv.1412.6980.

[45] S. Garnier, J. Gautrais, and G. Theraulaz, "The Biological Principles of Swarm Intelligence", *Swarm Intelligence*, vol. 1, no. 1, pp. 3–31, Oct. 17, 2007, ISSN: 1935-3812, 1935-3820. DOI: 10.1007/s11721-007-0004-y.
URL: http://link.springer.com/10.1007/s11721-007-0004-y (visited on 07/17/2025).

[46] Z. Jakšić, S. Devi, O. Jakšić, and K. Guha, "A Comprehensive Review of Bio-Inspired Optimization Algorithms Including Applications in Microelectronics and Nanophotonics", *Biomimetics*, vol. 8, no. 3, p. 278, Jun. 28, 2023, ISSN: 2313-7673. DOI: 10.3390/biomimetics8030278.
URL: https://www.mdpi.com/2313-7673/8/3/278 (visited on 07/17/2025).

[47] A. Singh, S. M. H. Mousavi, and K. Gaurav. "SHS: Scorpion Hunting Strategy Swarm Algorithm". arXiv: 2407.14202 [cs]. (Aug. 30, 2024),
URL: http://arxiv.org/abs/2407.14202 (visited on 09/08/2025), pre-published.

[48] C. W. Reynolds, "Flocks, Herds, and Schools: A Distributed Behavioral Model", *Computer Graphics*, vol. 21, no. 4, pp. 273–282, Jun. 1987. DOI: 10.1145/280811.281008.
URL: https://dl.acm.org/doi/10.1145/280811.281008 (visited on 07/17/2025).

[49] T. Vicsek, A. Czirok, E. Ben-Jacob, I. Cohen, and O. Sochet, "Novel Type of Phase Transition in a System of Self-driven Particles", *Physical Review Letters*, vol. 75, no. 6, pp. 1226–1229, Aug. 7, 1995, ISSN: 0031-9007, 1079-7114. DOI: 10.1103/PhysRevLett.75.1226. arXiv: cond-mat/0611743.
URL: http://arxiv.org/abs/cond-mat/0611743 (visited on 08/16/2025).

[50] A. Jadbabaie, J. Lin, and A. S. Morse, "Coordination of Groups of Mobile Autonomous Agents Using Nearest Neighbor Rules", *IEEE Transactions on Automatic Control*, vol. 48, no. 6, pp. 988–11 001, Jun. 2003, ISSN: 1558-2523. DOI: 10.1109/TAC.2003.812781.

[51] F. Cucker and S. Smale, "Emergent Behavior in Flocks", *IEEE Transactions on Automatic Control*, vol. 52, no. 5, pp. 852–862, May 2007, ISSN: 0018-9286. DOI: 10.1109/TAC.2007.895842.
URL: http://ieeexplore.ieee.org/document/4200853/ (visited on 08/17/2025).

[52] H. Zhang, P. Nie, Y. Sun, and Y. Shi, "Fixed-time Flocking Problem of a Cucker–Smale Type Self-Propelled Particle Model", *Journal of the Franklin Institute*, vol. 357, no. 11, pp. 7054–7068, Jul. 2020, ISSN: 00160032. DOI: 10.1016/j.jfranklin.2020.05.012.
URL: https://linkinghub.elsevier.com/retrieve/pii/S0016003220303501 (visited on 08/17/2025).

[53] R. Olfati-Saber, "Flocking for Multi-Agent Dynamic Systems: Algorithms and Theory", *IEEE Transactions on Automatic Control*, vol. 51, no. 3, pp. 401–420, Mar. 2006, ISSN: 0018-9286. DOI: 10.1109/TAC.2005.864190.
URL: http://ieeexplore.ieee.org/document/1605401/ (visited on 08/23/2025).

[54] M. Dorigo, G. Theraulaz, and V. Trianni, "Swarm Robotics: Past, Present, and Future [Point of View]", *Proceedings of the IEEE*, vol. 109, no. 7, pp. 1152–1165, Jul. 2021, ISSN: 0018-9219, 1558-2256. DOI: 10.1109/JPROC.2021.3072740.
URL: https://ieeexplore.ieee.org/document/9460560/ (visited on 08/24/2025).

[55] M. Rubenstein, C. Ahler, N. Hoff, A. Cabrera, and R. Nagpal, "Kilobot: A Low Cost Robot with Scalable Operations Designed for Collective Behaviors", *Robotics and Autonomous Systems*, vol. 62, no. 7, pp. 966–975, Jul. 2014, ISSN: 09218890. DOI: 10.1016/j.robot.2013.08.006.
URL: https://linkinghub.elsevier.com/retrieve/pii/S0921889013001474 (visited on 08/30/2025).

