UNIVERSITY OF MISKOLC
FACULTY OF MECHANICAL ENGINEERING AND INFORMATICS

DEEP REINFORCEMENT LEARNING FOR SWARM ROBOTIC SYSTEMS

PHD THESES

Prepared by

Alaa Iskandar
Engineering of Mechatronics (BSc),
Engineering of Mechatronics (MSc)

ISTVÁN SÁLYI DOCTORAL SCHOOL OF MECHANICAL ENGINEERING SCIENCES
TOPIC FIELD OF BASIC ENGINEERING SCIENCES
TOPIC GROUP OF MECHANICS OF SOLIDS

Head of Doctoral School

Dr. Gabriella Bognár
DSc, Full Professor

Head of Topic Group

Dr. Páczelt István

Scientific Supervisor

Dr. Béla Kovács

Miskolc
2025

CONTENTS

SUPERVISOR'S RECOMMENDATIONS

Date 22/08/2024

Alaa Iskandar is a Syrian PhD candidate specializing in mechatronics engineering. During his M.Sc. studies, he focused on multi-agent systems as a form of distributed artificial intelligence to manage micro-electrical grids. He commenced his doctoral studies at the University of Miskolc in the autumn of 2020 under the Stipendium Hungaricum scholarship program.

Alaa has consistently demonstrated academic excellence, successfully passing all his examinations and actively engaging in scientific research within the field of swarm robotics. His research contributions exemplify his work on tasks such as the "foraging task" using swarm robots, showcasing his ability to apply advanced concepts to practical challenges.

Alaa Iskandar is an exceptionally dedicated researcher. He has authored eight scientific publications, five of which are high-quality articles published in Scopus-indexed journals, all co-authored with his supervisor. Additionally, he has frequently presented his work at national and international scientific conferences, further highlighting his commitment to disseminating knowledge and contributing to the scientific community.

The most significant contribution of Alaa's research lies in developing novel models for swarm robotics systems. His work includes the introduction of curriculum learning integrated with reinforcement learning to enhance the generalization of these systems. Furthermore, he has developed a hybrid model for foraging behavior that combines behavioral design with automatic design methods, such as particle swarm optimization and reinforcement learning, to leverage the strengths of both approaches. Additionally, he has proposed a continuous inverse reinforcement learning framework to address the diverse behaviors within swarm robotics systems.

In light of these achievements, I, Dr. Béla Kovács, Associate Professor and supervisor of Alaa Iskandar, consider his PhD studies highly successful and commend his contributions to mechatronics and swarm robotics.

Supervisor:
Dr. Béla Kovács

LIST OF SYMBOLS AND ABBREVIATIONS

GREEK LETTERS

$\alpha_{current}$	The angle between the robot and the nest with [0,2π] (rad).
α, β	Control the influence of the pheromone trail and heuristic information.
γ	The discount factor.
ΔD	The difference between the current state and the previous state.
ϵ	A hyperparameter, typically small (e.g., 0.1 or 0.2), determines the range within which the ratio $r_t(\theta)$ is allowed to vary without being clipped.
τ	The trajectories (a sequence of states and actions) in the dataset *K*.
$\Lambda_{ij}(t)$	The pheromone level on the path from node i to node j at time t.
ρ	The evaporation rate, $0 < \rho < 1$.
$\Delta\Lambda_{ij}(t)$	The amount of pheromone deposited by the ants.
η_{il}	The heuristic value (e.g., the inverse of distance).
$\theta_i(t + \Delta t)$	The new direction of particle *i* at time $t + \Delta t$ (rad).
$\xi_i(t)$	A random variable with uniform distribution in the interval [−1,1].
ω	Weight.
π	The policy.
π^*	The optimal policy.
π_i	The policy of robot (agent) *i*, mapping states to actions.
π_E	The expert policy that generates trajectories τ from the expert.
$\pi_\theta(s_t\|a_t)$	The policy function, parameterized by θ, gives the probability of taking action a_t at state s_t.
$\Phi(\acute{s}), \Phi(s)$	The potential functions that assign a value to the current and subsequent states, respectively.
$\emptyset(S)$	The feature vector that describes the state S.

LATIN LETTERS

A	Set of actions.
$\hat{A}_t$	The advantage estimator at time t.
a_{align}	The acceleration due to alignment.
a_{coh}	the acceleration due to cohesion.
ac_i	The resultant acceleration for Boid *i*.
a_{sep}	The acceleration due to separation.
c_1	Cognitive learning factor.
c_2	Social learning factor.
$d_{current}$	The normalized distance between the robot and the nest at time *t* with [0,3] (m).
Dc	This discriminator distinguishes between trajectories generated by the expert policy and those generated by the policy G.
D_{rg}	Goal Distance (Distance between robot and target)
d_{prev}	The normalized distance between the robot and the nest at time *t-1* with [0,3](m).
d_{robots}	Distance between two robots when they present around a big box(m).
dis_{reward}	Distance reward is when two robots are present around a big box within a certain distance (m).
E	Efficiency.
$FindThreshold$	The threshold to consider the robot is inside the nest.
$FindThreshold_{searching}$	The threshold value of the light sensor where the box is found. The normalized readings sensors are more than 0.85, meaning the robot reaches the box boundary. It is defined experimentally.
G	The Return.
$J(\theta)$	The objective function J concerning the policy parameters θ.
κ	The noise strength.
K	Data set of trajectories.
L_k	The length of the tour performed by ant *k*.
$LS_0^{(t)}, LS_7^{(t)}$	The normalized current readings of light sensors 0 and 7, respectively, at time *t*.
$LS_0^{(t-1)}, LS_7^{(t-1)}$	The previous normalized readings of light sensors 0 and 7, respectively, at time $t-1$.
$\mathcal{M}_i$	The set of neighbors of particle *i* within radius *R*.

Max_{Output}	The upper value in the output range of ∅(s).
Min_{Output}	The lower value in the output range of ∅(s).
Max_{Value}	The upper value is in the raw range of states.
Min_{Value}	The lower value in the raw range of states.
N	Number of retrieved items (boxes)
$\mathcal{N}_i^k$	The set of nodes that ant *k* can visit from node *i*.
$\|neighbors\|$	The number of Boids neighbors.
P	The Penalty.
x_j, x_i	The positions of particle *j* and its neighbor *i*.
$x_i(t)$	The position of particle *i* at time *t*.
$P_{i,t}^d$	The best fitness value particle has at the *t* moment.
p_g^d	The best fitness value among the swarm.
$Prob_{ij}^k(t)$	Probability of ant k moving from node i to node j.
$const$	A constant.
$Q(s,a)$	The Q-value for taking action a at state s.
$R(s_i, a_i)$	The reward function evaluates the performance of the agent *i* based on its actions a_i and the state of the environment s_i.
$r_t(\theta)$	The probability ratio of the new policy over the old policy.
$r_{box}(t)$	The additional reward is when the robot finds the box. The common approach to choose rewards values like1.1 are defined experimentally to fit the environment.
r_{nest}	The obtained reward is when the robot reaches the nest.
$R(s,a,\acute{s})$	The received reward for moving from state s to $\acute{s}$ when action a is chosen.
$R(S)$	This represents the reward associated with a particular state S.
R_{expert}	The output of the reward neural network for states from the expert.
$R_{training}$	The output of the reward neural network for collected states from training process.
S	Set of states.
V	Linear velocity.
$v_i(t)$	The velocity of particle *i* at time *t*.

$v_{i,t+1}^{d}$	The velocity of particle *i* at *t + 1* in d dimensions.
$v_{i,t}^{d}$	The velocity of particle *i* at *t (m/s)*.
v_j, v_i	The velocities of particle *j* and its neighbor *i*.
v_0	the constant speed of the particles.
$V(s_t)$	The value function at state s_t.
V_r	Avoiding speed for the right wheel.
V_l	Avoiding speed for the left wheel.
$v_{right-motor}$	Speeds of the right motor(rad/s).
$v_{left-motor}$	Speeds of the left motor(rad/s).
W	Turning velocity.
$w_{sep}, w_{align}, w_{coh}$	Weights for separation, alignment, and cohesion components.
$x_{i,t}^{d}$	The position of PSO's particle *i* at the *t* moment.
x_r, y_r	The Cartesian coordinates x, y of the robot
x_g, y_g	The Cartesian coordinates x, y of the goal

SUBSCRIPTS

$align$	Alignment behavior.
coh	Cohesion behavior.
sep	Separation behavior.
box	The items that the foraging swarm should collect.
nest	To indicate the target of the foraging swarm.
prev	The previous position of the robot.
Current	The current position of the robot.
t	Time.
Expert	The swarm or robot that has the solution to the problem.
Training	The swarm or robot trains to collect the experience.

1. INTRODUCTION

1.1. Overview of Swarm Robotics Systems

Swarm robotics systems (SRs) are a specialized field that focuses on developing models inspired by the decentralized decision-making processes of natural entities, such as insects, fish, herds, and others, to mimic their social behaviors [1],[2]. They rely on the dynamics of autonomous robots collaborating to perform complex tasks without centralized control architecture. So, SR models depend on a distributed approach to tackle the problems where each robot in the swarm contributes to generating collective behavior. They are seen as groups consisting of small and simple robots. These robots have limited capabilities and local perception of their environment. Each robot in the group is autonomous and makes its own decisions based on its local knowledge by interacting with other robots and the environment to complete designated tasks by the entire swarm. Based on the hardware structure, there are two categories: homogeneous, where robots are structurally similar, and heterogeneous, where differences in design are present. Homogeneous SRs are widely used due to their simplicity in design and ease of implementation. They consist of identical robots, making them easier to manage and requiring less complex programming and maintenance. Heterogeneous swarms are less common even though they offer diverse capabilities and task specialization, allowing them to handle more complex and varied tasks [3].

The principles underlying these systems have given rise to the field of swarm engineering, which is related to creating systems of simple robots that work together in known or unknown environments to achieve complex objectives through decentralized control, emergent collective behavior, and effective local interactions. It involves a systematic process that starts with defining the problem and analyzing requirements, followed by designing the individual robots and their behaviors. Then, simulation and modeling are used to test and refine the system, as well as prototyping and iterative improvements. Finally, the system is deployed, monitored, and maintained, with continuous data collection to optimize performance. The produced SRs have several advantageous characteristics, including scalability, flexibility, and adaptability. Scalability ensures that changes in the SR occur without significant changes to the system's functionality. Flexibility allows the SR to adapt to a wide range of tasks because each robot operates based on simple local rules and interactions, and the collective behavior can shift dynamically to meet new objectives. Adaptability is a key strength, as SRs can modify their strategies in response to environmental or task requirements changes. This allows them to remain effective in unpredictable scenarios, maintaining their performance and reliability [4].

1.2. Mathematical models of Swarm Robotics systems

Developing mathematical models of SRs has been inspired by observing the natural systems and replicating them with artificial ones with decentralized structures to represent the interaction among robots and the environment as it is called 'collective behavior.' Understanding and modeling this collective behavior can be approached from microscopic and macroscopic perspectives. The microscopic perspective focuses on the behavior and interactions of individual robots within the swarm, where each robot operates based on simple, predefined rules that select its actions in response to its immediate changes in the environment, like the angle of rotation, linear speeds, and others. In contrast, the macroscopic perspective focuses on the behavior of the entire swarm and the emergent properties that arise from local interactions like flocking, foraging aggregation, and others [5].

Mathematical models often build upon each other, enhancing performance and adaptability. The foundational Boids model by Craig Reynolds (1986) introduced simple rules for decentralized control and emergent behavior [6]. This concept influenced Particle Swarm Optimization (PSO) by James Kennedy and Russell Eberhart (1995), an optimization algorithm inspired by social behaviors [7]. Similarly, Ant Colony Optimization (ACO) by Marco Dorigo (1992) used pheromone trails to solve combinatorial problems, demonstrating the power of indirect communication [8]. Self-propelled particle (SPP) models by Tamás Vicsek (1995) further explored local interaction-based collective motion [9]. Meanwhile, behavior-based models by Rodney Brooks (1986) emphasize simple, reactive behaviors [10]. Multi-agent systems (MAS) provided a framework for modeling interactions and coordination among autonomous agents [11], where reinforcement learning (RL) brought adaptability and learning through trial and error to generate the collective behavior of SRs modeled by MAS [12],[13],[14]. Each model has evolved by incorporating insights from its predecessors, leading to increasingly sophisticated swarm control and optimization approaches.

1.2.1. Boids Model

This model simulates flocking behavior using three simple rules applied to each robot (boid *i*). The overall movement of each boid is determined by combining these rules. The rules are separation (avoiding crowding $\boldsymbol{a_{sep}}$) as in equation (1.1), alignment $\boldsymbol{a_{align}}$ (steering towards the average direction of neighbors), equation (1.2), and cohesion $\boldsymbol{a_{coh}}$ (moving towards the average position of neighbors), equation (1.3). In total, the resultant acceleration for each boid i $\boldsymbol{ac_i}$ is computed as in equation (1.4) to update the position $\boldsymbol{x}$ and velocity $\boldsymbol{v}$ as in equations (1.5) and (1.6) "In the following equations, vector quantities are represented using boldface notation , while scalar quantities are represented using regular italic notation. This convention is used consistently throughout the thesis."

$$\boldsymbol{a_{sep}} = -\sum_{j \in neighbors} \frac{\boldsymbol{x_j} - \boldsymbol{x_i}}{|\boldsymbol{x_j} - \boldsymbol{x_i}|^2} \tag{1.1}$$

$$\boldsymbol{a_{align}} = \frac{1}{|neighbors|} \sum_{j \in neighbors} \boldsymbol{v_j} - \boldsymbol{v_i} \tag{1.2}$$

$$\boldsymbol{a_{coh}} = \frac{1}{|neighbors|} \sum_{j \in neighbors} \boldsymbol{x_j} - \boldsymbol{x_i} \tag{1.3}$$

$$\boldsymbol{ac_i} = w_{sep}\boldsymbol{a_{sep}} + w_{align}\boldsymbol{a_{align}} + w_{coh}\boldsymbol{a_{coh}} \tag{1.4}$$

$$\boldsymbol{v_i}(t+1) = \boldsymbol{v_i}(t) + \boldsymbol{ac_i} \tag{1.5}$$

$$\boldsymbol{x_i}(t+1) = \boldsymbol{x_i}(t) + \boldsymbol{v_i}(t+1) \tag{1.6}$$

1.2.2. ACO model

Ants use pheromones ʎ to communicate and coordinate activities like foraging, particularly finding the shortest paths from their nest to a food source and back. ACO is primarily used to solve combinatorial optimization problems such as the Traveling Salesman Problem (TSP) and vehicle routing. Pheromone evaporation decreases the pheromone level to avoid unlimited accumulation, as in equation (1.7).

$$ʎ_{ij}(t+1) = (1-\rho)ʎ_{ij}(t) + \Delta ʎ_{ij}(t) \tag{1.7}$$

Each ant deposits pheromones on the paths it uses. The pheromone deposit $\Delta ʎ_{ij}$ is calculated based on the quality of the solutions found by the ants. That is demonstrated in both of equations (1.8) which accumulates pheromone contributions from all m ants that have traveled on edge (i,j), and (1.9) defines how much pheromone each ant deposits on an edge. The amount depends on L_k (tour length of ant k).

$$\Delta ʎ_{ij}(t) = \sum_{k=1}^{m} \Delta ʎ_{ij}^{k} \tag{1.8}$$

$$\Delta ʎ_{ij}^{k} = \begin{cases} \dfrac{const}{L_k} & if\ ant\ k\ uses\ edge(i,j)\ in\ its\ tour \\ 0 & otherwise \end{cases} \tag{1.9}$$

The optimal const value in ACO depends on the problem scale; a common choice is const ≈ L_{best}, with experimental tuning recommended for best results.

An ant k at node i chooses the next node to move to with a probability $Prob_{ij}^{k}(t)$ based on the pheromone level and the heuristic value, equation (1.10).

$$Prob_{ij}^{k}(t) = \frac{[\Lambda_{ij}(t)]^{\alpha}[\eta_{ij}]^{\beta}}{\sum_{l\epsilon\mathcal{N}_i^k}[\Lambda_{il}(t)]^{\alpha}[\eta_{il}]^{\beta}} \tag{1.10}$$

1.2.3. SPP Model

In the SPP model, each particle moves at a constant speed but continuously adjusts its direction based on the average direction of its neighbors within a certain interaction radius. This model is beneficial for understanding how simple local interactions can lead to complex global behaviors such as flocking, swarming, and schooling. Each agent *i* updates its new position $\boldsymbol{x_i}(\boldsymbol{t} + \Delta \boldsymbol{t})$ based on its current velocity $\boldsymbol{v_i}(t)$, adhering to the equation (1.11):

$$\boldsymbol{x_i}(t + \Delta t) = \boldsymbol{x_i}(t) + \boldsymbol{v_i}(t)\Delta t \tag{1.11}$$

This ensures that each agent moves in the direction dictated by its velocity at each time step. The velocity of each agent is influenced by the average direction of its neighbors within a certain radius, with added randomness to simulate real-world perturbations. The new direction $\theta_i(t + \Delta t)$ for each agent is determined by equation (1.12):

$$\theta_i(t + \Delta t) = \arg\left(\sum_{j\in\mathcal{M}_i} \boldsymbol{v_j}(t)\right) + \kappa\xi_i(t) \tag{1.12}$$

After determining the new direction, the velocity vector is normalized to maintain a constant speed v_0 as in equation (1.13).

$$\boldsymbol{v_i}(t + \Delta t) = v_0 \begin{pmatrix} cos(\theta_i(t+1)) \\ sin(\theta_i(t+1)) \end{pmatrix} \tag{1.13}$$

1.2.4. Behavior-Based Models

These models are inspired by biological systems, where complex behaviors emerge from the interaction of simpler behavioral modules. Behaviors are essential building blocks of the system, each representing a specific action or response to an event. For example, behaviors can include avoiding obstacles, following a path, or seeking a goal. Behaviors are typically reactive, meaning they respond directly to sensor inputs without the need for complex computation. Then, behavior arbitration is defined as a mechanism for resolving conflicts between competing behaviors, ensuring that the most appropriate behavior is executed at any given time. Complex actions and responses emerge from the interaction and combination of simpler behaviors.

So, each robot has a set of behaviors, $B = \{b_1, b_2, b_3 \ldots\ldots, b_n\}$ where each behavior b_i is a function that maps states S, which here represented as sensor inputs to actions A represented as actuator outputs as $b_i : S \rightarrow A$.

Each behavior b_i can be activated based on specific conditions or triggers. Let C_i represent the condition for activating behavior b_i. $C_i : S \rightarrow \{0,1\}$, Where $C_i(S) = 1$ if the condition for behavior b_i is met, given the sensor input S, and 0 otherwise. When multiple behaviors are triggered simultaneously, an arbitration mechanism decides which behavior to execute. This can be done

using fixed priority, dynamic priority, or a combination of behaviors through weighted sums, as in equation (1.14).

$$A = \sum_i w_i b_i(s) \tag{1.14}$$

The final action is a weighted sum of the activated behaviors, ensuring that the contributions of all active behaviors are normalized and combined to produce the final action, equation (1.15).

$$A = \frac{\sum_i \omega_i b_i(s)}{\sum_i \omega_i} \tag{1.15}$$

1.2.5. PSO Model

It is a computational method that optimizes a problem by iteratively improving a candidate solution about a given quality measure. It mimics the social behavior of birds flocking or fish schooling. The collection of particles (robots) moves through the solution space (environment), adjusting their positions based on their updated velocities, equation (1.16). The particles update their velocities by considering three key factors: their personal best position, the best-known position of the entire swarm, and a current position. This update process is mathematically represented in Equation (1.17). Figure 1.1 visually depicts these vector components, illustrating how each contributes to the overall velocity adjustment, guiding the particles toward optimal solutions."

$$\boldsymbol{v_{i,t+1}^d} = \omega * \boldsymbol{v_{i,t}^d} + c_1 * rand_i * \left(P_{i,t}^d - \boldsymbol{x_{i,t}^d}\right) + c_2 * rand_i * \left(p_g^d - \boldsymbol{x_{i,t}^d}\right) \tag{1.16}$$

$$\boldsymbol{x_{i,t+1}^d} = \boldsymbol{x_{i,t}^d} + \boldsymbol{v_{i,t+1}^d} \tag{1.17}$$

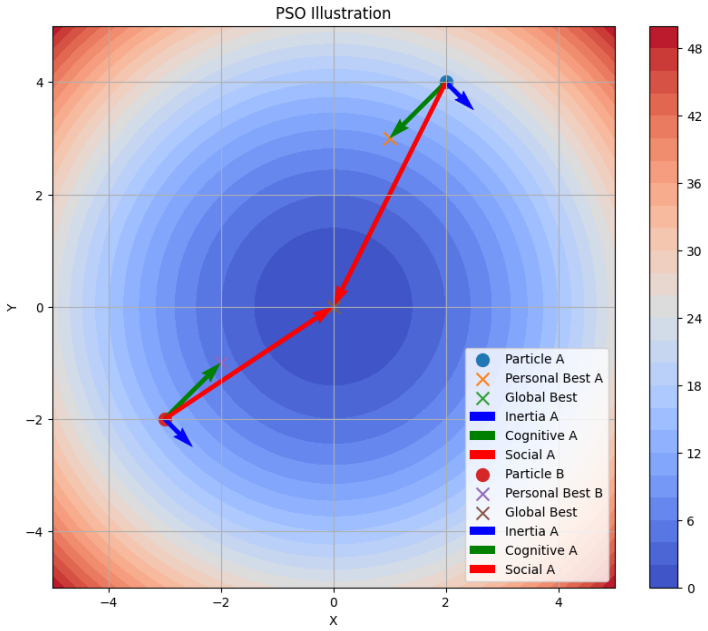

Figure 1.1. PSO illustration – Interia vector is $\omega * v_{i,t}^d$, and Personal best vector is $c_1 * rand_i * \left(P_{i,t}^d - x_{i,t}^d\right)$. Then, the social vector is $c_2 * rand_i * \left(p_g^d - x_{i,t}^d\right)$.

1.2.6. MAS Model

Multi-agent systems provide a robust framework for modeling the behavior and interactions of multiple autonomous robots. The mathematical model of MAS includes representing agents (robots), their states, actions, policies, interactions, and rewards. By optimizing individual and collective policies by reinforcement learning (RL), MAS can achieve complex tasks through local interactions and cooperation. Each robot i in the system is represented by its state s_i, action a_i, and policy π_i, where $s_i(t) \in S_i$, $a_i(t) \in A_i$, $\pi_i: s_i \rightarrow a_i$.

Reward and objective function: $R(s_i, a_i) = r_i\left(s_{i,t}, a_{i,t}, s_{i,t+1}\right)$.