[56] C. Perry. "A Self-Organizing Thousand-Robot Swarm", seas.harvard.edu. (Aug. 2014),
URL: https://seas.harvard.edu/news/2014/08/self-organizing-thousand-robot-swarm (visited on 08/24/2025).

[57] M. Duarte, V. Costa, J. Gomes, *et al.*, "Evolution of Collective Behaviors for a Real Swarm of Aquatic Surface Robots", *PLOS ONE*, vol. 11, no. 3, L. Wang, Ed., e0151834, Mar. 21, 2016, ISSN: 1932-6203. DOI: 10.1371/journal.pone.0151834.
URL: https://dx.plos.org/10.1371/journal.pone.0151834 (visited on 08/31/2025).

[58] P. Zahadat and T. Schmickl, "Division of Labor in a Swarm of Autonomous Underwater Robots by Improved Partitioning Social Inhibition", *Adaptive Behavior*, vol. 24, no. 2, pp. 87–101, Apr. 2016, ISSN: 1059-7123, 1741-2633. DOI: 10.1177/1059712316633028.
URL: https://journals.sagepub.com/doi/10.1177/1059712316633028 (visited on 08/31/2025).

[59] G. Vásárhelyi, C. Virágh, G. Somorjai, T. Nepusz, A. E. Eiben, and T. Vicsek, "Optimized Flocking of Autonomous Drones in Confined Environments", *Science Robotics*, vol. 3, no. 20, eaat3536, Jul. 25, 2018, ISSN: 2470-9476. DOI: 10.1126/scirobotics.aat3536.
URL: https://www.science.org/doi/10.1126/scirobotics.aat3536 (visited on 08/31/2025).

[60] X. Zhou, X. Wen, Z. Wang, *et al.*, "Swarm of Micro Flying Robots in the Wild", *Science Robotics*, vol. 7, no. 66, eabm5954, May 25, 2022, ISSN: 2470-9476. DOI: 10.1126/scirobotics.abm5954.
URL: https://www.science.org/doi/10.1126/scirobotics.abm5954 (visited on 08/31/2025).

[61] Y. Zhang, Y. Hu, Y. Song, D. Zou, and W. Lin, "Back to Newton's Laws: Learning Vision-based Agile Flight via Differentiable Physics", *Nature Machine Intelligence*, vol. 7, no. 6, pp. 954–966, Jun. 16, 2025, ISSN: 2522-5839. DOI:

10.1038/s42256-025-01048-0. arXiv: 2407.10648 [cs].
URL: http://arxiv.org/abs/2407.10648 (visited on 08/31/2025).

[62] L. D. Stone, *Theory of Optimal Search* (Mathematics in Science and Engineering v. 118). New York: Academic Press, 1975, ISBN: 978-0-12-672450-9.

[63] S. Binitha and S. S Siva, "A Survey of Bio inspired Optimization Algorithms", *International Journal of Soft Computing and Engineering*, vol. 2, no. 2, pp. 137–151, May 2012, ISSN: 2231-2307.

[64] O. Walker, F. Vanegas, and F. Gonzalez, "A Framework for Multi-Agent UAV Exploration and Target-Finding in GPS-Denied and Partially Observable Environments", *Sensors*, vol. 20, no. 17, p. 4739, Aug. 21, 2020, ISSN: 1424-8220. DOI: 10.3390/s20174739.
URL: https://www.mdpi.com/1424-8220/20/17/4739 (visited on 09/15/2025).

[65] R. M. Francos and A. M. Bruckstein, "Search for Smart Evaders with Swarms of Sweeping Agents- a Resource Allocation Perspective", *Journal of Intelligent & Robotic Systems*, vol. 106, no. 4, p. 76, Dec. 2022, ISSN: 0921-0296, 1573-0409. DOI: 10.1007/s10846-022-01783-1.
URL: https://link.springer.com/10.1007/s10846-022-01783-1 (visited on 09/14/2025).

[66] L. Yang, Y. Hao, J. Xu, and M. Li, "Multi-UAV Collaborative Target Search Method in Unknown Dynamic Environment", *Sensors*, vol. 24, no. 23, p. 7639, Nov. 29, 2024, ISSN: 1424-8220. DOI: 10.3390/s24237639.
URL: https://www.mdpi.com/1424-8220/24/23/7639 (visited on 09/15/2025).

[67] Y. Ma, H. Li, X. Meng, *et al.* "Path Planning for Submarine Search with Cooperative Coverage of UAVs Cluster Based on Irregular Sea Area Uniform Segmentation". (2023),
URL: https://www.ssrn.com/abstract=4585924 (visited on 09/15/2025), pre-published.