Policy optimization: Robots aim to optimize their policies to maximize their expected cumulative reward, as in equation (1.18). This can be done using various reinforcement learning approaches.

$$\pi_i^* = \arg\max E\left[\sum_{t=0}^{\infty} \gamma^t R_i(s_t, a_t)\right] \tag{1.18}$$

While approaches like Boids, ACO, and behavior-based models have significantly contributed to the field of SRs, they are less popular today compared to RL and PSO. The Boids model primarily focuses on visual simulations and is less suited for solving optimization problems or dynamic task allocation. ACO is effective for solving combinatorial optimization problems like routing and scheduling. Despite its strengths, ACO tends to be computationally intensive and has slower convergence than PSO, making it more specialized for static optimization tasks than dynamic ones. Behavior-based models are simple and effective for implementing reactive behaviors and local interactions. Nevertheless, they lack flexibility and adaptability, struggling with complex and dynamic tasks that require learning and optimization over time. SPP model is suitable for studying collective motion and emergent behaviors, but it remains primarily theoretical and less practical for real-world applications. In contrast, Reinforcement Learning (RL) is highly adaptable and can learn and adjust to complex and dynamic environments, continuously improving performance through trial and error. Its versatility makes it suitable for various applications, including navigation, task allocation, and coordination.

1.3. Reinforcement learning

RL is a subfield of machine learning concerned with training agents through interacting with their environments rather than using labeled data as in supervised learning. This involves a sequential decision-making process where the chosen actions are refined over time based on a feedback signal from the environment in the form of a reward. The agent seeks to take action in an environment that maximizes cumulative reward. So, the agent's primary goal is to learn a policy that maximizes the cumulative reward over time. In RL, an agent interacts with its environment by taking actions based on a policy, a strategy, which is a rule that maps states of the environment to actions. The environment responds to these actions and provides feedback through rewards and new states. The agent's objective is to learn a policy that maximizes the long-term sum of rewards

[16]. The scope of RL extends across various domains and applications, driven by its ability to handle complex decision-making problems. Here are the key areas within the scope of RL:

1.3.1. Markov Decision Processes (MDPs)

MDPs are fundamental to the RL approach. An MDP provides a mathematical framework for modeling sequential decision-making. Formally, an MDP is defined by the tuple of (S, A, P, R, γ). State space S represents the set of states of environment that the agent encounters. Actions space A is the set of actions the agent can perform. P is the function of the probabilities that describe the environment's dynamic. R is the received reward for each selected action that moves the agent to the next state under probability P. The discount factor γ is usually between 0 and 1 and controls the importance of received rewards over time. So the problem of RL is formed as MDP to determine the policy π that chooses the action a for each state s that maximizes the cumulative received rewards, often termed as Return G as in equation (1.19). The Q- value represents the expected return of taking action a in state s, equation (1.20):

$$G = E\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)\right] \tag{1.19}$$

$$Q(s, a) = E[G_t | s_t = s, a_t = a] \tag{1.20}$$

Finding the optimal policy π^* involves maximizing G. Once the Q-values have been learned, the optimal policy π^*can be derived from them. The optimal policy selects the action with the highest Q-value for each state, equation (1.21).

$$\pi^*(s) = \underset{a}{argmax}\, Q(s, a) \tag{1.21}$$

1.3.2. Exploration vs. Exploitation

Learning the optimal policy requires maximizing G. So, the agent always seeks to select the actions that maximize the received rewards. In many situations, the agent may encounter low rewards in current states, but these states lead to other states with high rewards. So, it is better to give a probability ε to explore the environment more than exploit it to gain high rewards. This problem is termed an Exploration-Explotation dilemma. The ε-greedy algorithm is selected in all models in this dissertation, which is illustrated in algorithm 1 [16]:

Algorithm 1: $\varepsilon - Greedy$ pseudocode.

```
Initialize Q(s, a) arbitrarily for all state − action pairs
Set ε
while not terminal state: Generate a random number between 0 and 1
if random number < ε:
# Exploration: choose a random action
a = random action from set of possible actions
else:
# Exploitation: choose the action with the highest Q value
a = action with highest Q(s, a)
```

1.3.3. Model-Free and Model-Based Approaches

Models-based RL uses a pre-defined environment model to predict future states and rewards, enabling efficient learning through planning and simulation. Examples include Value Iteration, Policy Iteration, and Model-Based Dyna-Q [16]. This approach is sample-efficient but computationally intensive. If the environment is unknown, there is no way to create a model to determine P. In this case, model-free RL methods, such as Q-learning [17] and SARSA [18], do not require a model of the environment because they learn optimal policies directly from interactions without explicitly estimating the transition probabilities P. They allow the learning process to be simplified, computational requirements reduced, and the learning process adapted more quickly to complex and dynamic environments. RL methods have become more efficient and have less computations by incorporating them with deep learning techniques [19]. Moreover, deep learning introduces new methods for RL, which is termed deep RL. Instead of depending on Q-values like $Q(s,a)$ to obtain the optimal policy, which is called value-based methods, neural networks have been employed to learn the optimal policy directly, such as Soft Actor-Critic (SAC) [20], Trust Region Policy Optimization (TRPO) [21], Deep Deterministic Policy Gradient (DDPG) [22]. For instance, DDPG is an actor-critic algorithm designed for continuous action spaces. It learns an actor network to select actions and a critic network to estimate the value of the chosen actions. Proximal Policy Optimization (PPO) is also based on actor-critic and policy gradient methods that optimize the objective function by clipping the policy update. It is designed to be computationally efficient and stable [23]. Figure 2 is an example of classification RL algorithms.

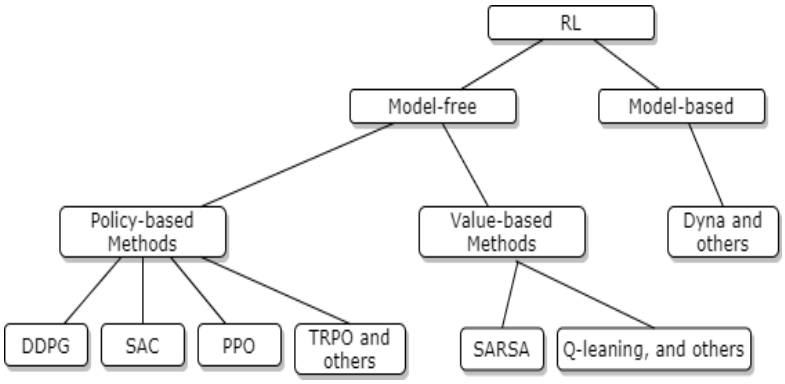

Figure 1.2 RL algorithms diagram.

1.3.4. Why PPO for robots domain

To select the RL algorithm for robots, particularly in the context of SRs, several factors need to be considered, including the complexity of the environment, the need for real-time decision-making, computational resources, and the specific objectives of the task. There are many algorithms like DQN, DDPG, TRPO, PPO, and others. Firstly PPO is a policy gradient method as in equations (1.22), This equation represents the gradient ascent update rule for optimizing the policy π_θ. The gradient is computed using log-probabilities, making it efficient for policy updates,

and $clip(r_t(\theta), 1-\epsilon, 1+\epsilon)$ restricts the update to be within a small range [1−ϵ,1+ϵ]. As a results, that gives an advantage when it compares to other like TRPO for computational cost. While $\hat{A}_t$ reducing variance and stabilizing training which given in (1.23).

Ensures the policy is updated efficiently, focusing only on beneficial changes.

$$\nabla_\theta J(\theta) = E_t\left[\nabla_\theta log\pi_\theta(s_t|a_t)\hat{A}_t\right] \tag{1.22}$$

$$\hat{A}_t = Q(s_t, a_t) - V(s_t) \tag{1.23}$$

It optimizes the policy directly by using a clipped surrogate objective function, considered a powerful tool for ensuring stability. This approach ensures stable and reliable policy updates, making PPO highly effective in complex environments. PPO is suitable for continuous and discrete action spaces, which benefits the diverse tasks encountered in SRs. The clipped surrogate objective $L^{CLIP}(\theta)$ can be mathematically represented as equations (1.24),(1.25):

$$L^{CLIP}(\theta) = E_t\left[\min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\right] \tag{1.24}$$

$$r_t(\theta) = \frac{\pi_\theta(s_t|a_t)}{\pi_{\theta_{old}}(s_t|a_t)} \tag{1.25}$$

This ratio compares the probability of taking action a_t under the **new policy** π_θ versus the **old policy** $\pi_{\theta_{old}}$. If $r_t(\theta) >1$, the new policy assigns a **higher probability** to action a_t than before. If $r_t(\theta) <1$, the new policy assigns a **lower probability** to action a_t .

While DQN is more straightforward and sample-efficient, it is limited to discrete action spaces, which may not suffice for more complex robotic tasks. DDPG provides a deterministic policy suited for continuous actions but suffers from lower sample efficiency and higher sensitivity to hyperparameters, which were tested in our study. TRPO, like PPO, offers high stability but at the cost of greater computational complexity and implementation difficulty. Several studies have demonstrated the effectiveness of PPO in various robotics contexts [24], [25], [26], [27], including SRs, where PPO shows its ability to manage continuous action spaces and provide stable performance that has been highlighted in research [28], [29], [30].

1.4. Outline of the Thesis

In Chapter 2, a new comparative study investigates two prominent methodologies, PSO and DRL, to analyze the performance of a swarm of mobile robots through extensive experimentation. The objective is to produce a navigation of collective behavior through unknown environments, highlighting the strengths and weaknesses of each approach. Chapter 3 explores two enhancement techniques for DRL. Firstly, it proposed a method for achieving collective navigation behavior in a swarm of robots using DRL. It includes an enhanced PPO by utilizing curriculum learning. A comparative analysis between the enhanced PPO, the original PPO, and the DDPG algorithm highlights the strengths of the proposed enhancment. A novel hybrid approach with a modular design (Behavior-Based Model) combining DRL and PSO is introduced to address dynamic

foraging tasks characterized by non-static environments and objectives. Chapter 4 examines DRL's reward formulation, focusing on using sparse and shaping rewards to optimize learning, then developing a deep inverse RL model to uncover the reward structures that guide SR in achieving tasks through demonstrations. The proposed model is tested under different collective behaviors according to the required objectives like searching and navigation.

2. AUTOMATIC DESIGN METHODS

Automatic design methods refer to a set of computational techniques and algorithms that autonomously generate efficient and effective collective behaviors of SRs without requiring explicit human intervention for each step. These methods leverage various forms of artificial intelligence, like machine learning and optimization algorithms, to automatically create, evaluate, and refine robot behaviors based on required objectives.

2.1. A Survey on Automatic Design Methods

Researchers have investigated various design strategies and control mechanisms to generate the collective behavior of SRs. The approaches are generally classified into two fields: behavior-based design methods and automatic design methods [31].

Behavior-based design methods are based on modularization of robot behaviors into distinct modules. Each module is produced by many actions based on sensor readings, like avoiding obstacles, grabbing boxes, and others. Thus, the collective behavior is generated by interactions and coordination among modules. These methods are considered simpler and easier to interpret than automatic design approaches. However, coordinating and tunning many modules is challenging, especially for complex tasks, because of potential conflicts between modules and managing the interactions and dependencies between modules. Each module must work harmoniously with others to achieve the desired collective behavior. At the same time, Automatic design methods like RL, PSO, genetic algorithms (GA), ACO, and others are employed to generate the collective behavior of SRs. They utilize machine learning approaches and optimization algorithms to tune the parameters and policies that govern the behavior of individual robots within a swarm [32], [33]. For example, evolutionary algorithms like genetic algorithms and PSO are traditionally favored for fine-tuning robot behaviors by defined fitness functions, while RL is often preferred for applications requiring continuous learning, adaptability, complex decision-making, and dynamic exploration-exploitation balance [34].

2.1.1. PSO-Driven Solutions in SRs

Using the PSO model to enhance the coordination and performance of SRs, particularly for tasks like target search and navigation. Its strength lies in simplicity, effectiveness, and low computational cost. , making it a practical choice for swarm-based applications.

Many contributions have been carried out to enhance PSO's applicability in SR by adapting various PSO versions to fit diverse contexts with mathematical refinements [34]. An asynchronous

PSO (APSO) version refines the basic model by updating the velocity and position of each particle immediately after evaluating an individual particle's fitness without waiting for the entire swarm's fitness evaluation. It ensures the adaptivity of the system to the limited knowledge of the environment [36]. An Adapted PSO model is introduced as the modified version that is employed in physical SRs for localization sources. This modification linked the parameters to the velocity and acceleration of each robot. The physical properties of each robot, like the desired maximum velocity and acceleration, and relating these to the inertia weight and the cognitive and social coefficients via a state model [37]. Many studies incorporate artificial intelligence approaches like fuzzy logic or neural networks to fine-tune and optimize PSO in swarm robotics. There are many strategies to adapt PSO in SRs [38]. However, to overcome key challenges of PSO in swarm robotics, such as path planning, collision avoidance, maintaining coordination as the swarm size grows, and preventing congestion, it is essential to establish robust communication protocols.

2.1.2. RL-Driven Solutions in SRs

Many improvements have been introduced to RL for SR applications [39]. For example, various methods were compared, like Deep Q-Network (DQN), N-Step-Q Network (NSQ), and Double DQN. All these methods were used to generate a navigation collective behavior of a proposed SR. Then, based on them a combined version named Double N-step Q-Network (DNQ) was derived. It showed a superior performance in convergence to the optimal policy [40]. Another architecture was introduced with a hierarchical RL structure, one for the controller responsible for executing the tasks and another for choosing the controller itself [41]. Defining the reward plays a vital role in obtaining optimal swarm behavior. Sparse rewards with different values were used to navigate an obstacle-environment. The aim was to clarify the importance of selected reward values in swarm behavior. It showed that selecting high penalties makes the swarm focus on avoiding obstacles rather than directed toward the goal [42].

These contributions and others help us understand how to adapt RL in SR. They explored many DRL architectures and reward formulations to optimize the swarm performance. However, many challenges arose, especially in complex environments like random initialization of neural networks, which led to extra time for the DRL algorithms to converge to the solution, Crucial formulating and tuning of the reward function, exploration-exploitation dilemma, and others [43].

Too many studies have deployed both PSO and RL in SR applications due to their effectiveness; there is a relatively scarce comparative analysis between these two prominent methods. PSO and Q-learning were compared for the multi-robot systems to avoid obstacles, this comparison was limited to measuring the time of convergence where the findings showed that Q-learning with continuous states is faster than PSO [44].

Another comparative study explored three automatic methods: Bat Algorithm (BA), Grey Wolf Optimizer (GWO), and PSO; PSO outperforms BA, and BA surpasses GWO [45]. There is a gap in comparing the two main approaches, PSO and RL, in SRs. This gap is important because it provides deeper insight into the best scenarios for each approach based on their theoretical

foundations. By comparing these methods, we can identify their strengths and weaknesses, helping to determine the most effective algorithm for specific tasks.

Swarm intelligence algorithms have been extensively studied and developed for various applications, including optimization, robotics, and distributed computing. Due to their decentralized and adaptive nature, these algorithms are often modified or extended to enhance performance in specific problem domains. According to the existing research, it is noticeable that enhancements generally fall into three major categories: parameter modifications, algorithmic combinations, and structural modifications. The effectiveness of each enhancement strategy depends on the nature of the problem, the search space characteristics, environmental constraints, and others.

1. Modifications Through Parameter Tuning

One of the most common approaches to enhancing swarm intelligence algorithms is through parameter optimization, where RL techniques are used to dynamically adjust hyperparameters in swarm algorithms. For instance, iSOMA-PPO (Self-Organizing Migrating Algorithm + PPO) utilizes PPO to adjust the step size and migration patterns of the iSOMA algorithm, improving its performance in complex optimization tasks [46]. Similarly, RL-LSOP (RL for Large-Scale Optimization Problems) applies RL to fine-tune PSO parameters in high-dimensional search spaces ($D \geq 500$) [47]. These approaches allow swarm algorithms to become more adaptive and self-tuning, improving their convergence speed and efficiency without requiring manual parameter adjustments. However, these methods are primarily focused on global optimization tasks rather than real-time multi-agent coordination, which is critical in SRs and other decentralized systems.

2. Modifications by Combining Multiple Algorithms

Another major trend in swarm intelligence research involves hybridizing multiple algorithms to leverage their complementary strengths. This can be seen in PSO-GA (PSO + Genetic Algorithm), where GA introduces evolutionary operators (crossover & mutation) that introduce randomness and diversity, preventing premature convergence [48]. Similarly, ACO-PSO (Ant Colony Optimization + PSO) integrates pheromone-based path selection from ACO with PSO's velocity-based updates, enhancing performance in dynamic environments [49]. While these hybrid approaches show significant performance improvements, they often introduce higher computational complexity and may require domain-specific adaptation to be effectively deployed in real-world applications.

3. Structural Modifications

Structural modifications focus on how individual agents interact and adapt over time. One such approach is hierarchical swarm intelligence, where the swarm is divided into different levels of decision-making, allowing for more scalable and coordinated behaviors. For example, Hierarchical PSO (H-PSO) introduces leader-follower dynamics, where a subset of high-performing agents guides the overall swarm [50], another example is my proposed foraging system in chapter 3.

While numerous studies have explored modifications to swarm intelligence algorithms by integrating RL techniques, most of these works focus primarily on optimizing algorithmic parameters for efficiency rather than analyzing their impact on emergent collective behaviors in swarm robotics. Many hybrid methods, such as iSOMA-PPO, RL-tuned PSO, and LRPSO, aim to enhance the performance of PSO by adjusting its search strategies, but these studies do not examine how such modifications affect swarm-level interactions, adaptability, or decision-making in dynamic environments. A broad comparison across all swarm intelligence methods would be neither effective nor logical, as the effectiveness of these algorithms is highly dependent on the nature of the problem and the environment. In swarm robotics, collective behaviors such as foraging and path planning are the key performance indicators rather than just convergence speed or solution accuracy. Thus, a direct comparison between PSO and PPO is essential to understand how each method contributes to swarm-level coordination and adaptability.

This research is one of the few that directly evaluates how PPO and PSO influence the collective behavior of a swarm. Understanding their strengths and limitations in swarm robotic tasks is crucial for designing structured control architectures that leverage the best of both approaches. By focusing on collective behavior rather than algorithmic fine-tuning, this research fills a critical gap in the literature, demonstrating why a comparative analysis between PSO and PPO is not just relevant but necessary to advance SR research.

2.2. PSO vs. RL Methodologies in Swarm Navigation Behavior

This section conducts a comparative analysis of PSO and RL methodologies in the context of SRs. It primarily explores their effectiveness in generating navigational collective behavior. It assesses SR's flexibility and adaptivity in the context of these algorithms, providing valuable insights into the selection of control strategies for SR [51].

2.2.1. Defining the task and the environment

The 3D Webots robot simulator implements the environment of the swarm robots [52], where the E-puck mobile robot are chosen to create a homogenous swarm [53]. The Deepbots framework integrates abstract PPO with the simulator[54]. The system is designed to enable collective navigation behavior, guiding the robots to converge at a specified target position while avoiding collisions. This behavior is tested across various environments, as depicted in Figure 2.1. The testing areas have a square shape surrounded by four walls, with sizes of $1 \times 1\ m^2$, $1.3 \times 1.3\ m^2$, and 1.6×1.6. Available in three different layouts: no obstacles, one obstacle, and two obstacles. The E-puck robots are configured with a linear velocity range of $[0, 0.25]\ m/s$ and an angular velocity range of $[-3.14, 3.14]\ rad/s$, with their Infra-Red sensors having a range of $[0, 0.04]\ m$.

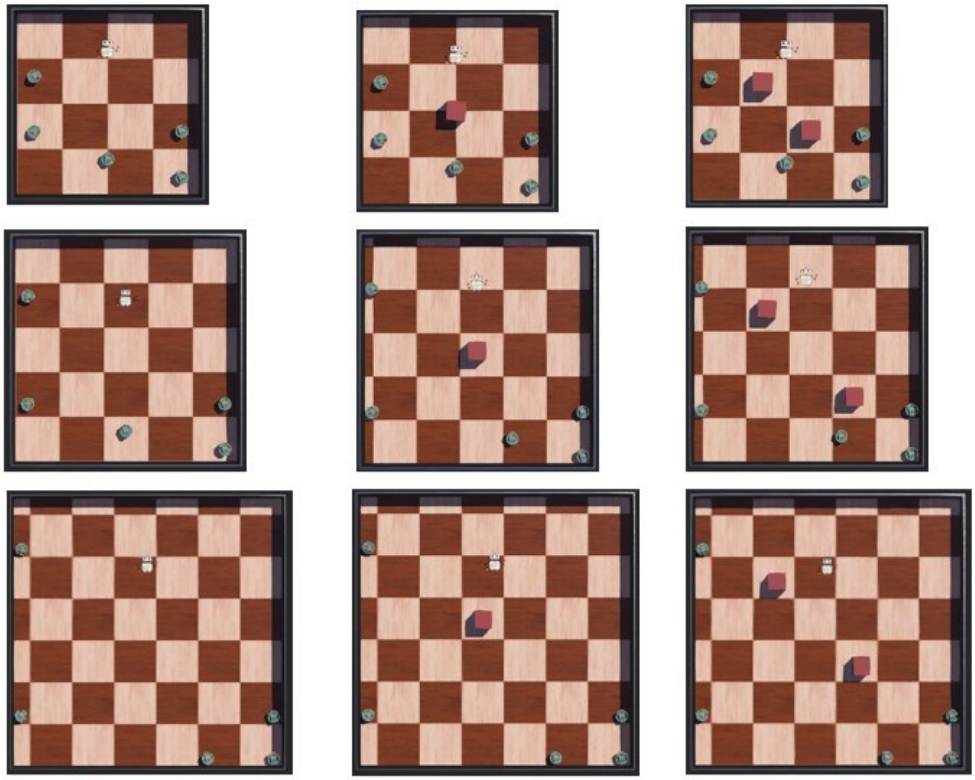

Figure 2.1. Environments for the navigation task.

2.2.2. PSO Methodology

The PSO model described in equations (1.16) and (1.17) is adapted for solving a navigation task, where each robot is treated as a particle. The particle update in the PSO model corresponds to the path planning in SR. The navigation criterion for the robots is their fitness value. This value is determined by the distance from each robot to the target. Robots exchange information about their fitness values and current positions with one another. Robots' optimal personal and global positions are selected based on the smallest fitness values. To direct a robot to its next position, the speeds of its left and right motors $v_{left-motor}$, $v_{right-motor}$ are adjusted by equations (2.1) and (2.2), the process is detailed in Algorithm 2.

$$v_{right-motor} = V + W + V_r \tag{2.2}$$

$$v_{left-motor} = V + W + V_l \tag{2.3}$$

Algorithm 2: PSO model for the navigational task.

Initialize the environment and robots with positions and velocities
Define fitness function ← DISTANCE between the robot and the target
repeat
for each robot, do
Evaluate robot fitness using the fitness function

if current robot fitness is better than the previous best robot fitness, then
Update the robot's best position to the current position
end if
GET all robot's fitness and positions
Update global best position according to best robot fitness
end for
for each robot, do
Define w = 0.7, c1 = 2, c2 = 2
Update robot velocity using the equation:
v = w×v+c1×r1× (robot best−current position) +c2×r2× (global best−current position)
Update robot position using the equation:
position = position + velocity
Calculate the linear velocity V ← Distance to the new position
Calculate the turning velocity W ← Angle to the new position
Calculate the avoiding speed for each wheel V_l, V_r
Apply left motor speed ← V +W + V_l,
Apply the right motor speed ← V +W + V_r
end for
until the robot reaches the target

2.2.3. DRL methodology

The MAS model described in section (1.2.6) is formulated as an MDP with the tuple (S, A, T, R, γ). The state space $S = [IR\ sensor's\ readings: LS0 \dots, LS7, D, \theta]$. Where D is the distance between the target and the robot, and θ is the angle between the robot and the target—action space $A = [velocities\ of\ the\ two\ motors]$. The transition function T describes the dynamics of the system. By using free models, there is no need to estimate it. Reward R: Shaping reward, as shown in Equations (2.4), (2.5), and (2.6), is applied to leverage the experience during the path by assessing the robot's current and previous positions at each time-step and employing a sparse method to punish robots when they collide with obstacles. The discount factor γ is a value that balances the importance of immediate rewards against future rewards.

$$Penalty = \begin{cases} -0.001 & There\ are\ obstacles\ detected\ by\ the\ IR\ sensors. \\ 0 & There\ are\ no\ obstacles. \end{cases} \quad (2.4)$$

$$Reward_{target} = \begin{cases} 0.1 & The\ robot\ reaches\ the\ target. \\ 0 & The\ robot\ does\ not\ reach\ the\ target. \end{cases} \quad (2.5)$$

$$Reward = Previous\ distance - Current\ distance + Penalty + Reward_{target} \quad (2.6)$$

The PPO Network, as in Figure 2.1, optimizes the policy by adjusting the weights to improve the expected return G while keeping the new policy not too far from the old policy. Both the actor and the critic networks use fully connected layers with dimensions (10×64×64×2 for the actor and 10×64×64×1 for the critic with ReLU activation function). The actor part proposes actions as two speeds based on the current state. The critic network evaluates the actions the actor takes by

computing the advantage, which indicates how much better an action is than the average. After computing the initial actions, a Gaussian process is applied to refine these actions or to add exploration noise. The hyperparameters of the PPO training process are presented in Table 2.1. All mentioned hyperparameters for PPO in are chosen based on empirical testing and [53].

Table 2.1: PPO hyperparameters.

Parameter	Value
Max training timesteps	1000000
Max timesteps per episode	1500
State space dimension	10
Action space dimension	2
Discount factor γ	0.99
PPO epsilon clip	0.2
PPO K epochs	80
Optimizer learning rate actor	0.0003
Optimizer learning rate critic	0.001

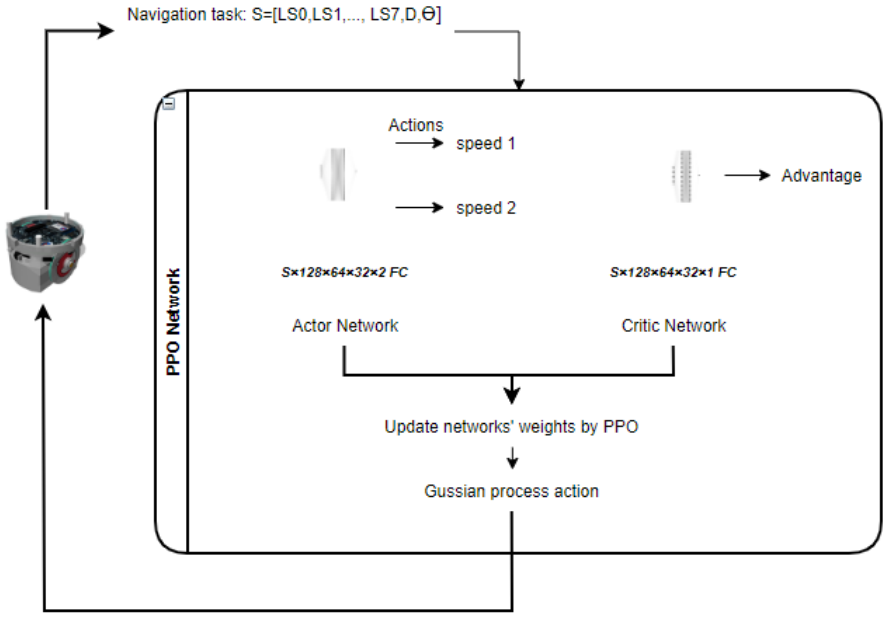

Figure 2.1. PPO architecture for the navigation task.

2.2.4. Results and discussion

A comparative analysis is carried out to assess the system's performance by defining three metrics:

- The effectiveness is measured by determining the overall time required to accomplish the task. Flexibility is evaluated by the ability to adapt to changes in environments. The generalization of the solution is tested in a new environment.

The comparison is performed under identical conditions using the defined performance metrics across multiple trials for validation. Assessing the efficiency by the time of the first and last robot in the swarm to complete the task for each environment. The time taken by the first robot to reach the target, denoted as $RLmin$ by using RL and $PSOmin$ by using PSO, reflects the fastest individual robot (Microscopic level – individual behaviors).

Conversely, the time recorded for the last robot to reach the target, indicated as $RLmax$ and $PSOmax$ represents the total completion time for the task (Macroscopic level – collective behavior).

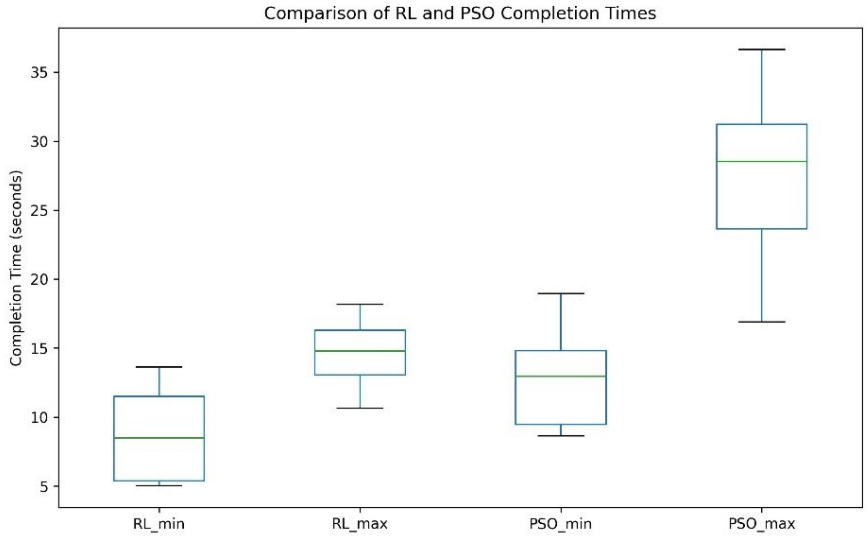

Figure 2.2. Efficiency of RL vs. PSO.

Analysis based on results in Figure 2.2 indicates that RL outperforms PSO. In detail, $RLmin$ recorded a median completion time of 8.48 seconds and a mean of 8.7 seconds, suggesting that RL facilitates faster individual robot performance compared to PSO, where $PSOmin$ achieved a median time of 12.96 seconds and a mean of 12.55 seconds.

Although PSO shows a narrower interquartile range (IQR), suggesting more consistent performance among the fastest robots, it highlights a uniformity in completion times close to the median, indicating similar performance levels across various environments.

For the coordination of robots, denoted as $RLmax$ versus $PSOmax$, RL shows a median completion time of 14.78 seconds with a mean of 14.64 seconds, along with a compact IQR that underscores a consistent performance among slower robots. In contrast, PSO has broader variability in completion times, as shown by its broader IQR, with a median of 28.54 seconds and a mean of 27.9 seconds. This variation is attributed to bottlenecks at the target, which delay the task completion.

Overall, RL demonstrates more uniform efficiency across individual and coordinated robot performance. In contrast, PSO displays greater variability in robot performance, suggesting

different strengths, like flexibility, that might not be evident from this box plot analysis. Furthermore, the paths followed by the robots, as indicated in Figures 2.3 and 2.4, show the flexibility of both algorithms in adapting to changes in the environment. PSO-guided robots follow convergent routes, exhibiting a collective behavior influenced by shared information as they navigate toward the target. Their paths are smoother, indicating efficient routing with minimal deviations, though looping near the target suggests adjustments within the swarm. Obstacles lead to longer, collaborative routes to the destination. RL, however, displays a range of trajectories that indicate a more autonomous, learning-based navigation strategy, with paths featuring sharp turns and complex maneuvers that demonstrate an adaptive learning process. Compared to PSO, RL shows less clustering of paths and more individualized responses to obstacle avoidance.

When comparing PSO with RL, the PSO's paths are marked by unified swarm behavior, possibly sacrificing individual efficiency for group cohesion. RL's ability to learn and adapt results in more strategically sound maneuvers than the more deterministic and collective patterns observed with PSO.

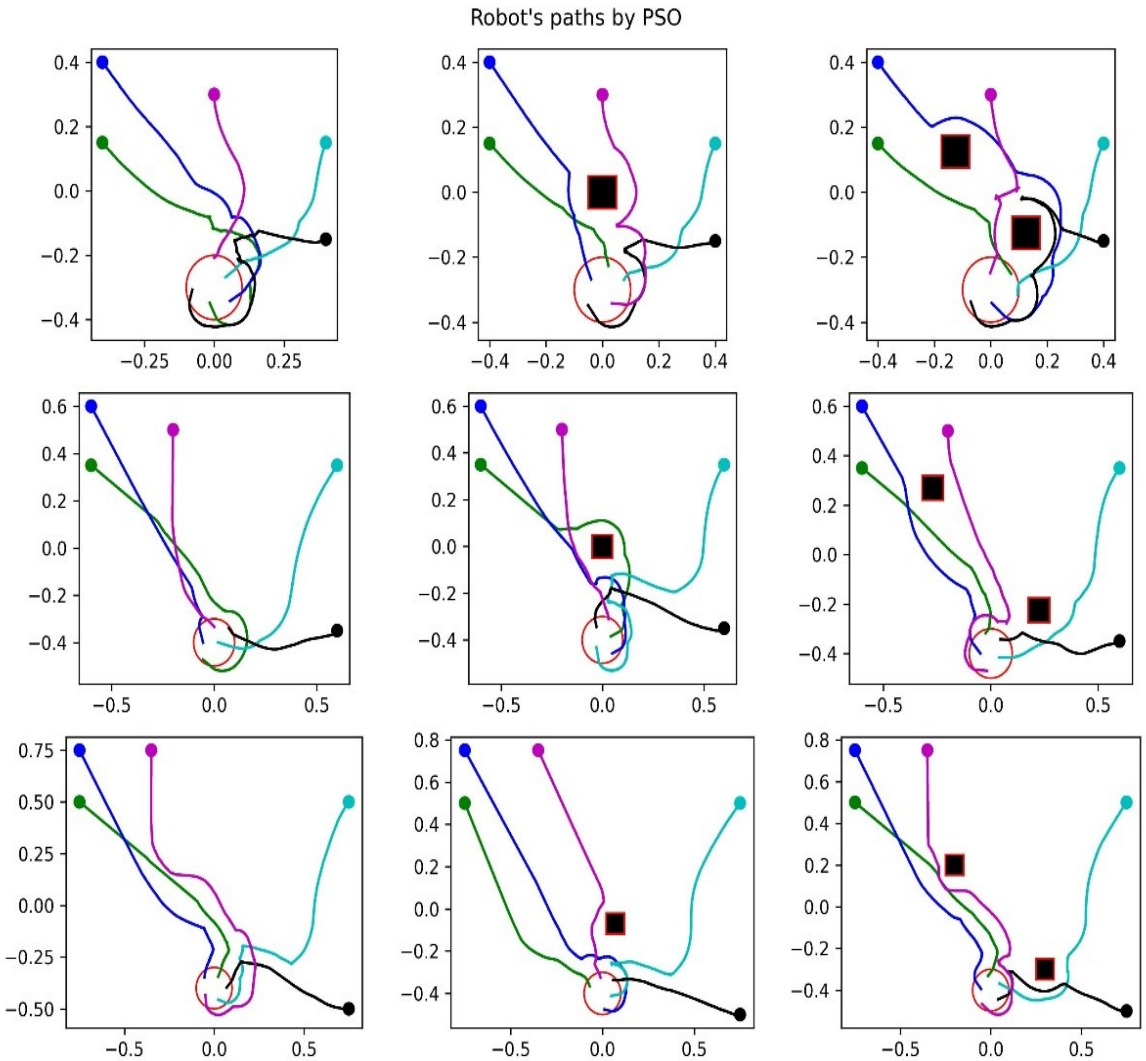

Figure 2.3. Flexibility of PSO.

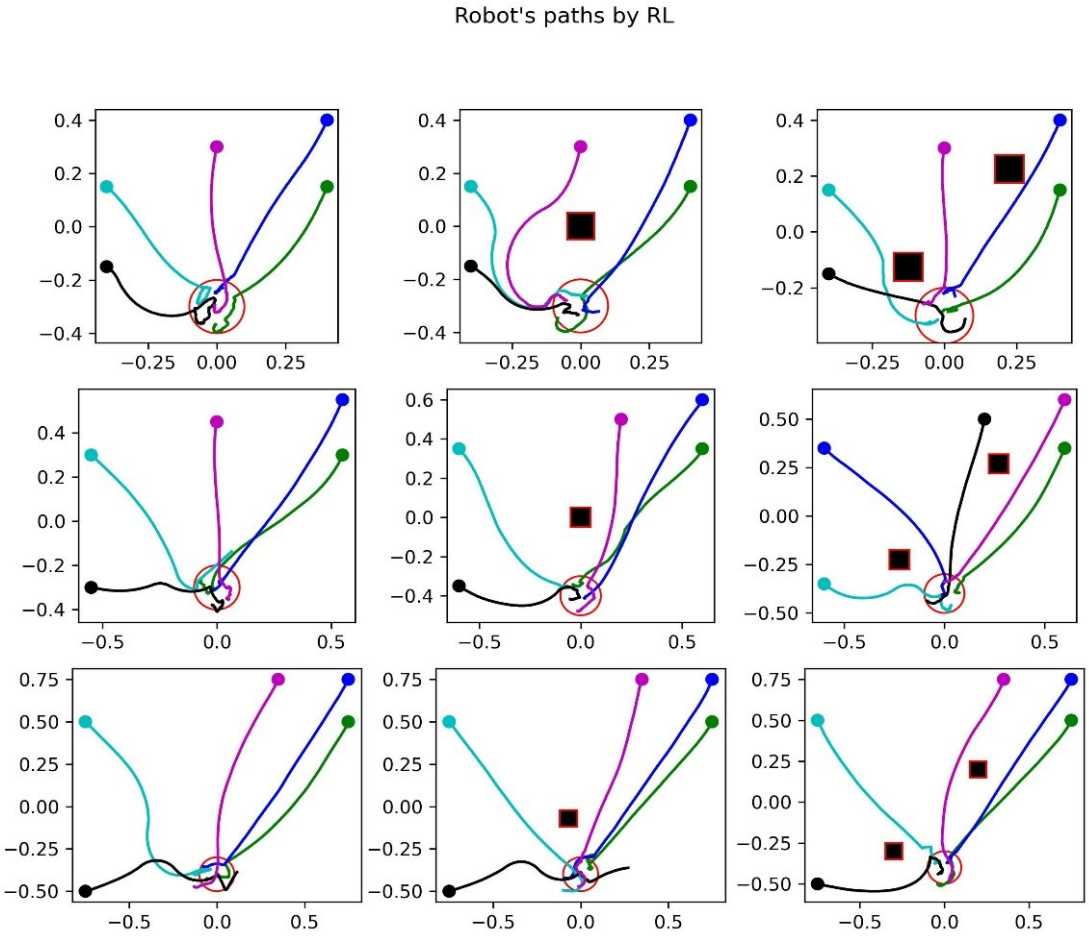

Figure 2.4. Flexibility of RL.

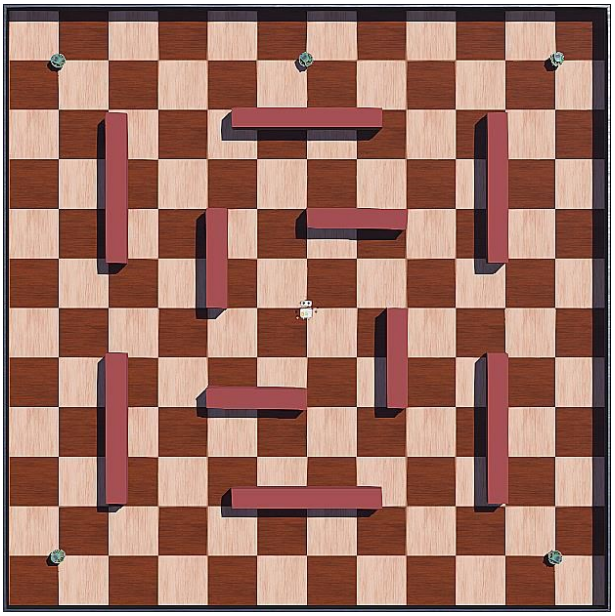

Figure 2.5. Environment for testing the generalization.

To assess the generalization of PSO and RL, the behavior patterns of robots were initially established in a 1.6×1.6 m² environment and subsequently evaluated in a larger 3×3 m² area, as depicted in Figure 2.5. RL encounters obstacles like limited prior experience and the tendency to overfit when faced with different conditions, such as new initial positions for the robots or additional obstacles. To mitigate these challenges, adjustments to the RL's actor-critic framework or the integration of progressive learning techniques may be necessary (studied in Chapter 3 in the section on curriculum learning). In contrast, PSO maintains a consistent performance level despite environmental changes. In tests conducted within this new, more complex setting, PSO–driven robots demonstrated the ability to navigate collaboratively toward a collective objective, as illustrated in Figure 2.6. However, RL-driven robots lacked collective swarm behavior and could not converge, as shown in Figure 2.7. These observations underscore the necessity for advanced strategies to enhance RL's adaptability in both complex and variable environments.

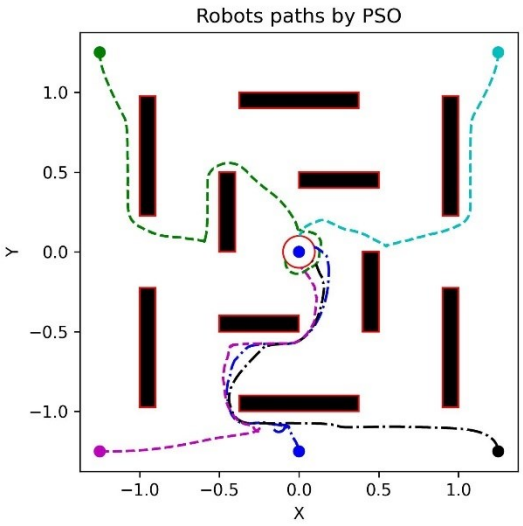

Figure 2.6. PSO generalization.

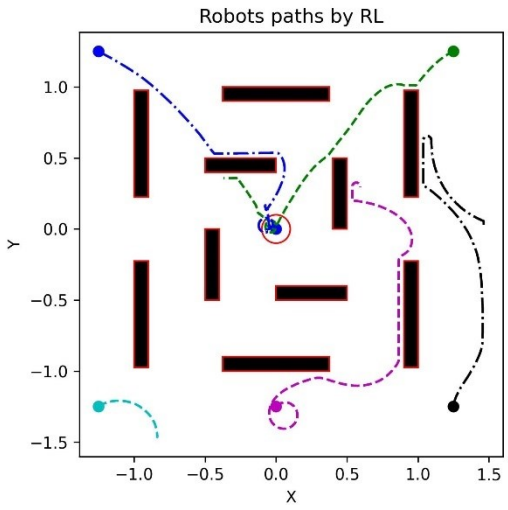

Figure 2.7. RL generalization.

2.3. Conclusion of comparative analysis

This study compares the effectiveness of PSO and RL algorithms in guiding a swarm of robots through an unknown environment. It highlights the strengths and weaknesses of each method. RL is faster and better at coordinating robots, but it works best in conditions similar to where it was trained. RL struggles to adapt to new environments without additional training or more complex training process with heavy structure. On the other hand, PSO may be slower than RL, but it performs consistently well across different environments. This makes PSO a reliable choice for tasks needing stability, especially in unpredictable settings. The slower speed of PSO does not greatly affect its ability to perform steadily. It is suggested to use RL, where quick reactions and close coordination are essential, mainly in familiar settings. PSO is recommended for tasks requiring reliability and consistent performance in new environmental challenges. The results of this study demonstrate fundamental differences between PSO and PPO in SRs, particularly in terms of adaptability, coordination, and generalization across different environments. Unlike previous research that integrates reinforcement learning into PSO for optimization purposes (e.g., iSOMA-PPO, RL-LSOP, and LRPSO), our findings reveal how these two approaches

independently influence swarm-level behaviors rather than simply enhancing computational efficiency. Moreover, our results indicate that swarm coordination differs significantly between PSO and PPO-based models. PPO agents tend to learn individualized strategies that do not always align with collective decision-making, leading to less structured swarm behavior compared to PSO-based swarms, which naturally converge toward a globally coordinated movement pattern. This observation is crucial for designing hybrid PSO-PPO frameworks, as it suggests that combining PPO's learning-based adaptability with PSO's swarm coordination could lead to more effective decentralized decision-making in multi-agent robotic systems.

These findings suggest that future work should focus on structured control architectures that integrate PPO's adaptive learning with PSO's decentralized search and coordination mechanisms. Rather than viewing PSO and PPO as competing methods, this study highlights their complementary strengths, providing a basis for designing hierarchical or modular control strategies for swarm robotic tasks such as foraging, path planning, and collective exploration.

3. ADVANCING DRL FOR SRS : INNOVATIVE ENHANCEMENT TECHNIQUES

In this chapter, we explore integrating RL approaches within SRs, a domain that significantly benefits from RL's adaptability and autonomous decision-making capabilities. SRs must adapt to dynamic environments by adjusting their behaviors through trial and error; this is autonomy, which is the essential concept in the SRs. The trial- error approach allows robots within the swarm to independently refine their strategies until they determine the most effective methods. So, RL is ideal for these systems, such as optimizing foraging behaviors, which are crucial for efficient environmental mapping and robust execution of missions. Recent studies have focused on deploying RL to emerge collective foraging behaviors. To address the persistent challenges in this field, including the complexity of dynamic environments, scalability of learned behaviors, communication constraints, balance between exploration and exploitation, and the limitations posed by energy and computational resources. A spectrum of strategic solutions to these challenges is presented in Table 3.1, providing an overview of current advancements and methodologies in enhancing DRL within SRs.

Table3.1. Challenges of deploying RL in foraging swarms.