[68] W. Yue, Y. Xi, and X. Guan, "A New Searching Approach Using Improved Multi-Ant Colony Scheme for Multi-UAVs in Unknown Environments", *IEEE Access*, vol. 7, pp. 161 094–161 102, 2019, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2019.2949249.
URL: https://ieeexplore.ieee.org/document/8882339/ (visited on 09/15/2025).

[69] L. Bertuccelli and J. How, "Search for Dynamic Targets with Uncertain Probability Maps", in *2006 American Control Conference*, Minneapolis, MN, USA: IEEE, 2006, 6 pp. ISBN: 978-1-4244-0209-0. DOI: 10.1109/ACC.2006.1655444.
URL: http://ieeexplore.ieee.org/document/1655444/ (visited on 09/15/2025).

[70] S. Liu, W. Yao, X. Zhu, Y. Zuo, and B. Zhou, "Emergent Search of UAV Swarm Guided by the Target Probability Map", *Applied Sciences*, vol. 12, no. 10, p. 5086, May 18, 2022, ISSN: 2076-3417. DOI: 10.3390/app12105086. URL: https://www.mdpi.com/2076-3417/12/10/5086 (visited on 09/15/2025).

[71] Y. Wang, X. Wang, W. Zhou, H. Yan, and S. Xie, "Threat Potential Field Based Pursuit–Evasion Games for Underactuated Unmanned Surface Vehicles", *Ocean Engineering*, vol. 285, p. 115 381, Oct. 2023, ISSN: 00298018. DOI: 10.1016/j.oceaneng.2023.115381. URL: https://linkinghub.elsevier.com/retrieve/pii/S0029801823017651 (visited on 11/17/2025).

[72] J. S. Corbett, *Some Principles of Maritime Strategy*. London, UK, 1911. URL: https://www.gutenberg.org/files/15076/15076-h/15076-h.htm.

[73] U. Navy, "Naval Warfare", *Naval Doctrine Publication 1*, Apr. 2020. URL: https://cimsec.org/wp-content/uploads/2020/08/NDP1_April2020.pdf.

[74] A. T. Mahan, *The Influence of Sea Power Upon History, 1660-1783*, 12th ed. Boston, MA, USA: Little, Brown and Company, Dec. 1890, ISBN: 978-1-4209-4847-9. URL: https://www.gutenberg.org/files/13529/13529-h/13529-h.htm (visited on 09/07/2025).

[75] G. Till, *Seapower: A Guide for the Twenty-First Century*, 2nd. ed. London: Routledge, 2009, 409 pp., ISBN: 978-0-203-88048-7.

[76] A. Bursuc, C. Munteanu, and S. Rus, "Overview on Sea Drones Evolution and their Use in Modern Warfare", *Land Forces Academy Review*, vol. 29, no. 2, pp. 195–209, Jun. 1, 2024, ISSN: 2247-840X. DOI: 10.2478/raft-2024-0021. URL: https://www.sciendo.com/article/10.2478/raft-2024-0021 (visited on 09/01/2025).

[77] T. Dimitrov, "Aspects in Usage of Unmanned Surface Vehicle in Ukranian War", *Voenen Zhurnal*, pp. 429–438, Apr. 2023. URL: https://rndc.bg/wp-content/uploads/2024/01/9-ASPECTS-IN-USAGE-OF-UNMANNED.pdf.

[78] J. Han, Y. Cho, J. Kim, J. Kim, N.-s. Son, and S. Y. Kim, "Autonomous Collision Detection and Avoidance for ARAGON USV: Development and Field Tests", *Journal of Field Robotics*, vol. 37, no. 6, pp. 987–1002, Sep. 2020, ISSN: 1556-4959, 1556-4967. DOI: 10.1002/rob.21935. URL: https://onlinelibrary.wiley.com/doi/10.1002/rob.21935 (visited on 09/06/2025).

[79] H. I. Sutton. "Overview Of Maritime Drones (USVs) Of The Russo-Ukrainian War, 2022-24", H I Sutton - Covert Shores. (Jun. 20, 2025), URL: http://www.hisutton.com/Russia-Ukraine-USVs-2024.html (visited on 09/06/2025).

[80] Y. Troshkin, "The Role of Naval Srike Drones in the Russia-Ukraine War", *Political Science and Security Studies Journal*, vol. 5, no. 2, Jun. 30, 2024, ISSN: 2719-6410. DOI: 10.5281/ZENODO.12701509. URL: https://zenodo.org/doi/10.5281/zenodo.12701509 (visited on 05/29/2025).

[81] E. M. de Andrade, J. S. Sales, and A. C. Fernandes, "Operative Unmanned Surface Vessels (USVs): A Review of Market-Ready Solutions", *Automation*, vol. 6, no. 17, Apr. 23, 2025. DOI: 10.3390/automation6020017.