Challenge Description	**Challenge Description**
Complexity of Dynamic Environments	Adapting to unpredictable changes, including moving targets and obstacles.
Scalability of Learning	Applying learned behaviors to swarms of varying sizes and compositions.
Communication Limitations	Coordinating actions without overwhelming the network control architecture.
Balancing Exploration and Exploitation	Efficiently discover new strategies and environments while utilizing learned behaviors to achieve optimal performance.
Energy and Computational Constraints	Managing energy consumption and computational demands for efficient operation.

For the "complexity of dynamic environments," creating macroscopic foraging behaviors while integrating fuzzy logic for fine-tuned obstacle navigation and avoidance is a solution introduced in [55]. This combination reduces the complexity of the RL problem space, leading to robust,

scalable foraging behaviors that hold up even in untrained scenarios. Another solution employed Multi-Agent Reinforcement Learning (MARL) to refine the foraging strategies of active particles, allowing them to locate and harvest food sources that appear randomly and efficiently [56]. This research showed that optimizing individual agents' behavior could enhance the entire swarm's collective foraging efficiency. Another innovative strategy involved using deep RL combined with CL to master navigation tasks progressively, thereby boosting learning efficiency and environmental adaptability [57].

Various techniques have been proposed regarding scalability issues. One research [58] focused on enhancing system performance without increasing the complexity of individual robots or intensive inter-robot communication. It suggested that simple, decentralized interactions could facilitate complex collective tasks. Another study introduced a self-organizing task distribution model based on a response threshold mechanism, which allows SR to efficiently divide complex foraging tasks without centralized oversight or heavy communication, ensuring robust performance across different scenarios [59]. Regarding communication limitations within SRs, the literature highlights several forward-thinking solutions that improve system robustness and efficiency in environments with restricted communication. Approaches such as federated learning combined with deep RL have been noted to enhance performance generalization [60]. Additionally, biologically inspired communication strategies support decentralized swarm operations [61]. These methods boost the adaptability of swarm systems to dynamic conditions by enabling collective decision-making and task optimization without relying on complex individual robot capabilities.

Methodologies like the Mutual-Information Upper Confidence Bound (MI-UCB) [62], and virtual pheromone systems [63] have been explored to balance exploration and exploitation. MI-UCB, for instance, optimizes drone coordination via a decentralized approach, balancing information gathering with reward maximization. Virtual pheromones, on the other hand, allow agents to switch between discovering new resources and exploiting known ones efficiently.

Moreover, research on addressing energy and computational constraints has led to the development of mobile edge computing solutions paired with mobility-aware deep RL models, minimizing computational demands and enhancing energy efficiency while maintaining response times [64]. Our contributions build on these individual approaches by promoting a modular strategy combined with the adaptability and efficiency of PPO in dynamic settings. This method simplifies management and debugging, boosting system adaptability and operational efficiency in dynamic environments. Our approach significantly improves task performance and resource utilization by focusing on adaptive learning and decision-making within a collective swarm intelligence framework.

This chapter explores advanced enhancement techniques of DRL, focusing on innovative applications and hybrid designs in SRs. It contains two aspects:

Firstly, a structured approach to integrating curriculum learning with DRL is presented. It involves progressive training SRs to generate more generalized and adaptive collective behaviors.

Then, a new hybrid modular design model is introduced, combining automatic design methods like DRL and PSO with a modular design model. It is verified by tackling the dynamic foraging task, highlighting the proposed model's flexibility and efficiency. Detailed experimental setups and results illustrate how these enhanced techniques advance the capabilities of autonomous swarms, showcasing their potential for building adaptable and robust SRs.

3.1. Introduction to Curriculum Learning

Curriculum learning (CL) is inspired by the pedagogical approach of structuring education, where learners tackle complex topics gradually by beginning with basics and simpler parts until solving the entire task. So, CL is based on a gradually increasing difficulty.

This concept has been adapted to various machine-learning algorithms and applications [65]. By incorporating CL, models demonstrate improved generalization in new, unseen data. This approach also accelerates the training process, especially in non-convex scenarios where the optimization landscape contains multiple local minima. [66].

When using machine learning in robotics problems like navigation or path planning, incorporating CL enhances the robot's navigation skills by progressively training it on increasingly complex scenarios. This method improves the robot's adaptability and efficiency in varied environments. It is effective in mapping, localization, and optimizing path planning [67] and incorporating CL in two stages with the RL to solve navigation and obstacle avoidance in a 2D simulated unmanned aerial vehicle environment. This approach significantly enhanced learning efficiency by reducing the learning time and cost [68].

The primary goal of CL is to improve the learning process's efficiency. The challenge of an RL-driven swarm of robots is training them to learn cooperative strategies and adapt to complex environments effectively. CL addresses this by segmenting the learning into simpler, progressively more complicated tasks, making it manageable. Starting with easier tasks enables SR to succeed quickly. It helps prevent being stuck in suboptimal solutions. Skills acquired in basic scenarios are transferable to more complex ones, where transfer learning is vital in SRs. That led to enhancing the swarm's capability in varied scenarios. It can also introduce the robots to diverse environmental conditions and task variations from the outset, boosting robustness and adaptability.

Deploying CL across various methods to solve tasks in a swarm serves as a powerful tool to enhance RL's adaptability and generalization in complex scenarios, here are some examples. The author in [69] utilizes a centralized critic network of MADDPG (Multi-agent DDPG) for a group of defended drones, enabling them to adapt their strategies based on a shared understanding of successful behaviors. CL is used to enhance this network, where direct learning of cooperation is challenging; instead, the process starts with two agents and gradually expands to include the entire system. In my research, CL is applied more broadly, independent of any specific RL algorithm

(PPO, DDPG, and others), it improves learning efficiency by progressively increasing environmental complexity. This highlights how CL has been applied from very different perspectives, demonstrating its flexibility in addressing diverse challenges within swarm systems. The "Autonomous Swarm Shepherding" model in [70] employs a hierarchical RL framework, where a single agent (the dog) guides non-learning agents (the sheep) through curriculum learning (CL), progressively training it to collect and drive the sheep. However, this approach limits adaptability, as only one agent learns while others remain passive. In contrast, our model enables all agents in the swarm to learn independently, significantly increasing system complexity and realism. Our CL-RL integration teaches collective behaviors rather than focusing on a single leader, ensuring scalability across diverse swarm configurations. Unlike hierarchical RL, our approach addresses multiple objectives simultaneously, avoiding unnecessary computational complexity while improving learning efficiency.

This section introduces our significant contribution to the field: a model that integrates CL with DRL to address a navigation challenge for SR. Initially, this model was tested on individual robots before extending its application to a swarm setting [71]. Specifically, we have enhanced the efficiency of the PPO algorithm by incorporating a CL, significantly boosting adaptability and convergence efficiency in complex environments [57]. A comprehensive comparative analysis of three models is conducted to evaluate the effectiveness of the approach: modified PPO (PPO+CL), the standard PPO, and the DDPG. This comparison highlights the improvements the proposed model offers over existing methods.

3.1.1. PPO with CL for individual robots

Traditional PPO in Figure 2.1 is used with a given reward in equations (2.4, 2.5, 2.6). To investigate the effectiveness of PPO with CL in solving the navigation problem as in Figure 3.6. The environment is segmented into many sub-environments for training, each measuring 0.7×0.7 m^2. The first training environment is the simplest without obstacles, as shown in Figure 3.1. In contrast, the second one is more complicated, with multiple small boxes and narrow passages, as shown in Figure 3.2. The third one contains a long obstacle, as in Figure 3.3. The final training environment is combined with previous ones, measuring 1×1 m^2, as shown in Figure 3.4. The environment in Figure 3.5 is designed to have a new distribution of obstacles in a new area that

the robot has not encountered during training, considering that the testing environment combines training environments.

Figure 3.1. Training environment 1- 0.7×0.7 m^2.

Figure 3.2. Training environment 2- 0.7×0.7 m^2.

Figure 3.3. Training environment 3- 0.7×0.7 m^2.

Figure 3.4. Training environment 4- 1×1 m^2.

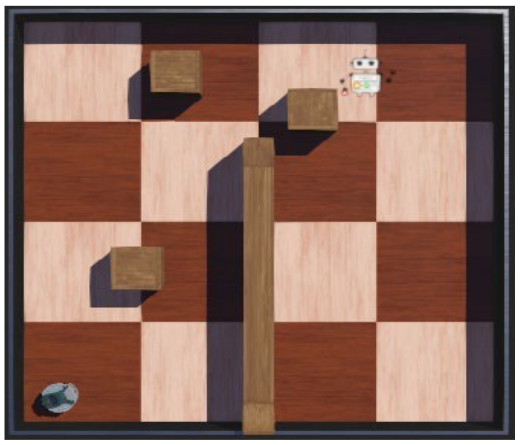

Figure 3.5. Testing environment 5- 1×1 m^2.

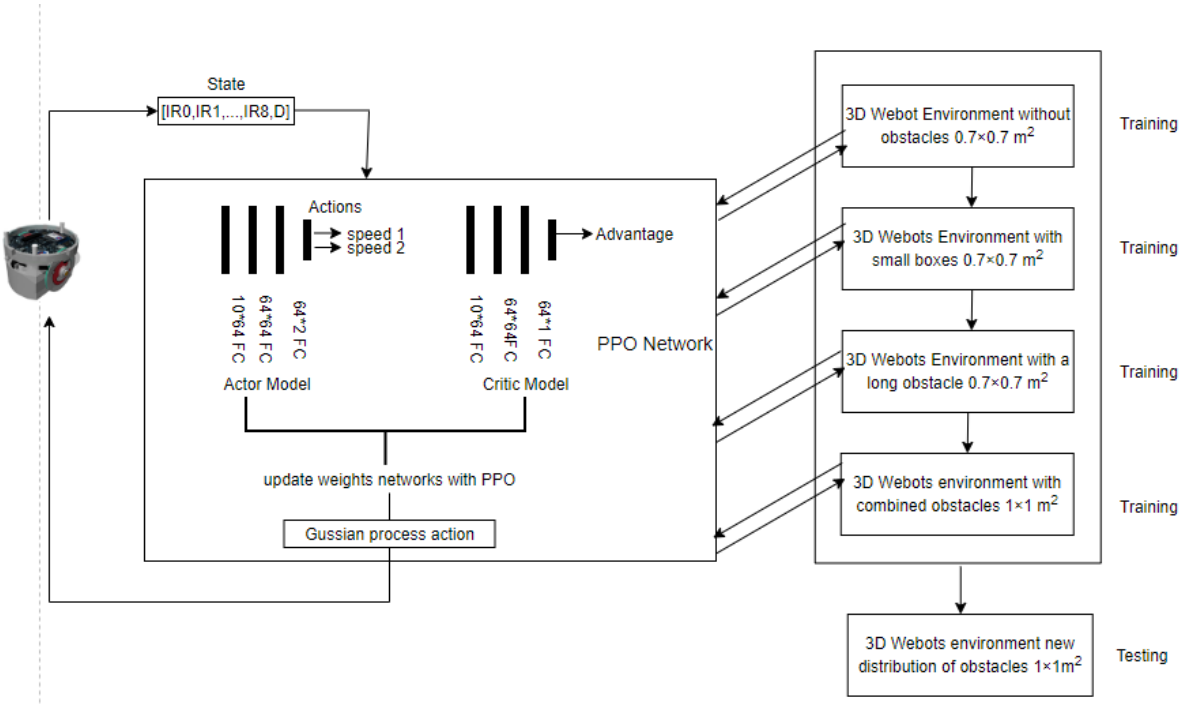

Figure 3.6. PPO with CL schematic.

3.1.2. Convergence efficiency

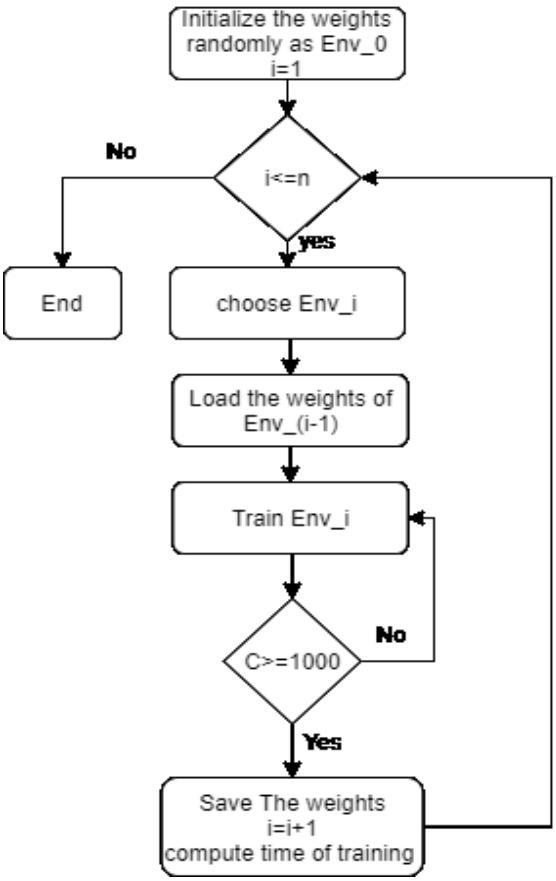

Figure 3.7. Curriculum learning- Training procedure.

The flowchart in Figure 3.7 illustrates the proposed training process of PPO with CL. Firstly, the weights are initialized with random values for (Env_0). Then, the training process continues sequentially as a series of incremental challenging environments i=1,2,...n, (n=4) in the proposed model. Each environment (Env_i) begins the training after transferring the learning from the previous environment by uploading the weights from the previous one (Env_(i-1)). When the robot records 1000 successful attempts to reach the goal, the model is learned, and the weights are saved. It is called C criteria. The training time is computed as ($t_{Env1}, t_{Env2}, t_{Env3,} t_{Env4}$), and the process iterates to the following environment. The cycle continues until the model has been trained among all environments. The convergence efficiency is measured by the time of training as in equation 3.1:

$$Training\ time = t_{Env1} + t_{Env2} + t_{Env3} + t_{Env4} \quad (3.1)$$

In Env_1 (i=1), which considers the no-obstacles environment, as shown in Figure 3.8, the robot reached the target at around 124000-time steps, a noticeable increase in the cumulative received reward. Then, the weights of actor-critic networks are used to initialize the actor-critic to be trained in Env_2, where small boxes fill the environment, as shown in Figure 3.9. The robot needs around 236,000 timesteps to reach the target and avoid obstacles. It requires more time because of the penalties of collisions, which is necessary for learning to avoid obstacles effectively. The

environment Env_3 introduced a long obstacle that was considered a different challenge. The weights are transferred from Env_2 to initialize PPO. The robot is trained to maneuver in a shorter period; it takes about 32000 timesteps, as shown in Figure 3.10. This quick adaptation is interpreted as the transfer learning that enabled the robot to generalize its avoidance strategies. Env_4 combines previous ones; the robot mastered this environment in roughly 44,000 timesteps, as shown in Figure 3.11. This indicates CL's ability to speed up the training process by initializing it with suitable weights. PPO without CL and DDPG are used to train the robot to navigate Env_4 to compare our approach. The robot reaches the target after 452000 timesteps with some fluctuations, indicating a less efficient learning process due to the complexity of the environment. DDPG failed to converge to the solution. Table 3.2 shows the readings of times to ease the comparison.

Table 3.2. Convergence efficiency.

	t_{Env1}	t_{Env2}	$t_{Env3,}$	t_{Env4}	$Training\ time$
DDPG	-	-	-	-	∞
PPO	-	-	-	452×10^3	452×10^3
PPO+CL	128×10^3	209×10^3	32×10^3	44×10^3	413×10^3

Table 3.1 compares the convergence efficiency of DDPG, PPO, and PPO with CL. DDPG did not successfully learn any environments, as indicated by infinite training time. PPO with CL shows progressive learning across environments with a cumulative training time. In contrast, standard PPO's slower learning in the final, most complex environment Env_4 indicates it requires more extensive exploration and training time to adapt without the structured progression that CL provides.

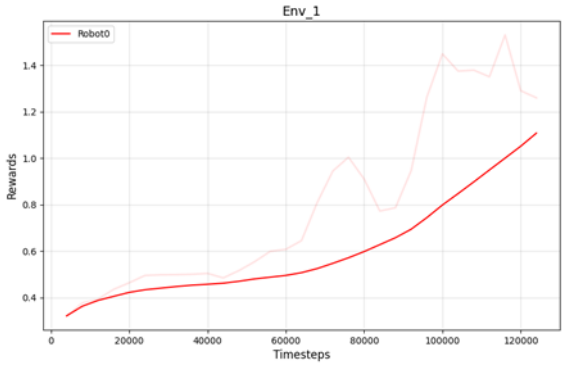

Figure 3.8. Env_1, rewards in training.

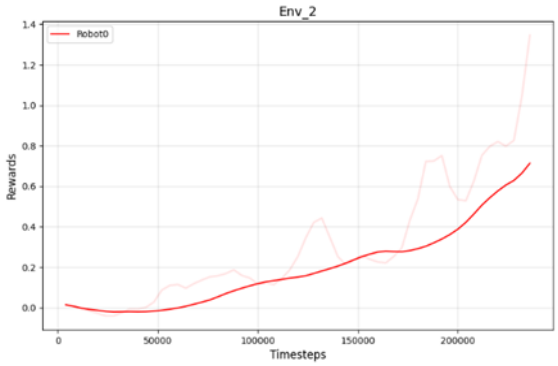

Figure 3.9. Env_2, rewards in training.

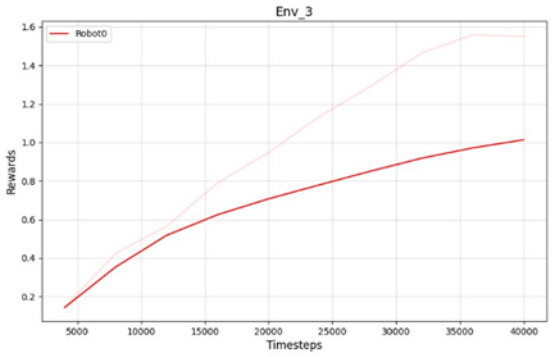

Figure 3.10. Env_3, rewards in training.

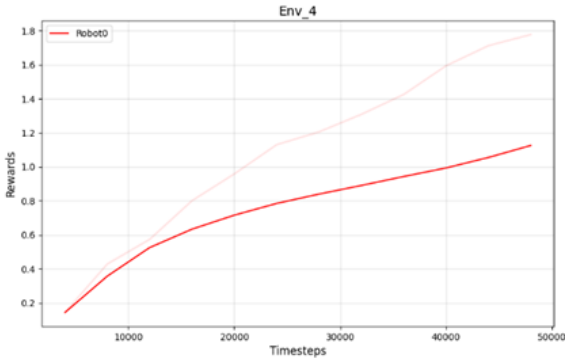

Figure 3.11. Env_4, rewards in training.

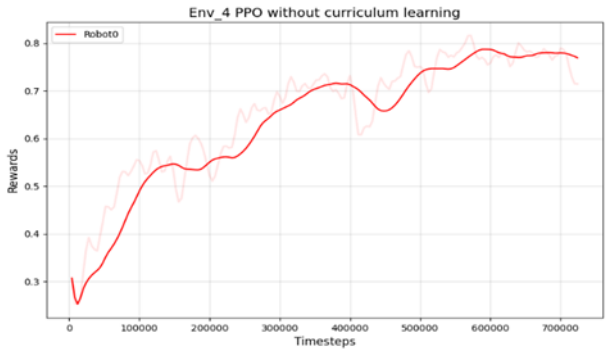

Figure 3.12. Env_4, rewards in training PPO without CL.

3.1.3. Robot's path planning

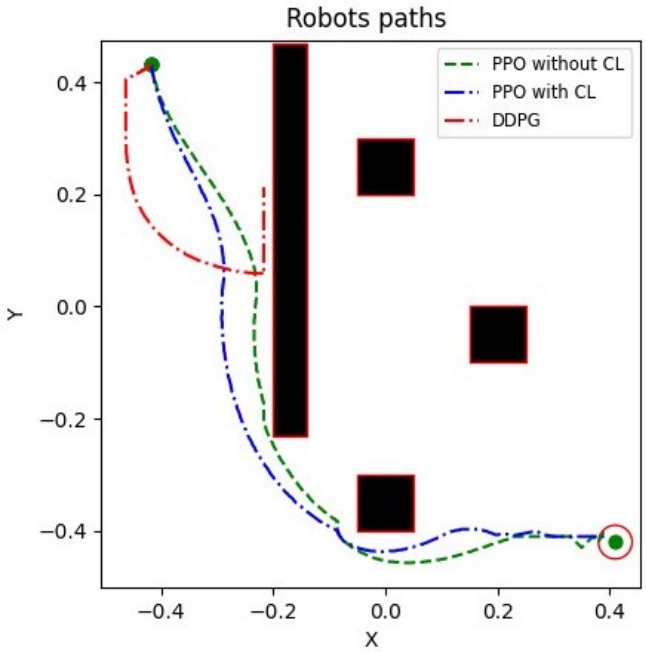

Figure 3.13. Robot's paths.

Figure 3.13 illustrates the outcomes of path planning using three different algorithms, PPO with CL, PPO, and DDPG, where the design of the actor-network of DDPG was replicated in the actor-critic network of PPO, maintaining structural consistency between them. Additionally, the same hyperparameter values were used for both of them. PPO with CL steers the robot along a more optimized trajectory, leading to a more direct approach to the goal and reflecting an enhanced navigation technique.

The trajectory produced by PPO without CL effectively avoids obstacles but is indirect and, thus, less efficient. On the other hand, DDPG does not reach the desired results in this scenario due to its failure to learn practical obstacle avoidance, with its performance heavily dependent on precise hyperparameter tuning.

The structure and hyperparameters optimized for PPO used in this study may not suit DDPG's needs. By starting with simpler challenges and progressively introducing more complexity, PPO with CL avoids overfitting to specific scenarios, potentially leading to a more broadly effective policy.

CL supports a more stable learning journey by segmenting the learning into minor, manageable phases. This is especially beneficial in RL, where extensive exploration might cause significant policy shifts that disrupt learning. The smoother trajectory observed with PPO and CL indicates a more consistent progression in learning.

3.1.4. Generalization

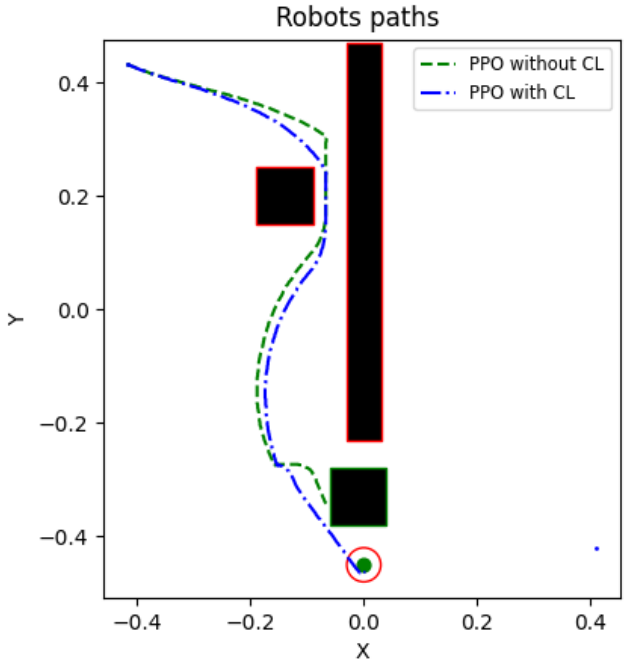

Figure 3.14. Ability to generalize.

Figure 3.8 illustrates the trajectories followed by a robot navigating in a modified environment where obstacles placements were changed. In this scenario, PPO with CL successfully adapts to the new modifications. This success can be attributed to the incremental learning approach of CL, which exposes the robot to a wide variety of situations during training. This diverse experience helps to develop a robust policy that effectively handles new and unexpected environmental changes. On the other hand, the version of PPO that lacks CL does not perform well in the altered environment.

This failure is likely due to the model's tendency to overfit to the specific conditions it encountered during training. Without the varied exposure provided by CL, the robot's strategy remains inflexible and less capable of adapting to environments that differ from those in which it was specifically trained.

3.1.5. PPO with CL for swarm robots

Challenges frequently emerge in complex settings or tasks when employing DRL, particularly in training SRs. A previous section illustrated how integrating DRL with CL can address these issues. The combination is particularly effective in overcoming the challenges posed by randomly initialized network parameters in DRL systems. Random initialization produces an initial policy that may be far from optimal. It takes significant time in complex environments to converge to a desirable solution, or it may get stuck in suboptimal solutions.

The exploration phase can be time-consuming and resource-intensive. CL helps by structuring the learning process in stages, gradually introducing the swarm robots to increasingly complex scenarios. This structured approach contrasts sharply with typical DRL methods, where robots might struggle with adaptation due to the randomness of initial network parameters. By incrementally adjusting to more complex environments, PPO with CL enables the swarm robots to develop a robust policy that is adaptable and effective across varying conditions. We have extended the PPO with CL approach to be applied for SRs, as in Figure 3.15, to generate a navigation collective behavior. The objective is to train the robots to reach their targets while avoiding collisions with obstacles and each other. The swarm has five robots with two targets and an obstacle, as shown in Figure 3.15.

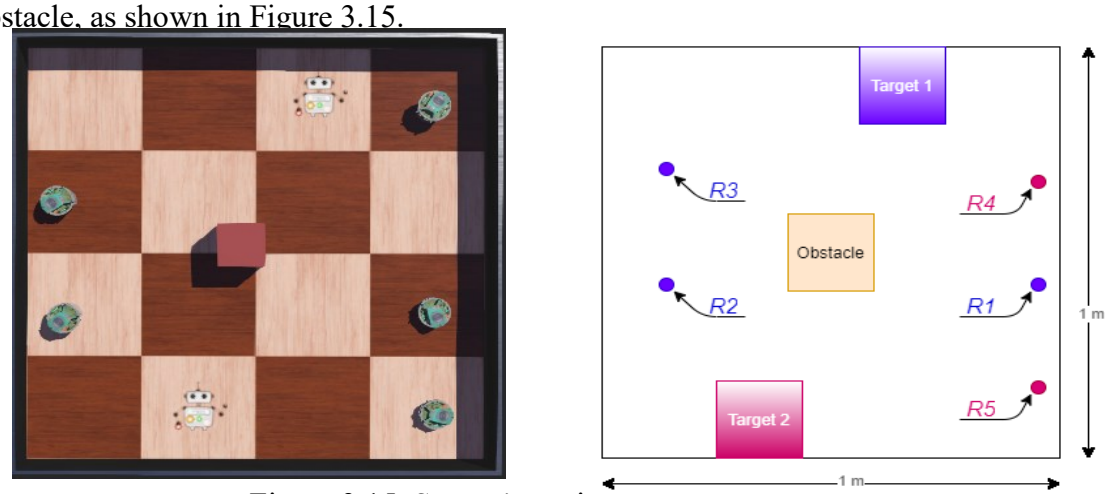

Figure 3.15. Swarm's environment.

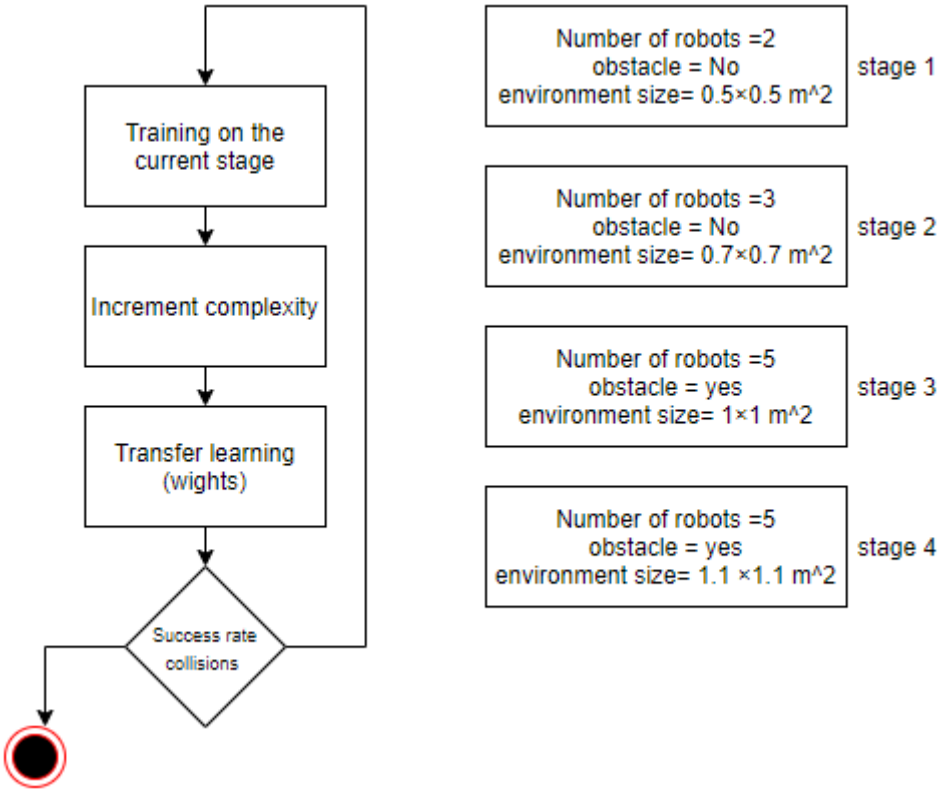

Figure 3.16. DRL with CL for SR.

As shown in Figure 3.16, the training process is iterative, gradually increasing the complexity of the stages and transferring the learning at each stage by uploading the weights from the previous stage. The decomposition process of the training environment is obtained based on three metrics: swarm sizes (2 robots, 3robots, and five robots), collision avoidance complexity (the existence of the obstacle or not), and the distances between the targets and robots (by changing the size of the environment from 0.5×0.5 m^2, 0.7×0.7 m^2,0.1×0.1 m^2, and 1.2×1.2 m^2). We assess the swarm's performance at each stage by measuring the success rate (percentage of targets reached) and collision rate.

Figure 3.17 clearly illustrates the comparative success rates of robots trained by PPO with CL and PPO, and Figure 3.18 demonstrates the avoiding collision performance. They demonstrate significantly superior performance of PPO with CL.

These robots could navigate and explore the environment more swiftly and efficiently and apply their acquired knowledge to achieve designated objectives. In contrast, robots trained using the PPO alone showed lower success rates. CL enhances robot training by introducing tasks incrementally, which prevents overfitting and builds a robust policy.

This method allows robots to generalize their skills better across diverse environments, leading to higher success rates. In contrast, PPO may limit robots to specific scenarios, hindering their adaptability and overall performance.

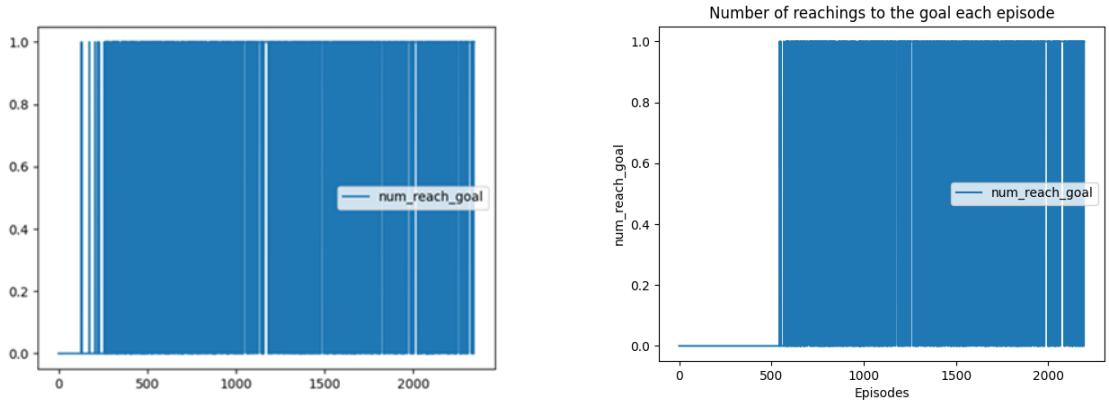

Figure 3.17. Success rate 1. PPO with curriculum learning 2. PPO without curriculum learning.

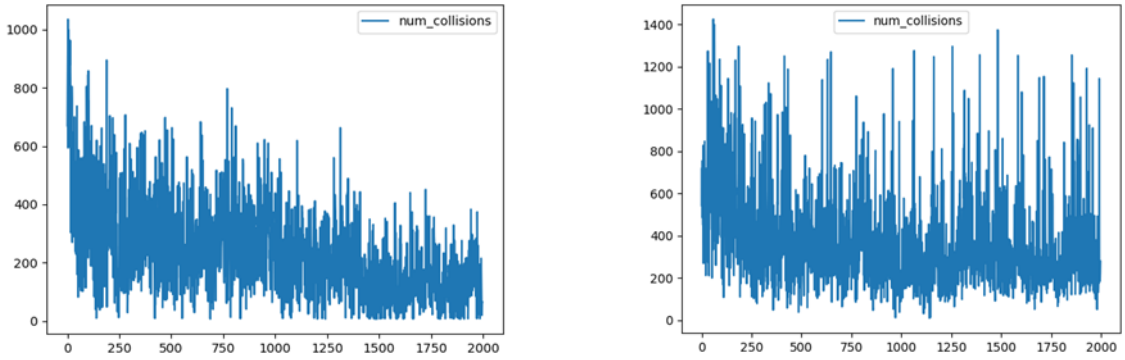

Figure 3.18. Collison rate 1. PPO with curriculum learning 2. PPO without curriculum learning.

The graph in Figure 3.19 contrasts the cumulative rewards earned by robots trained with PPO and those trained using PPO with CL. The robot employing PPO with CL achieves higher rewards more rapidly, reflecting a more effective training process. This superior performance is likely due to better initial weight settings, as shown in the values of rewards at the beginning of the training process, a structured learning path that incrementally introduces challenges, and more efficient exploration strategies. Additionally, the CL approach equips the robot to handle environmental variability better and avoid overfitting, thus enhancing its ability to generalize skills to new situations.

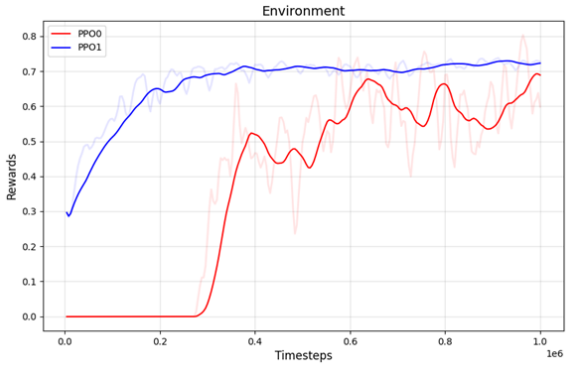

Figure 3.19. The average cumulative rewards: PPO0 without curriculum. PPO1 with curriculum.

When the environment is enlarged from 1.2×1.2 m^2 to 1.5×1.5 m^2, a swarm of five robots trained using PPO with CL demonstrates significantly improved performance. As the size of the environment increases, the CL framework aids the swarm in developing and refining effective navigation strategies in larger spaces. In contrast, robots trained with traditional PPO struggle to adapt, as shown in Figure 3.20. These robots fail to achieve predefined goals.

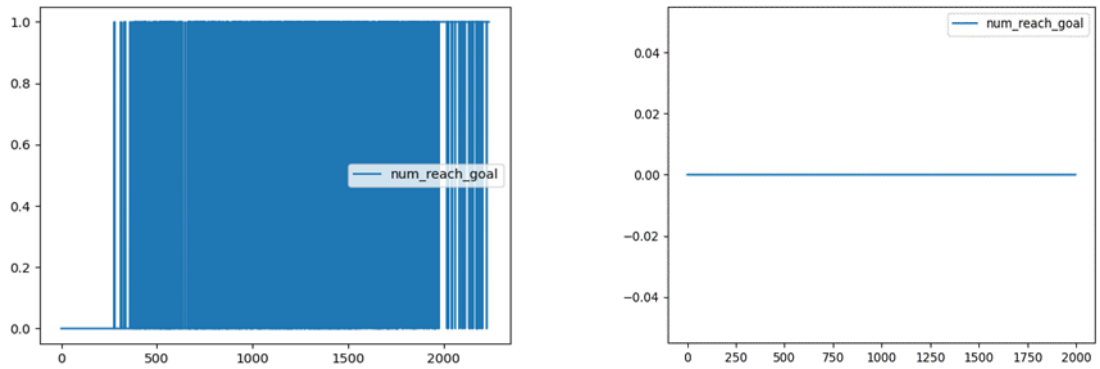

Figure 3.20. The success rate for expanded environment 1. PPO with CL 2. PPO without CL.

3.2. New Hybrid Modular Design with DRL and PSO

Foraging swarm behavior refers to the collective effort of a group of robots or organisms working together to locate, gather, and transport resources back to a designated location. This behavior is characterized by coordination and communication among individuals to optimize the efficiency and effectiveness of the resource collection process.

It mimics natural systems, such as ants or bees, which exhibit sophisticated group strategies to maximize their foraging success. This section introduces a foraging swarm system with a hybrid model combined with modular design and the deployment of automatic design methods like RL and PSO [72]. The swarm system is simulated in a 3D robot simulator, Webots.

The E-Puck robot is selected to construct the SR. Foraging collective behavior is required to be generated to search for two size boxes, small and big ones, through the environment and then transport them to the nest. The environment is shown in Figure 3.21. The workspace dimensions are defined as $3 \times 3\ m^2$. The parameters of E-Puck robots are set as in section 2.2.1. A group of robots gathered in the nest area as initial positions; they had to search for boxes in the environment. Then, if one box is found, the robot checks if it can grab it (small box); otherwise, it waits for help from another to grab it together (big box). Finally, transport the box to the nest to begin another round until all the boxes are collected.

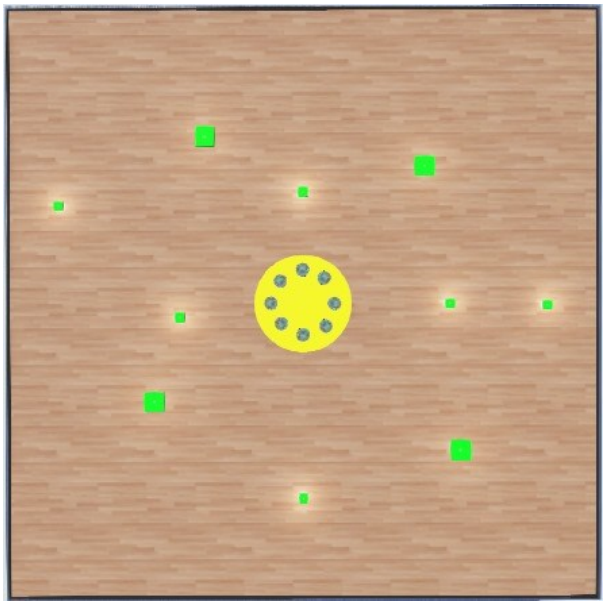

Figure 3.21. Foraging environment.

Based on the behavior-based model in section 1.2.4, the foraging task is decomposed into many modules.

- **Searching module**: The robots search for boxes and locate them using light sensors, where boxes emit light at different intensities. They use PPO or PSO.
- **Gripping module**: Robots catch the box once a box is located.
- **Waiting module**: This module is for big boxes that require cooperative work.
- **Transporting module**: Robots navigate back to the nest using PPO.
- **Release module**: Robots release the grasped box when it reaches the nest.
- **Return module**: Robots return to the searching module, creating iterable rounds until all the boxes are collected.

Figure 3.22 illustrates the flow process of the proposed system; the system initially sets up the positions of all components: the robots, foraging boxes (targets), and the nest. Once set up, the robots begin the search phase, employing either a PPO model or a PSO algorithm tailored for this task. This phase directs the robots to the targets using inputs from light sensors, which detect light emitted by the boxes. Upon locating a box, the robot assesses the box's size to decide the following action. A single robot can manage retrieval for smaller boxes, while larger boxes require waiting for an additional robot to assist, facilitating cooperative transport. This scenario mirrors real-world tasks that demand varying degrees of effort and teamwork. Once a box is located, the robots switch to a navigation mode, guided by another PPO model that calculates the optimal route back to the nest by considering the distance and angle to the nest. Following a successful delivery, the robot determines whether any targets remain. If targets are left, it resumes the search phase, thus perpetuating the foraging cycle. The simulation concludes once all targets have been collected.

$$R(t) = \frac{\left(LS_0^{(t)} - LS_0^{(t-1)}\right) + \left(LS_7^{(t)} - LS_7^{(t-1)}\right)}{2} + r_{box}(t) \tag{3.2}$$

$$r_{box}(t) = \begin{cases} 1.1 & LS_0^{(t)} > FindThreshold_{searching} \\ 1.1 & LS_7^{(t)} > FindThreshold_{searching} \\ 0 & Otherwise \end{cases} \quad (3.3)$$

The threshold $FindThreshold_{searching}$ is set to 0.85. So, when the normalized readings of the sensors are more than 0.85, the robot can catch the box. The module is set to the transporting module where rewards are used in Equations (3.4) for successfully retrieving when the threshold $FindThreshold_{transporting}$ is less 0.2. The nest circle has a radius of 0.2, so if the distance between the robot and the center of the nest is less than 0.2, the robot is in the nest. Equation (3.5) for leveraging the experience each time step to speed up the learning process by considering the angle and the distance to know if the robot is in the direction of the nest. These equations reflect the situation where collaboration is not required if one robot is enough to transport a small box to the nest. Still, when collaboration is needed in case two robots have to transport a big box to the nest together, Reward equations (3.6) and (3.7) reflect the robots' behavior in the swarm. When two robots can catch a big box, a positive reward for catching the box together and keeping the distance between the two robots less than 0.035 m, or a negative reward when it is bigger than 0.1. The overall reward in equation (3.8) includes the difference in distance to the target between the previous and current time step, the target reward, the cosine of the current angle to the target, and the distance reward.

$$r_{nest} = \begin{cases} 0.1 & d_{current} < FindThreshold\ . \\ 0 & otherwise \end{cases} \quad (3.4)$$

$$reward = (d_{prev} - d_{current}) + r_{nest} + \frac{cos(\alpha_{current})}{1000} \quad (3.5)$$

$$dis_{reward} = \begin{cases} 0.01 & 0.035 \leq d_{robots} \leq 0.1 \\ -0.001 & d_{robots} < 0.035 \\ -\frac{d_{robots}}{100} & otherwise \end{cases} \quad (3.6)$$

$$r_{nest} = \begin{cases} 0.15 & d_{current} < threshold\ and\ friend\ is\ present. \\ 0.1 & d_{current}\ threshold\ and\ friend\ is\ not\ present. \\ 0 & otherwise. \end{cases} \quad (3.7)$$

$$reward = (d_{prev} - d_{current}) + r_{nest} + \frac{\cos(\alpha_{current})}{1000} + dis_{reward} \quad (3.8)$$

In reward formulation, parameters such as radius = 0.2m, robot distance < 0.035m, also 1000 in the fraction $\frac{cos(\alpha_{current})}{1000}$, and others were chosen empirically based on problem constraints. These values were tested iteratively to ensure realistic behavior, considering factors like robot size and environment constraints. This highlights the challenge of manually designing reward functions, motivating our proposed RL-IRL model to learn rewards automatically.

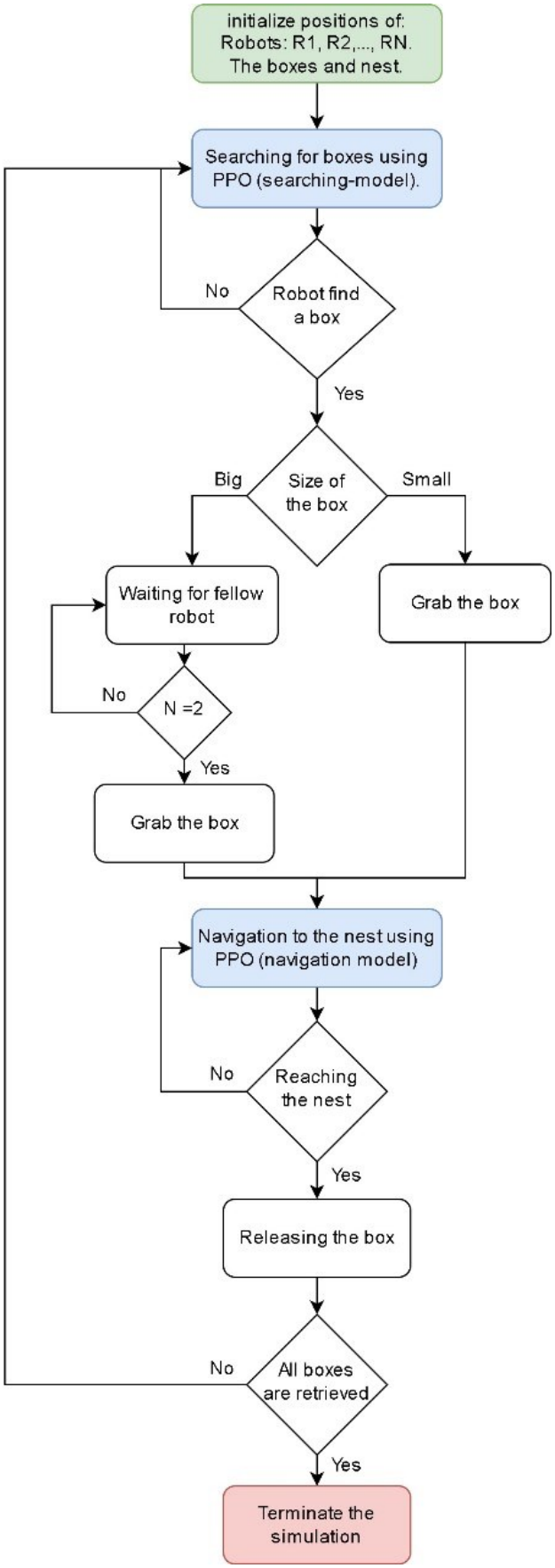

Figure 3.23. Hybrid modular design for foraging swarm.