[82] P. Zhang and G. Li, "A Cooperative Dynamic Target Search Approach for Multi-UAV Systems Utilizing the MAPPO Algorithm", *Discover Artificial Intelligence*, vol. 5, no. 1, p. 153, Jul. 17, 2025, ISSN: 2731-0809. DOI: 10.1007/s44163-025-00411-9. URL: https://link.springer.com/10.1007/s44163-025-00411-9 (visited on 10/24/2025).

[83] Y. Li, M. Ma, J. Cao, G. Luo, D. Wang, and W. Chen, "A Method for Multi-AUV Cooperative Area Search in Unknown Environment Based on Reinforcement Learning", *Journal of Marine Science and Engineering*, vol. 12, no. 7, p. 1194, Jul. 16, 2024, ISSN: 2077-1312. DOI: 10.3390/jmse12071194. URL: https://www.mdpi.com/2077-1312/12/7/1194 (visited on 09/15/2025).

[84] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor", in *Proceedings of the 35th International Conference on Machine Learning*, Stockholm, Sweden: PMLR 80, 2018. DOI: 10.48550/arXiv.1801.01290. URL: https://proceedings.mlr.press/v80/haarnoja18b/haarnoja18b.pdf.

[85] Y. Zhou, Y. Yue, B. Yan, L. Li, J. Xiao, and Y. Yao, "Collaborative Target Tracking Algorithm for Multi-Agent Based on MAPPO and BCTD", *Drones*, vol. 9, no. 8, p. 521, Jul. 24, 2025, ISSN: 2504-446X. DOI: 10.3390/drones9080521. URL: https://www.mdpi.com/2504-446X/9/8/521 (visited on 10/24/2025).

[86] C. Zhang, R. Zeng, B. Lin, Y. Zhang, W. Xie, and W. Zhang, "Multi-USV Cooperative Target Encirclement Through Learning-Based Distributed Transferable Policy and Experimental Validation", *Ocean Engineering*, vol. 318, p. 120 124, Feb. 2025, ISSN: 00298018. DOI: 10.1016/j.oceaneng.2024.120124. URL: https://linkinghub.elsevier.com/retrieve/pii/S0029801824034620 (visited on 10/30/2025).

[87] O. Khatib, "Real-Time Obstacle Avoidance for Manipulators and Mobile Robots", in *Proceedings of the 1985 IEEE International Conference on Robotics and Automation*, vol. 2, 1985, pp. 500–505.

[88] M. Bellemare, J. Veness, and M. Bowling, "Investigating Contingency Awareness Using Atari 2600 Games", *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 26, no. 1, pp. 864–871, 2012, ISSN: 2374-3468, 2159-5399. DOI: 10.1609/aaai.v26i1.8321.
URL: https://ojs.aaai.org/index.php/AAAI/article/view/8321 (visited on 02/28/2026).

[89] S. Kalyanakrishnan, S. Aravindan, V. Bagdawat, *et al.* "An Analysis of Frame-skipping in Reinforcement Learning". arXiv: 2102.03718 [cs]. (Feb. 7, 2021),
URL: http://arxiv.org/abs/2102.03718 (visited on 12/15/2025), pre-published.

[90] S. Sharma, A. Srinivas, and B. Ravindran, "Learning to Repeat: Fine Grained Action Repetition for Deep Reinforcement Learning", presented at the ICLR, 2017. DOI: 10.48550/arXiv.1702.06054. arXiv: 1702.06054 [cs].
URL: http://arxiv.org/abs/1702.06054 (visited on 02/28/2026).

[91] A. Y. Ng, D. Harada, and S. J. Russell, "Policy Invariance Under Reward Transformations: Theory and Application to Reward Shaping", in *Proceedings of the Sixteenth International Conference on Machine Learning*, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999, pp. 278–287.

[92] F. Rosenblatt, "The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain", *Psychological Review*, vol. 65, no. 6, pp. 386–408, 1958, ISSN: 1939-1471, 0033-295X. DOI: 10.1037/h0042519.
URL: https://doi.apa.org/doi/10.1037/h0042519 (visited on 03/02/2026).

[93] K. Cho, B. Van Merrienboer, C. Gulcehre, *et al.*, "Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation", in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Doha, Qatar: Association for Computational Linguistics, 2014, pp. 1724–1734. DOI: 10.3115/v1/D14-1179.
URL: http://aclweb.org/anthology/D14-1179 (visited on 03/02/2026).

[94] S. Narvekar, B. Peng, M. Leonetti, J. Sinapov, M. E. Taylor, and P. Stone, "Curriculum Learning for Reinforcement Learning Domains", *Journal of Machine Learning Research*, vol. 21, no. 1, pp. 1–50, Jan. 2020. DOI: 10.5555/3455716.3455897.