3.2.1. Results and performance analysis

The hybrid modular configuration described in Figure 3.23 utilizes PPO and PSO to improve decision-making processes in search and navigation tasks. Simultaneously, It reduces demanding computations, such as gripping or releasing and switching between modules because of the behavior-based model. This method boosts computational effectiveness and supports tailored optimization where needed. Several benefits arise from this hybrid modular strategy:

- Specialized Optimization: PPO is integrated into critical modules to enhance task performance, notably in search and transport activities. This ensures optimal use of PPO's capabilities.
- Computational Efficiency: PPO, a resource-intensive algorithm, is selectively applied to manage the computational load effectively. This is essential for controlling numerous robots with limited processing abilities.
- Simplicity in Routine Tasks: Simpler tasks, like gripping or releasing, utilize straightforward control schemes that do not require complex decision-making and facilitate system programming and maintenance.
- Minimized Overfitting Risk: Restricting PPO to complex tasks helps avoid overfitting, keeping the model versatile and suitable for various situations.
- Accelerated Training Periods: Concentrating on specific modules decreases the total time required for training, thus expediting system rollout and adaptation.
- Optimized Reward System: The reward framework is carefully designed to match the objectives of each module, ensuring the primary aims are met and avoiding unintended actions.
- Comprehensive System Autonomy: when the system encompasses an end-to-end deep RL architecture that manages all behaviors—ranging from navigation to gripping—across the entire swarm of robots, it is purely on autonomous decision-making, which may not be the most efficient. So, integrating with rule-based components enhances the system's resilience."

A. Foraging performance (PSO-PPO vs PPO-PPO)

The hybrid modular design stands out for its ability to facilitate the testing and integration of various algorithms, including PPO, PSO, and beyond. This flexibility enables the system to adapt dynamically, evaluating different computational strategies under consistent conditions. The 3D visualizations in Figures 3.24 and 3.25 illustrate the performance of foraging behavior according to PSO-PPO and PPO-PPO as automatic design methods.

Numerical samples in Table 3.3 are also derived to measure the performance as follow: For the number of retrieved boxes $N \epsilon [0,10]$. ΔT is the measured time to collect and transport all the boxes to the nest. P_N is the average path length of the SR needed to find and transport all the boxes. So, Efficiency E is defined as the number of boxes retrieved per unit of time and effort.

Efficiency can be calculated using the parameters ΔT, P_N, and N by Equation (3.9) and Table 3.3.

Table 3.3. Foraging performance metrics.

Retrieved items (Boxes)	PPO-PPO		PSO-PPO	
	Time (sec)	Average Path (m)	Time (sec)	Average Path (m)
1	13.024	0.726	7.552	0.377
2	13.696	0.774	25.024	1.173
3	27.712	1.767	36.928	1.719
4	33.92	2.243	78.08	3.513
5	36.512	2.434	98.112	4.451
6	59.488	4.129	125.92	5.725
7	61.408	4.268	163.776	7.436
8	69.92	4.942	181.6	8.256
9	72.192	5.126	272.64	12.051
10	88.096	6.305	320.096	14.211

$$E = \frac{N}{\Delta T \times P_N} \tag{3.9}$$

$$E_{PPO-PPO} = \frac{10}{88.096 \times 6.31} = 18 \times 10^{-3}$$

$$E_{PSO-PPO} = \frac{10}{320.096 \times 14.21} = 2.19 \times 10^{-3}$$

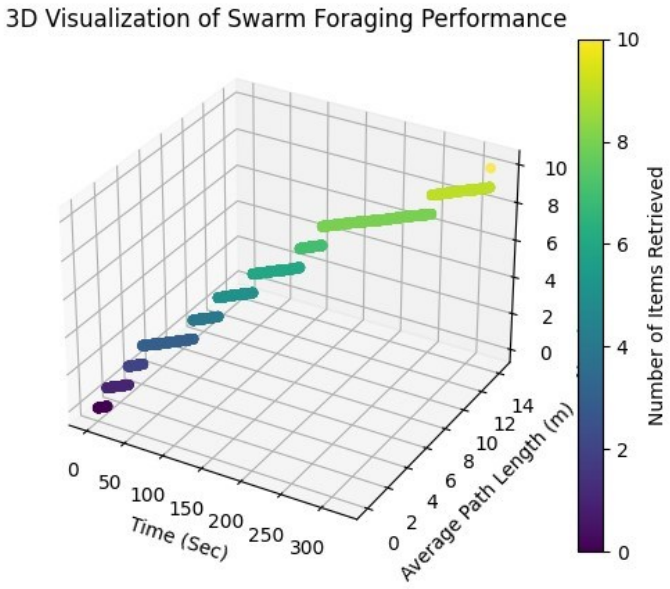

Figure 3.25. PSO-PPO-driven swarm.

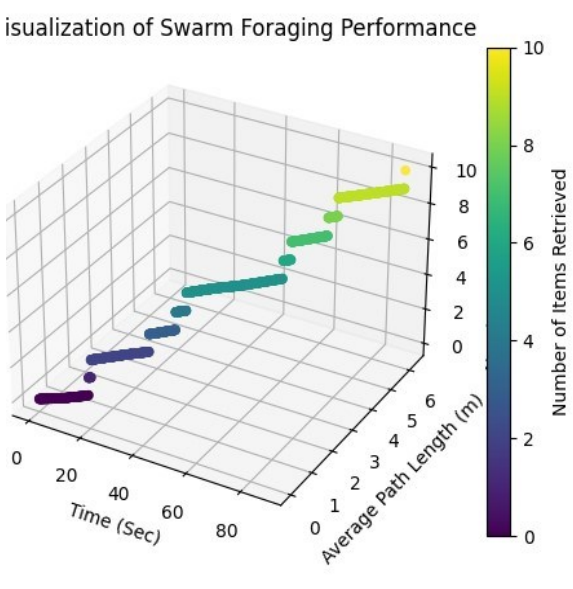

Figure 3.24. PPO-PPO-driven swarm.

The swarm guided by the PPO-PPO model has shown greater efficiency than the one driven by PSO-PPO. The analysis reveals that the PPO-equipped swarm is more adept at item retrieval, evidenced by a more pronounced increase in the number of items collected over time. This system also benefits from shorter average journey lengths.

Conversely, the PSO's performance graph displays a softer ascent, suggesting longer durations required to complete the foraging tasks. These findings underscore PPO's superior ability to adapt quickly and handle the task more effectively. PPO excels not only in learning speed but also in maintaining robust performance levels.

This advantage stems from the policy gradient optimization inherent to PPO, which fine-tunes the robotic actions based on rewarded outcomes, resulting in a more optimized strategy. In contrast, PSO is prone to settling at local optima and lacks precise adjustment capabilities. Additionally, PSO's dependence on the collective dynamics of the swarm can decrease the performance when individual robots do not effectively mimic or communicate within the swarm.

The mean execution time for PPO-PPO is 47.60 seconds, whereas PSO-PPO requires 130.97 seconds on average to complete the same task. Furthermore, the standard deviation for execution time in PPO-PPO is 26.08 seconds, whereas PSO-PPO has an extremely high deviation of 104.71 seconds. This suggests that PPO-PPO demonstrates stable and predictable execution times, whereas PSO-PPO exhibits high fluctuation, making it unreliable for real-world deployment where timing consistency is crucial.

In terms of navigation efficiency, PPO-PPO follows a significantly shorter path with an average distance of 3.27 meters, compared to 5.89 meters for PSO-PPO. This demonstrates that PPO-PPO optimizes movement better, allowing agents to reach targets more directly and energy efficiently.

The standard deviation of the path length follows the same trend—PPO-PPO exhibits a deviation of 1.94 meters, while PSO-PPO has a higher deviation of 4.63 meters. This confirms that PSO-PPO's paths are inconsistent and sometimes highly suboptimal, whereas PPO-PPO maintains a more structured and predictable trajectory. That suggests that PSO-PPO frequently takes inefficient paths, which could lead to increased energy consumption and longer exploration times.

B. Dynamic behavior and autonomy

The behavior of SR for dynamic foraging is analyzed for two proposed systems, PPO-PSO and PPO-PPO, based on the proposed hybrid modular model. They are used to collect two boxes where these boxes are not static. They move through the environment randomly. In this dynamic situation, the SR follows each box until it is grabbed. When grabbed, the box changes to red; it stops its dynamic nature, moving to a static box to be transported to the nest (yellow area).

- Behavior analysis for PSO

Figure 3.26 illustrates the PSO's capability to identify moving objectives in the environment.

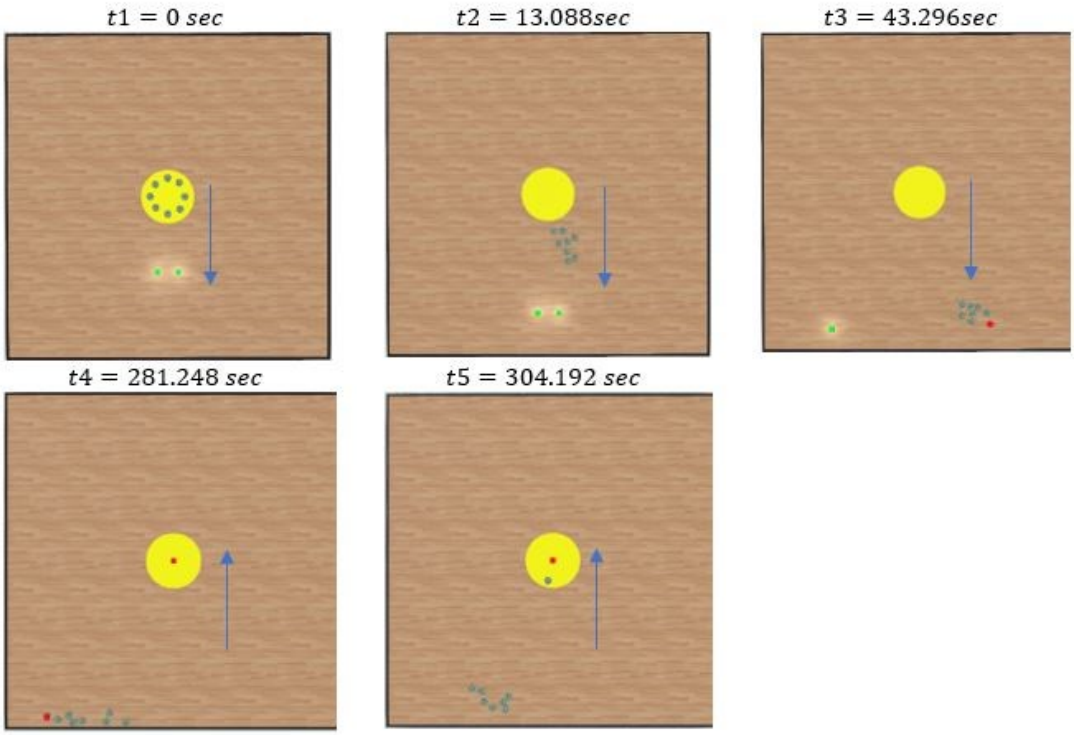

Figure 3.26. Dynamic Foraging performance/ PSO-driven swarm.

Initial response (t1-t2): The swarm begins at the nest's initial location and then tracks a single moving box without the other, showing powerful collective behavior with minimal autonomy.

Mid-phase (t3): The swarm locates the green box and forms a tight group around it. The robots' movements are heavily influenced by their surroundings.

Gripping action (t4-t5): Upon grasping the box (signified by a shift in its color to red), the robot holding the box heads back to the nest. The robot then releases the box and reverts to PSO mode.

- Behavior analysis for RL

Figure 3.27 illustrates the RL's capability to identify moving objectives in the environment.

Initial response (t1-t2): The RL swarm moves in a distributed, exploratory form, suggesting an exploratory approach. This allows the swarm to track both moving boxes simultaneously, indicating high autonomy.

Mid-phase (t3): The swarm methodically adapts to the dynamic target, determining the box's location more rapidly than the PSO, showcasing superior adaptability to the target's changing position.

Gripping action (t3-t4-t5): After grasping the box, the robot transports it to the nest area, releases it, and locates another target using the PPO method.

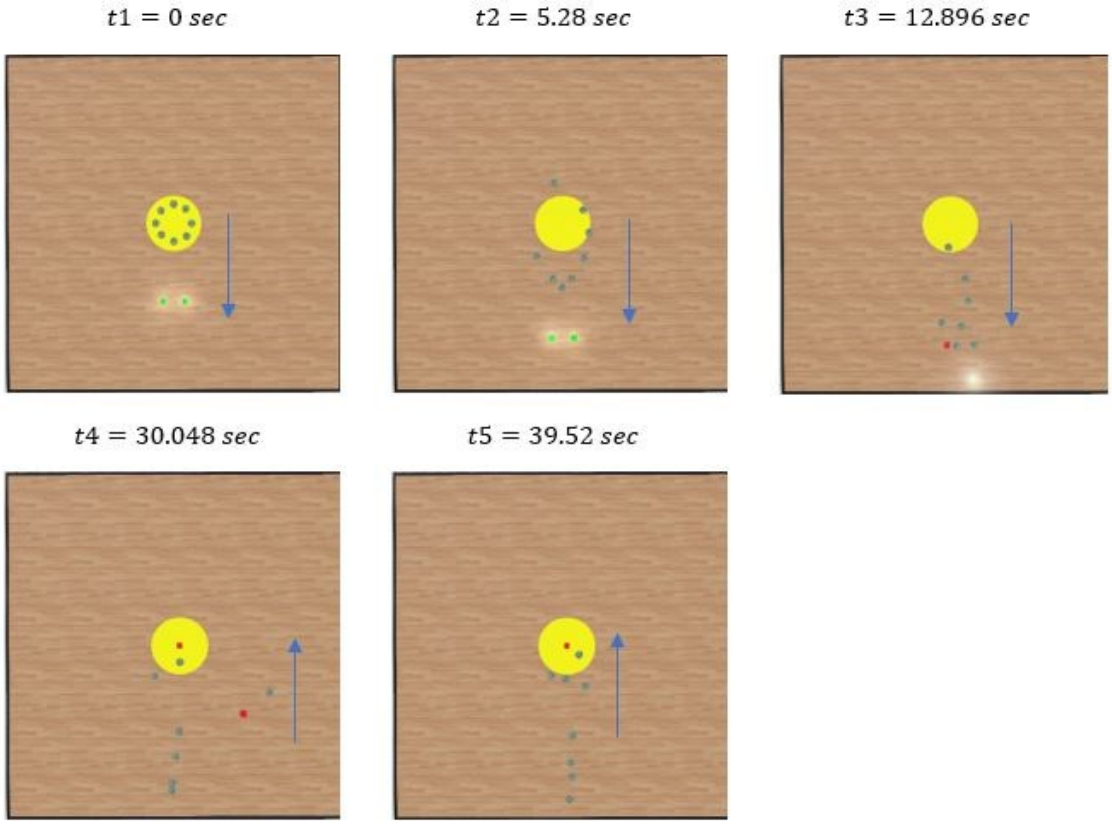

Figure 3.27. Dynamic Foraging performance/RL-driven swarm.

3.3. Conclusion of Proposed Enhancement Techniques

The study offered an in-depth evaluation of the impact of CL on enhancing the training process and leverages combining a behavior-based model with automatic design methods like RL and PSO models to improve the swarm's behavior. CL specifically assessed the efficiency of SR trained in a dynamic 3D environment. By structuring the navigation task into increasingly complex stages, the research highlighted the robots' ability to adapt to intricate scenarios effectively. Robots equipped with the PPO augmented by CL demonstrated superior path-planning capabilities in environments with variable obstacle configurations. This enhanced capability allowed for a more generalized application of learned behaviors to new environments. Furthermore, the study utilized complexity metrics, considering factors like swarm size, collision avoidance, and the size of the environment. The curriculum-based training achieved a higher success rate in reaching targets and reduced collision rates through improved obstacle avoidance tactics. This method also accelerated the learning process, as evidenced by faster convergence times. The swarm trained with CL demonstrated enhanced performance metrics, robust generalization, and adaptation abilities regarding training and operational efficiency. The number of stages and the incremental adjustment of parameters at each stage are determined based on experience, taking into account computational costs and the risk of sub-optimal policies. This approach paves the way for future improvements, allowing the model to autonomously optimize these values for enhanced efficiency.

The other proposed model delves into a dynamic foraging challenge; it suggests that the strategy integrates a modular framework that manages activities such as gripping, waiting for assistance, and locating the box, paired with a sophisticated algorithm that facilitates the search and transport tasks using RL and PSO. This design enables continuous operation in varied environments while avoiding model overfitting. It also incorporates a module specifically for evaluating different algorithms. A detailed comparative analysis between PPO and PSO was carried out. The findings indicated that PPO was more effective, achieving quicker retrieval times and greater overall efficiency due to its superior adaptability and independence. On the other hand, PSO was less effective, showing limitations in both efficiency and autonomous function. Moreover, this study highlights the advantages of a modular design in SRs, laying the groundwork for future innovations that combine operational efficiency with adaptability.

4. REWARD STRUCTURES: IMPLICATIONS FOR BEHAVIOR OF SR

Deep RL has significantly advanced the capabilities of SRs, generating complex collective behaviors through decentralized decision-making processes. A critical component in DRL is the design of the reward structure, which guides the learning process and influences the swarm's emergent behavior:

- Learning Signal: The reward structure provides the feedback that the agent (robot) uses to evaluate the behavior based on their actions within an environment. This feedback loop is essential because it shapes the policy that guides agent behaviors. In SR, where multiple agents must coordinate, the reward structure often encodes the desired collaborative behaviors, influencing how individual agents contribute to the group's objectives.
- Behavior Shaping: Rewards play a crucial role in shaping the system's behavior by explicitly valuing specific actions over others. This shaping is particularly significant in SRs, where collective behavior emerges from interactions between multiple robots. For instance, a reward structure that emphasizes speed over accuracy can lead to the development of policies prioritizing quick movements and formations, possibly at the cost of precision or stability.
- Convergence to Optimal Policies: The reward design affects how quickly and effectively a learning algorithm can converge to an optimal policy. Therefore, a well-tuned reward structure is beneficial in accelerating learning by clearly generating effective strategies and reducing ambiguity in the rewards received. Poorly designed rewards, such as those that offer conflicting signals or insufficient differentiation between good and bad actions, can lead to slower learning, suboptimal policies, or failure to converge.
- Encouraging Exploration vs. Exploitation: As mentioned in section 1.3.2, agents must balance exploration (trying new actions to discover their effects) and exploitation (using known actions that yield high rewards). The complex and deep element reward structure significantly influences this balance; for example, a reward system that provides incremental feedback for novel strategies can encourage more explorative behaviors.

Reward shaping, sparse rewards, and Inverse RL (IRL) are three distinct methods used in deep RL to influence SR behavior. Reward shaping modifies the reward function by adding supplementary feedback to encourage specific behaviors, accelerate learning, and guide robots toward desired outcomes more efficiently. Sparse rewards, awarded only for significant actions like avoiding obstacles or reaching the goal, foster robust strategies without frequent feedback, simplifying reward design but potentially slowing learning and complicating exploration. Inverse RL derives rewards from observed optimal behaviors, enabling natural and efficient behavior learning without explicit reward programming. However, it relies heavily on the quality of demonstration data and involves greater computational complexity. These methods offer different advantages and drawbacks in training swarm robotics to achieve complex collaborative tasks.

Beyond the conventional reward methods discussed, such as reward shaping, sparse rewards, and Inverse Reinforcement Learning (IRL), several innovative approaches can further enhance the learning and performance of swarm robotic systems. These methods can be tailored to the unique challenges and opportunities presented by swarm behaviors and the complexity of their operational environments.
This section explores various reward methods, such as reward shaping and sparse rewards, mainly focusing on the impact of reward scaling in deep RL for SRs for these two methods. Finally, it introduces a new method of incorporating inverse RL with deep RL, which can deal with continuous systems and generalize to different environments. It also discusses its implications for swarm behavior in robotic systems.

4.1. Scales reward in Shaping and Sparse methods

This section delves into two primary methods of configuring rewards: Shaping and Sparse methods. Sparse rewards are structured so that the robot receives infrequent rewards, typically only for significant actions like reaching the target, gripping the box, and others rather than for every step or state transition. The general formula or representation for a sparse reward system can be described in a conditional format, where the occurrence of specific events primarily determines the reward as in equation (4.1):

$$R(s,a,\acute{s}) = \begin{cases} x & if\ condition\ met \\ P & otherwise \end{cases} \quad (4.1)$$

x: Represents the obtained value when the condition is met, typically ranging from xmin to xmax, where xmin ≥ 0 and xmax > xmin. *P*: Represents penalties when the condition is not met, ranging from *P*min to *P*max, where *P*min ≤ *P*max ≤ xmin. For example, in section 2.2.3 for equations (2.4) and (2.5), the sparse method has been used to reward the robot when it reaches the target or collides with obstacles.

The general formula for reward shaping involves modifying the original reward function $R(s,a,\acute{s})$ in equation (4.1). The shaping term is added to provide the robot with additional feedback to encourage specific actions. The modified reward function can be expressed as in equation (4.2):

$$\acute{R}(s,a,\acute{s}) = R(s,a,\acute{s}) + F(s,\acute{s}) \quad (4.2)$$

The shaping function $F(s,\acute{s})$ is carefully designed to align with the task's objectives, while ensuring that it does not change the optimal policy defined by the original reward function. Ideally, it should provide additional guidance to the learning process without altering the policy. A common choice for *F* is based on potential-based reward shaping, where:

$$F(s,\acute{s}) = \gamma\Phi(\acute{s}) - \Phi(s) \quad (4.3)$$

Equation (2.6) is a clear example of the shaping method. The term 'Previous distance - Current distance'. It rewards the robot for moving closer to the target where the shaping function F(s,s´) is defined as the reduction in distance from the target. By including this term, the reward function

directly encourages behaviors that decrease the distance to the target, facilitating faster and more focused learning toward the main objective.

Beyond the conventional reward methods discussed, such as reward shaping and sparse rewards, in addition to Inverse RL, several innovative approaches can further enhance the learning and performance of swarm robotic systems. These methods can be tailored to the unique challenges and opportunities presented by swarm behaviors and the complexity of their operational environments. They indeed have been used in the previous section as follows:

Multi-objective Rewards: They involve designing reward functions that consider multiple criteria simultaneously, which is essential for swarms that must balance competing objectives like orientation, collaborating, and others, as in equation (3.9)

Curriculum learning: This method can be structured so that swarm robots master individual skills before tackling collaborative strategies, as in section 2.2

Cooperative Reward Distribution: This method involves distributing rewards based on individual achievements and contributions to the swarm's collective success. It encourages cooperation and can be crucial in tasks that require tight coordination, like gripping the big box in equation (3.7).

Hierarchical Rewards: This approach decomposes complex tasks into simpler sub-tasks, each with its reward structure. It is particularly suitable for SR, where different layers of hierarchy could correspond to individual robot tasks, sub-swarm tasks, and overall swarm objectives, such as the mentioned section 3.2.1 to solve foraging tasks where sets of reward equations used to deal with small boxes (3.5), and (3.6) and other sets (3.7), (3.8), (3.9) to deal with big boxes. So, the tasks were divided into sub-tasks.

Understanding and selecting the appropriate reward scale is crucial for optimizing the efficiency and effectiveness of the learning process in RL. The analysis study is conducted using the primary approaches, sparse and shaping. It explores the balance between rewards for reaching goals and penalties for colliding with obstacles[73],[74]. Traditionally, fixed rewards and penalties have been used, but these may only sometimes yield the best performance across varying environments and tasks. By generalizing the reward to a variable x and defining the penalty P as a percentage of x as in equation (4.5)

$$P = -\alpha x \quad x > 0 \qquad (4.5)$$

So, The aim is to find the optimal balance that maximizes navigation efficiency and safety. Through a series of simulations for the environment in Figure 4.1, we evaluate the effects of different penalty percentages on key performance metrics, including the average time to reach the goal, the number of collisions, and the success rate of goal achievement. The results are analyzed to determine the optimal penalty percentage that provides the best trade-off between rapid goal attainment and minimal collisions. To find the optimal percentage between the reward x (for reaching the goal) and the penalty P (for colliding with obstacles) to achieve the best performance

in a robot navigation problem, we need to balance the motivation for the robot to reach the goal quickly while avoiding collisions effectively.

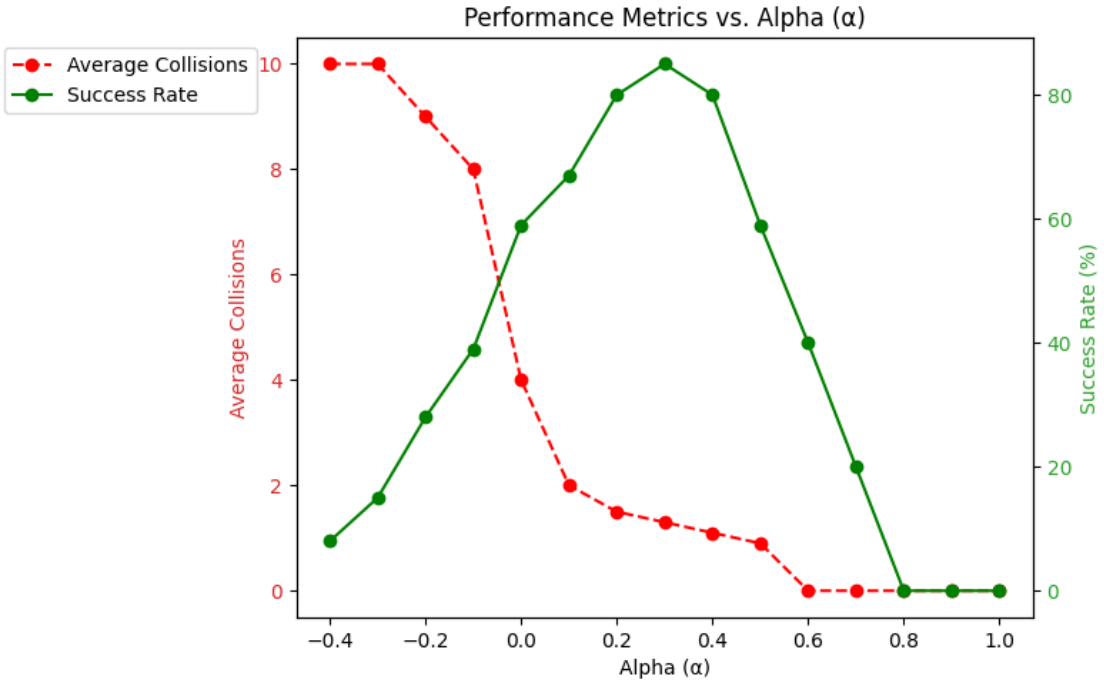

Figure 4.1. Reward scales in the sparse method.

The graph in Figure 4.1 presents the relationship between different values of α (the *P* as a percentage of x) and two key performance metrics for a robot navigation task: Average collisions and success rate during training, which records the number of reaching the goal. When α is lower than 0.4, approximately there is no penalty. The robot cannot achieve its goal because it is not punished when it distracts its path to the target or collides with walls, so it does not try to change its behavior. As α increases from 0.4 to 1, average collisions generally decrease, dropping to zero at α = 0.6. The success rate increases as α moves from -0.4 to 0.3, peaking at 85% at σ = 0.3. Beyond σ = 0.3, the success rate decreases significantly, reaching zero at σ = 0.8 and above. The data suggests that. The α value around 0.3 provides the best balance, achieving the highest success rate with relatively low collisions. Higher penalties $\alpha \geq 0.6$ effectively eliminate collisions but drop the success rate to zero, indicating that strict penalties discourage risky behavior but discourage task completion. By increasing the penalty, the learning algorithm focuses on avoiding obstacles rather than reaching the goal. The graph highlights a trade-off between minimizing collisions and maintaining a high success rate, showing that moderate penalties can enhance safety and efficiency. In contrast, overly harsh penalties may hinder the robot's ability to achieve its objectives. These findings can guide the design of reward structures in RL for autonomous navigation, optimizing the balance between safety and goal achievement.

To analyze the performance of robot navigation using the shaping method, we have to focus on the trade-off between the F value in equation 4.3 and P based on the graph in Figure 4.1. The

shaping reward is calculated based on the reduction in distance to the target ΔD_{rg}, with an added penalty for collisions P , equation (4.6),(4.7).

$$R = \Delta D_{rg} + P \quad where: \Delta D = D_{rg}(t-1) - D_{rg}(t) \tag{4.6}$$

$$D_{rg} = \sqrt{(x_r - x_g)^2 - (y_r - y_g)^2} \tag{4.7}$$

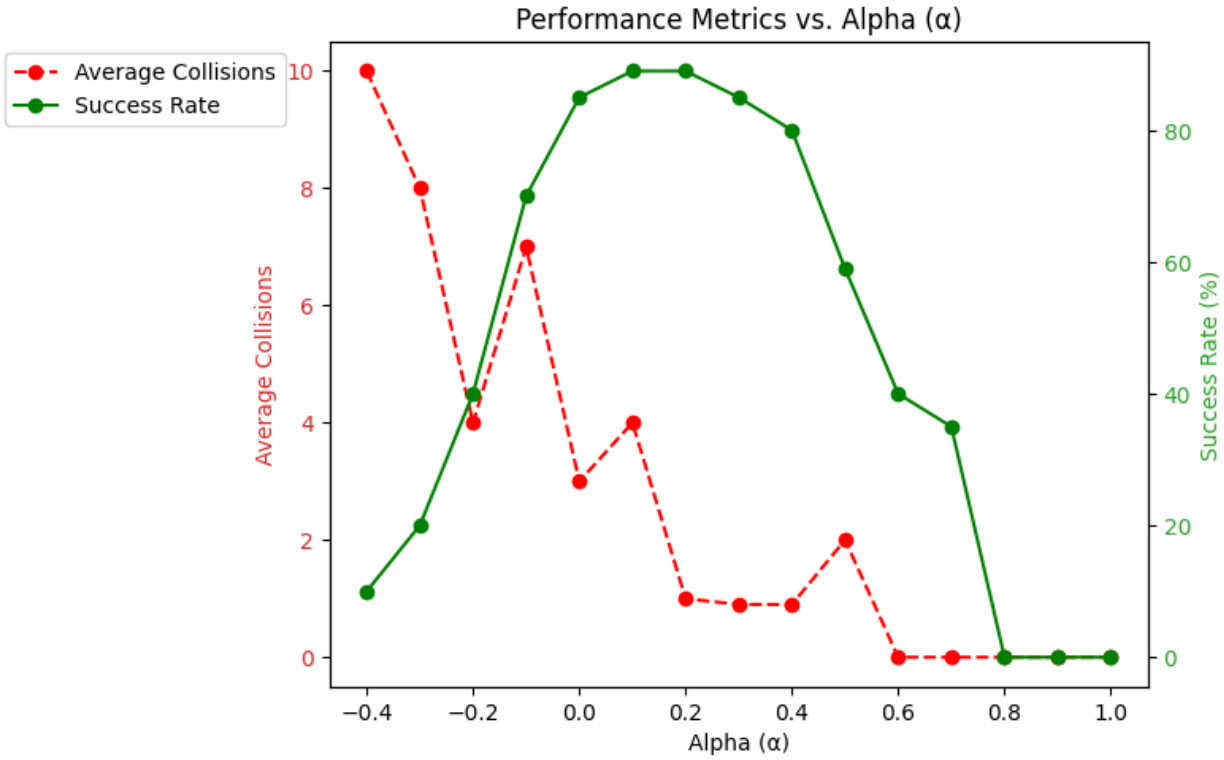

Figure 4.2. Reward scales in the sparse method.

By comparing the shaping method graph in Figure 4.2 to a sparse reward graph in Figure 4.1, it can be observed that the shaping method provides continuous feedback based on the robot's progress and penalties for collisions, leading to smoother performance curves and better adaptability. The success rate and collision metrics change more consistently across different σ values, with a clear optimal range (around 0.1 to 0.3) where the success rate peaks (89%) and collisions are minimized. In contrast, sparse rewards offer feedback only at specific events, resulting in slower learning and more abrupt performance changes. Identifying the optimal penalty range is more challenging with sparse rewards, and the robot might overfit the training environment due to less frequent feedback. These observations highlight the advantages of shaping methods in producing robust and efficient navigation strategies compared to sparse reward structures in complex or dynamic environments.

4.2. Inverse DRL for Swarm Reward Recovery

Inverse DRL for swarm reward recovery is an emerging approach that aims to figure out the underlying reward structures guiding the behavior of SRs. By observing the trajectories of the swarm, Inverse DRL techniques can infer the implicit rewards that drive effective coordination and task completion. This method not only helps in understanding the intrinsic motivations of the swarm but also aids in designing better reward mechanisms to enhance performance. Inverse DRL for swarm reward recovery leverages the strengths of inverse learning algorithms to improve the adaptability and efficiency of swarm systems in complex, dynamic environments.

4.2.1. Introduction to IRL

IRL is a machine learning framework that focuses on recovering the reward function that an agent is optimizing based on its observed behavior. This is in contrast to traditional RL, where the reward function is known, and the objective is to find the optimal policy. In IRL, the goal is to understand the motivations behind the observed behavior by inferring the reward function. This can be particularly useful in scenarios where the reward function is not explicitly defined but can be inferred from expert demonstrations. The key concept of IRL is expert demonstrations, which are collected data by observing an expert performing a task, assuming that an optimal policy generates the observed behavior. The observed data represented a set τ of trajectories as in equation (4.8)

$$\tau = \{(s_0, a_0), (s_1, a_1), \dots\dots\dots\dots, (s_T, a_T)\} \tag{4.8}$$

Several methods have been developed for IRL, including:

- Maximum Entropy IRL: This approach assumes that the observed behavior is optimal and that, among all possible behaviors, the expert's behavior maximizes the entropy of the policy distribution. The objective is to find the reward function that maximizes the likelihood of the observed trajectories under the maximum entropy principle, as in equation (4.9), [75].

$$R = \underset{R}{\operatorname{argmax}} \sum_{\tau \in K} log\ P(\tau|R) \tag{4.9}$$

- Feature-Based Linear IRL: This method assumes that the reward function is a linear combination of features, where the features extracted from states by mapping function $\emptyset(S): S \rightarrow [0,1]$ as in equation (4.10) [76].

$$R(S) = w^T \emptyset(S) \tag{4.10}$$

- Generative Adversarial Imitation Learning (GAIL) combines generative adversarial networks (GANs) with imitation learning to infer the reward function. A discriminator is trained to distinguish between expert $E_{\tau \sim \pi E}[\log Dc(\tau)]$ and agent

trajectories$E_{\tau\sim G}[log(1-Dc(\tau))]$, while the agent learns to produce trajectories that fool the discriminator, as in equation (4.11) [77].

$$\min_{G}\max_{Dc} E_{\tau\sim\pi E}[\log Dc(\tau)] + E_{\tau\sim G}[log(1-Dc(\tau))] \quad (4.11)$$

While Maximum Entropy IRL provides a principled way to handle uncertainty in reward inference, it needs help with the high-dimensional and complex interactions typical in SR. It also requires careful feature engineering, which can be challenging and time-consuming. On the other hand, Feature-Based Linear IRL relies on linear combinations of predefined features to infer the reward function. Although simpler and computationally less intensive, it might not effectively capture the non-linear and complex relationships in swarm behaviors. It also suffers from the limitation of requiring manual feature engineering, which might not be feasible for intricate swarm dynamics. GAIL is better suited for SRs due to its ability to handle high-dimensional data, robustness to complex behaviors, scalability, and adaptability to diverse environments. Its use of deep learning techniques allows for automatic feature extraction and learning from complex, coordinated behaviors within the swarm, making it an ideal choice for inferring reward structures and optimizing swarm performance.

4.2.2. Objective functions, reward functions, and collective behaviors

The objective function of swarm robots defines the swarm's overall goal. This could include area coverage, target tracking, formation control, or cooperative transportation. The objective function typically quantifies the swarm's performance in efficiency, accuracy, and robustness metrics. The reward function assigns a scalar value to each state or state-action pair, reflecting the immediate benefit of being in that state or performing that action. In the context of swarm robotics, the reward function should align with the swarm's objective function, guiding individual robots to take actions that collectively achieve the swarm's overall goal. It translates the high-level objectives into actionable feedback for individual robots. IRL helps in designing the reward function by observing expert swarm behavior. Instead of manually defining the reward structure, IRL can infer it from demonstrations of successful swarm operations.

For example, suppose a swarm of robots effectively covers an area in minimal time with no collisions. In that case, IRL can analyze the trajectories and actions to determine the implicit reward function being optimized. Once the reward function is inferred using IRL, it can be used to train new robots or improve existing ones using RL. The inferred reward function ensures the learned policies align with the swarm's objectives.

Generating collective behaviors in SRs is related to the nature of the specific task. For instance, SR systems deployed in search and rescue operations leverage algorithms that enhance area coverage and accelerate the detection of targets. In navigation-focused tasks, SR systems might use algorithms designed for pathfinding, allowing the robots to navigate complex terrains efficiently, avoiding obstacles, and reducing the time to reach the target by choosing the shortest path [78-81].

The task-specific design of SR is a field marked by complexity due to unpredictable interactions within individuals through swarm to obtain. The lack of a generalized method for crafting desired collective behaviors underscores the innovative research in this domain. Among these are bio-inspired algorithms [82-85], which mimic natural phenomena like the movement of bird flocks or ant foraging. These algorithms contribute to developing decentralized control systems, wherein each robot operates on simple rules derived from immediate surroundings and peer interactions. The modular design strategy also involves constructing robots with versatile modules that can be reconfigured according to the task requirements. This method enhances the adaptability and scalability of SR systems, permitting customization to various environments and challenges by adjusting the modules as needed [86],[87].

Evolutionary robotics is another method that employs evolutionary algorithms to refine robot control systems, thereby enabling their behaviors to advance and become more efficient over time. This approach parallels natural selection, allowing for the autonomous development of solutions optimally suited to their environments and tasks, with continuous enhancements as robots encounter new conditions [88]. Furthermore, machine learning techniques, particularly RL, offer a powerful mechanism for developing versatile SR systems capable of handling a broad spectrum of tasks. RL enables robots to be independent and learn by interactions with other robots and their environment, simplifying the design of collective behaviors into smaller, more manageable components that can dynamically adapt to new challenges. R is the crucial element in RL, which drives robots to optimize their actions for maximum cumulative rewards over time. R's design reflects the task's primary goals modeled as an objective function, which fosters the collective behavior of SR. By observing expert behavior, IRL offers a refined technique that teaches the underlying reward functions. Unlike conventional RL, where a policy is learned directly from a predefined reward setup, IRL gains insights into complex behaviors through demonstrations, bypassing the need to adjust the reward formula explicitly. This approach is particularly beneficial in SRs, where crafting explicit reward functions can be complex due to the intricate interactions and dynamics among the robots. IRL offers an alternative by deriving the reward function from expert demonstrations, thus eliminating the need for manual reward construction and providing a more structured learning trajectory than traditional methods. This allows robots to learn policies that match or surpass the expert's strategy based on the inferred rewards without necessitating deep mathematical knowledge or a detailed understanding of the operational dynamics. This approach reduces human bias and potential suboptimality [89].

Researchers have applied IRL in several contexts, such as using maximum entropy IRL to deduce reward functions from GPS data on pigeon flocks, simulating flocking behavior, and identifying leader-follower dynamics [90]. It has also been utilized in SRs for tasks like enhancing area coverage in search and rescue operations, where it leverages human expert demonstrations to train robots in optimal coverage strategies [91]. Some research combines IRL with automatic modular design to create control software for SRs based solely on these demonstrations, bypassing the need for explicitly defined rewards and objectives [92]. Despite challenges in handling high-dimensional, continuous state-action spaces, IRL has shown potential for generalizing across

various tasks and environments, crucial for developing adaptive and robust SR systems capable of operating in diverse scenarios. In this chapter, we have proposed a model that stands out from others in its ability to generalize across different conditions and automate reward design, proving effective in complex, continuous environments with continuous state-action spaces and enabling the generation of various collective behaviors like navigation and search tasks. In SRs, designing effective control strategies for collective behavior is challenging due to complex agent interactions and multiple objectives. Imitation Learning (IL) directly maps observed state-action pairs from expert demonstrations to policies, enabling agents to replicate behaviors without understanding underlying motivations. Supervised Reward Learning involves training a model to predict rewards from state-action pairs using labeled data, which requires explicit reward annotations and may not generalize well to new scenarios. Inverse Deep Reinforcement Learning (IDRL), on the other hand, infers the implicit reward function from expert behavior, allowing agents to learn the objectives driving the behavior and apply this understanding to novel situations. IDRL is particularly advantageous for collective behaviors in swarm robotics, as manually crafting reward functions for complex, multi-objective tasks are often infeasible. By uncovering the expert's implicit rewards, IDRL facilitates the emergence of desired collective behaviors without the need for explicit reward specification.

4.2.3. Proposed IRL-RL model

The proposed IRL-RL model is designed to infer rewards by demonstrating for different tasks. It is deployed for two tasks already solved in Chapter 3 by RL, with an explicit reward formula. So, we coped with two tasks: searching for green boxes by following the light that emerges from them. Second, the navigation task where robots move from initial positions to a target is illustrated as a yellow circle (Y), as shown in Figure 4.3 [93]. The environment is a 3×3 m^2 square area with four walls.

The parameters of mobile robots were set as in section 2.2.1.

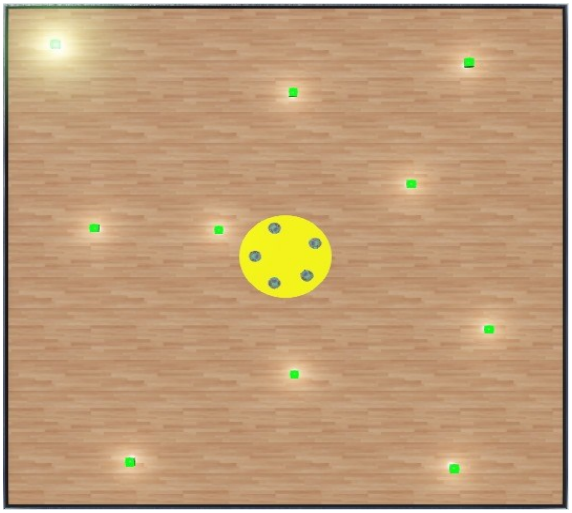

Figure 4.3. IRL-RL's environment.

Figure 4.4 illustrates the proposed IRL-RL model. It consists of two sections. RL is deployed to train the robots to generate the policy based on the reward inferred by IRL. The architecture of RL

and parameters are given in the section 2.2.3. In the IRL part, we proposed the following structure, The pseudo code 4.1 explains the steps of this model, :

- Data Loader: This component is a repository for data received from expert demonstrations and training sessions. Data from expert demonstrations is gathered using a pre-trained expert model, while training data is accumulated during the PPO training phase. The data comprises only state frames, which include flags but omit actions. These flags indicate task completion, such as locating a specific item in a search task or arriving at a designated point in navigation tasks. The model is equipped to handle both segmented and continuous state inputs. The functionality of these state types was explored in the results section. In segmented mode, sensor readings are first normalized and then categorized into five segments ranging from 0 to 1, each representing a different value.
- Feature Extractor: As illustrated in Table 2, this component details the types of data, and the operations applied to the data received from the data loader. The incoming data is in its raw form, where values from light sensors range between [0, 4095]. Meanwhile, the distance D spans from [0, 3] meters, and the angle θ varies from [−π, π] rad. The function ∅(s) outlined in Equations (4.12) and (4.13) transforms these raw states, S, into a feature vector that is more apt for input into the model.
- A shift function is implemented on the normalized states to derive values at *t−1*. These values facilitate the establishment of correlations between states, which is crucial for enhancing the *R* network's efficacy in determining the direction of state changes.

$$\emptyset(S): S \rightarrow [0,1] \tag{4.12}$$

$$\emptyset(S) = \frac{Max_{Output} - Min_{Output}}{Max_{Value} - Min_{Value}} . (S - Max_{Value}) + Max_{Output} \tag{4.13}$$

Table 4.1. Features Extractor Input and Output for Searching and Navigation Tasks.

Task	Input of features extractor (from the data loader)	Output of features extractor
Searching	$LS_0^{(t)}, LS_7^{(t)}, flag\ (Finding\ a\ box)$	$Normalized\ [\ LS_0^{(t-1)}, LS_0^{(t)}, LS_7^{(t-1)}, LS_7^{(t)}\ ],\ flag\ (Finding\ a\ box)$
Navigation	$D^{(t)}, \theta^{(t)}, flag\ (Reaching\ P)$	$Normalized\ \left[D^{(t-1)}, D^{(t)}, \theta^{(t-1)}, \theta^{(t)}\right],\ flag\ (Reaching\ P)$

- Reward Network: The reward neural network aims to estimate the underlying reward closely. This estimation is achieved by inputting the feature vector into the neural network, which outputs a scalar reward value. The network is structured with fully connected layers configured as (*length(feature−vectors) ×15×1FC*- ReLU activation function).
  In the scenarios described, the feature vectors are typically of length 5, as detailed in Table 4.1.

- Deep IRL: The backpropagation process in the reward network involves computing losses based on Equation (4.14), which ensures the reward neural network's weights are updated accordingly. The key loss function used here is the binary cross-entropy loss, which effectively differentiates between the rewards observed from experts and those generated during training. This loss function is used to distinguish expert demonstrations from learned policies. The first term maximizes the probability of expert rewards R_{expert}, while the second term minimizes the probability of learned policy rewards $R_{training}$. The sigmoid function ensures the outputs are in the range (0,1), making this loss similar to binary cross-entropy for classification.

$$loss = -\log\left(sigmoid\left(R_{expert}\right)\right) - log\left(1 - sigmoid\left(R_{training}\right)\right) \qquad (4.14)$$

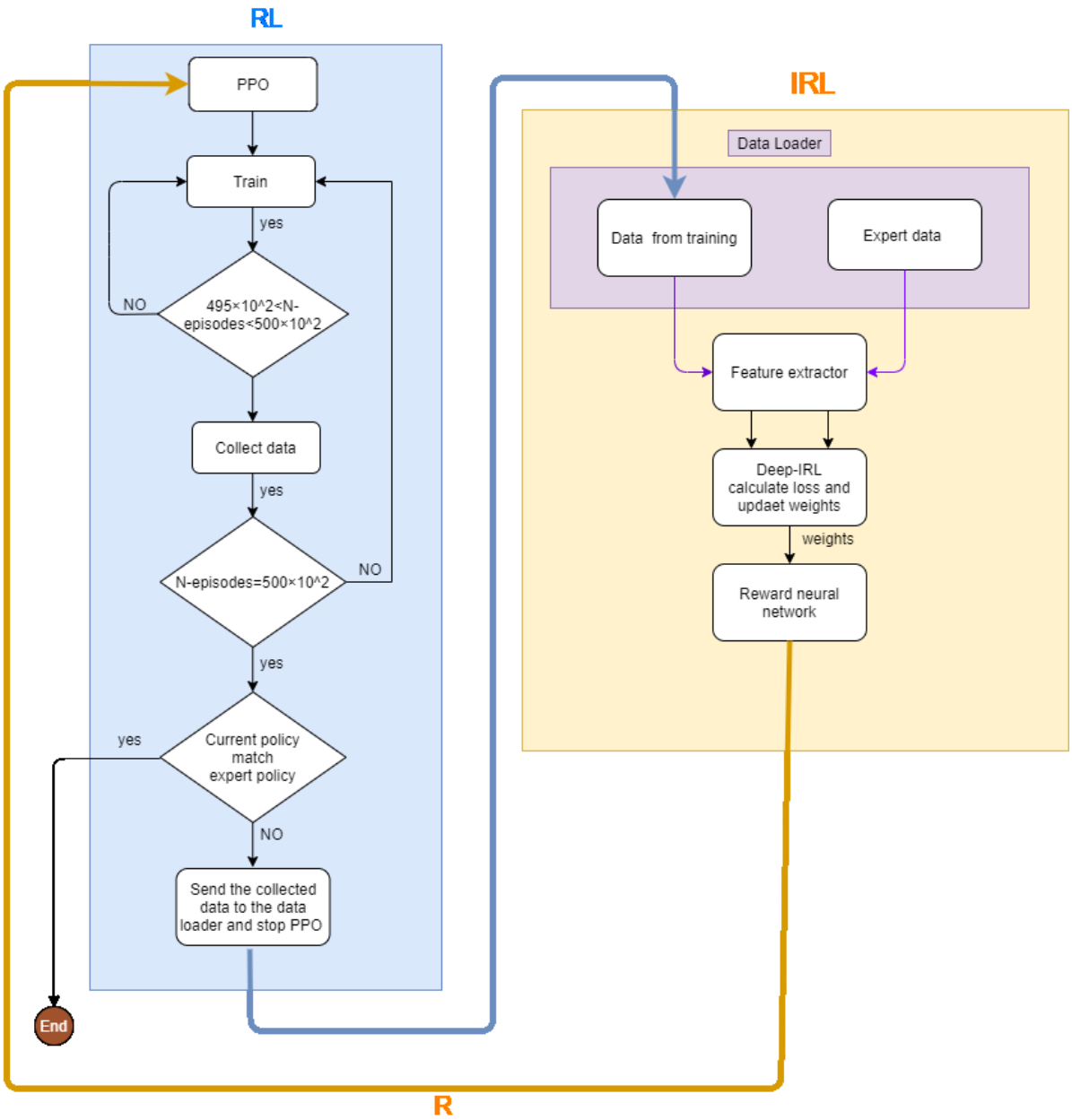

Figure 4.4. IRL-RL model.

Algorithm 1: Deep Inverse Reinforcement Learning (DIRL).

Step 1: Collect Expert Demonstrations
Collect expert state-action pairs: $D_{expert} = \{(s_e, a_e)\}$
Step 2: Initialize Reward Function: (length(feature−vectors) ×15×1 FC with ReLU).
Same hyperparameter of actor and critic of the PPO neural network.
Initialize reward neural network R_θ with weights w_0
Step 3: Train Initial Policy with RL
Train policy π_0 using RL with initial reward R_θ
Collect agent-generated data: $D_{agent} = \{(s_\pi, a_\pi)\}$
Step 4: Compute Predicted Rewards
Forward propagate through R_θ:
$R_e = R_\theta(s_e, a_e)$ # Predicted rewards for expert actions
$R_\pi = R_\theta(s_\pi, a_\pi)$ # Predicted rewards for agent actions
Step 5: Compute Loss Function
Compute loss $L(\theta)$ based on expert vs. agent rewards (Equation 4.14)
Step 6: Update Reward Function
Backpropagate loss and update reward weights: $w_1 = w_0 - \alpha * \nabla L(\theta)$
Generate new reward function $R_{\theta 1}$
Step 7: Train RL Agent with Updated Rewards
Train new policy π_1 using RL with updated $R_{\theta 1}$
Collect new agent-generated data: $D_{agent} = \{(s_{\pi 1}, a_{\pi 1})\}$
Step 8: Iterate Until Convergence
Repeat Steps 4-7 until policy π converges to optimal behavior
(The condition here is the number of iterations)
Return: Optimized policy π^* and learned reward function $R_{\theta *}$

4.2.4. Results and discussion

This model concentrated on investigating how rewards influence swarm behavior within a simulated setting, showcasing the capability of deep IRL to estimate the reward function without relying on complex mathematical procedures. We implemented this in two distinct tasks: searching for boxes using continuous RL with segmented features and navigating to a specific location identified as Y, with continuous RL but utilizing continuous features. The effectiveness of the swarm was assessed and validated by comparing the rewards from the IRL- RL model to those from a pre-trained model using expert RL, highlighting the proposed model's proficiency in guiding the swarm to accomplish the set tasks. Additionally, we analyzed the behaviors generated by the swarm related to the both models, emphasizing the importance of feature selection in accurately deriving the reward function. These features varied according to the defined problem and the objective function of the swarm system, as detailed in Table 4.1. It clarifies the distinct features selected for searching and navigation tasks. It also considers sensor readings at time steps t and t-1 to enable the R network to detect changes in light intensity for the searching task or alterations in distance for the navigation task. Using a deep neural network to represent R, coupled with a binary cross-entropy loss function, allows the model to manage continuous environments

effectively. Thus, in this models, we reconstructed the reward in both continuous and segmented modes to develop policies in RL that handle continuous state and action spaces.

a. Searching task

In this scenario, the reward mechanism is linked to variations in light intensity as detected by the robots' sensors. The reward increases when robots navigate towards areas with stronger light intensity and peaks when they locate the targeted boxes. Recovering the reward which leads to a successful behavior required three rounds as follows:
The reward neural network was initially set up with arbitrary weights labeled as ω0. At this stage, RL learned a stochastic policy π0. According to this policy, demonstrations were collected. The data loader then processed data gathered under this policy. Leveraging expert and newly collected data based on policy π0, the reward network is trained. This training adjusted the weights to ω1. Subsequently, the RL component, depicted in Figure 4.4, retrained to develop its updated policy π1 based on these adjusted weights. The cycle concluded with the robot learning policy π2, characterized by weights ω2, effectively accomplishing the desired task as depicted in Figure 4.5. The reward function in the expert-driven RL model, represented by a red line, optimizes the reward as the robot moves closer to the light source or box, maintaining high values upon arrival. Conversely, the reward deduced by the IRL-RL model, shown by a blue line, reflects the light intensity increases segmented, mirroring the RL model's behavior with distinct transitions due to the segmented mode of feature extraction. Therefore, the data is categorized into specific segments, with states from 0 to 0.2 indicating darker areas that receive uniform rewards. As illustrated in Figure 4.6, in the IRL-RL model, sensor readings LS_0 and LS_7 in the 0 to 0.2 range yield minimal rewards, which increase as the robot moves into the 0.2-0.4 segment. The reward significantly spikes in the 0.8-1 range, mentioning the robot's proximity to the box. The side-by-side visualization of the reward functions from the expert RL and IRL-RL model showcases their differences, with a darker blue denoting the latter. Notably, a consistent gradient pattern across both models indicates a direct correlation of rewards with increasing light intensity. This confirms that the robots have effectively learned to search for and locate boxes, validating the designed behavior. This demonstrates the IRL-RL model's capacity to replicate the decision-making strategy of the pre-trained robot.

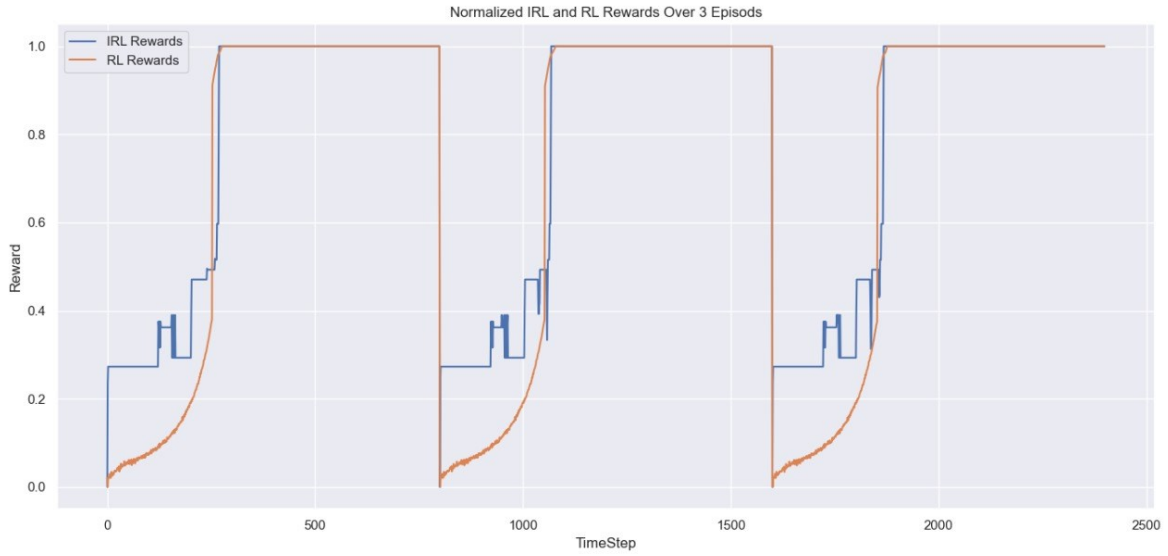

Figure 4.5. Normalized IRL and RL rewards over three episodes for the ω2-searching task.

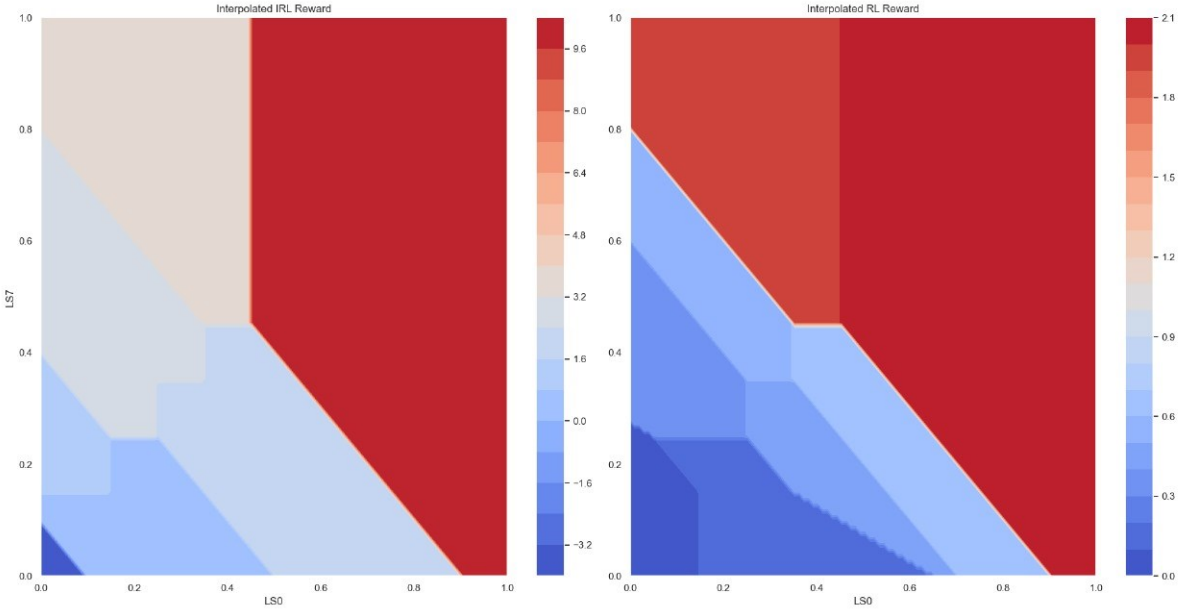

Figure 4.6. Heat map of the true reward (right) and the recovered reward (left) for the searching task.

The behavior of robots, as depicted in the bar chart of Figure 4.7, demonstrates the effectiveness of the IRL-RL model in comparison to a pre-trained expert RL model across a series of ten incremental tasks involving box collections. The IRL-RL model showcases a behavior pattern that allows robots to collect boxes sequentially round-triply. Notably, variations in the times taken to collect boxes indicate differing behaviors. This observation suggests that the IRL-RL model doesn't merely replicate the actions of the RL model; instead, it independently learns to accomplish the task by developing its unique behaviors. This indicates that IRL effectively masters the inferring of reward structures that lead to successful task completion rather than simply imitating actions.

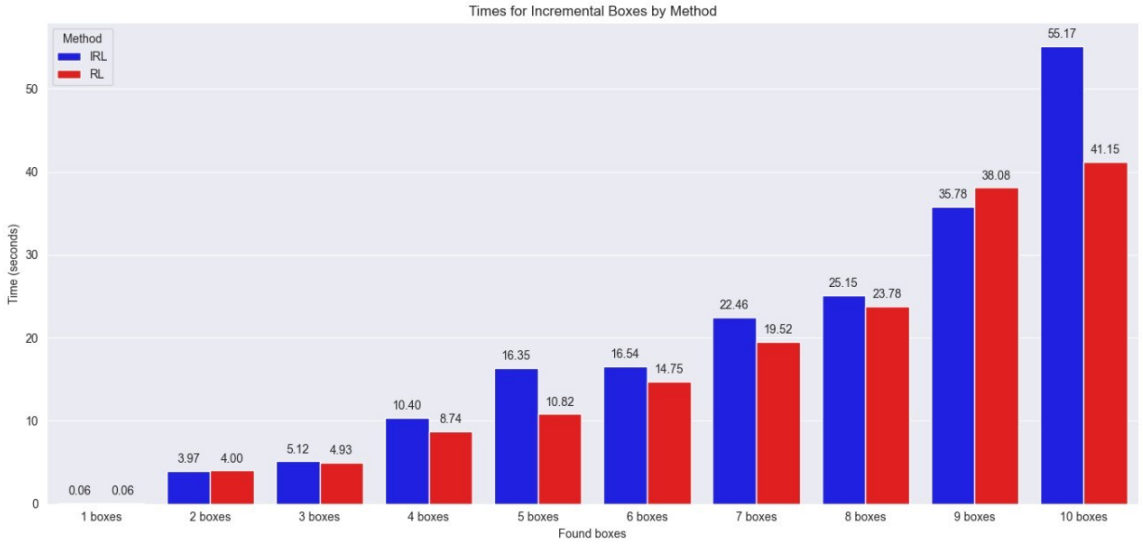

Figure 4.7. Swarm searching behavior by IRL-RL and traditional RL.

b. Navigation task

The identical procedure, encompassing the same number of iterations, was utilized for the navigation task. Initially, the reward neural network started with randomly assigned weights, designated as ω0, which generated a stochastic policy π0, initiating the RL model's learning trajectory. After the initial iteration, data gathered during this phase was employed to refine the reward network, adjusting the weights to ω1. This adjustment allowed the RL model to enhance its policy to π1. The cycle concluded with a second update, leading to the final weights ω2, which equipped the model to perform the required navigation tasks proficiently, as shown in Figure 4.8.

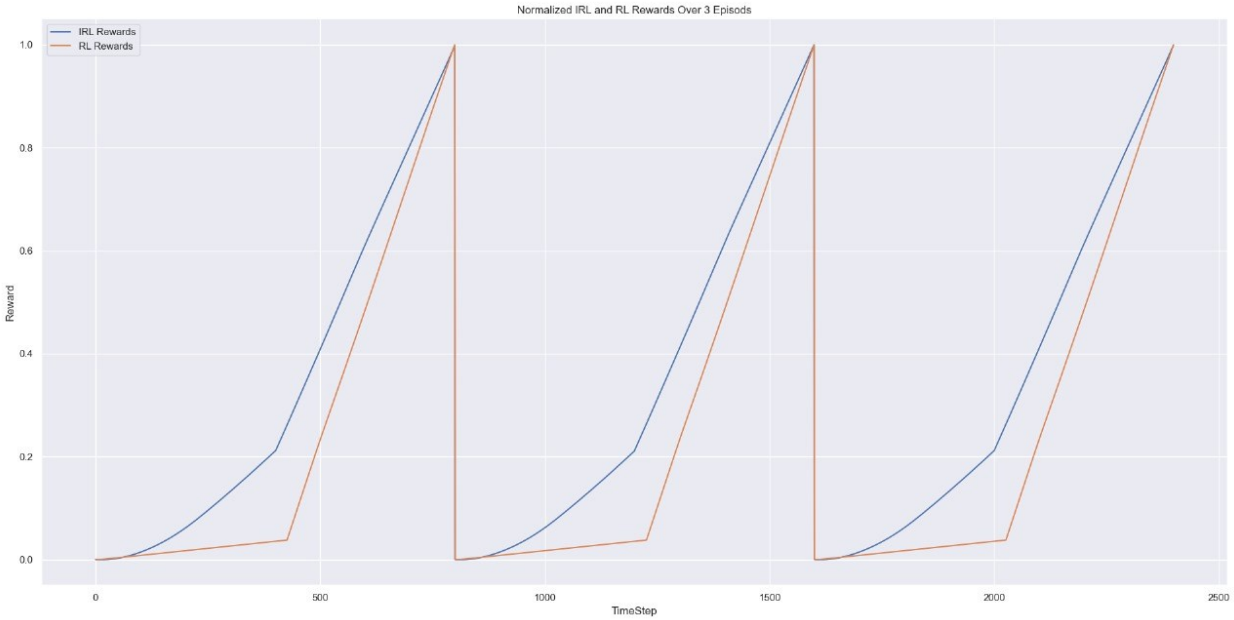

Figure 4.8. Normalized IRL and RL rewards over 3 episodes for ω2- Navigation task.

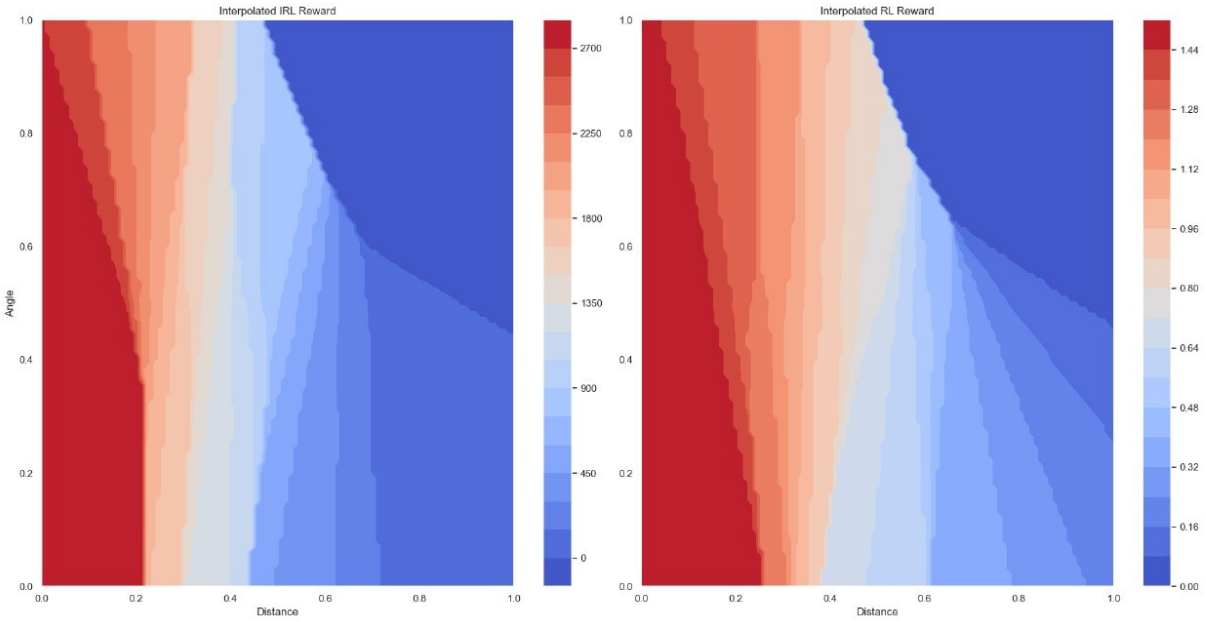

Figure 4.9. Heat map of the true reward (right) and the recovered reward (left) for the Navigation task.

Figure 4.9 displays the interpolated IRL and RL reward models, featuring a color gradient shifting from red to blue, which reflects changes in reward intensity based on the robot's orientation and proximity to the target. Higher rewards are indicated by red, associated with smaller angles and closer distances, signifying the robot's direct alignment and nearness to the target. Conversely, as the angle increases or the distance extends, the reward decreases, as demonstrated by the color transition to blue. Unlike the search task, there is no division into discrete state ranges. The continuity of the data facilitates a smoother visual gradient and more detailed adjustments of the reward to the robot's position and orientation to the target. The resemblance in the trajectories displayed in both charts in Figure 4.10 shows that IRL has effectively assimilated the data from RL, closely mimicking the expert RL's approach. This indicates a successful deployment of IRL, where the algorithm has adeptly deduced the strategies and decisions deemed optimal by RL.

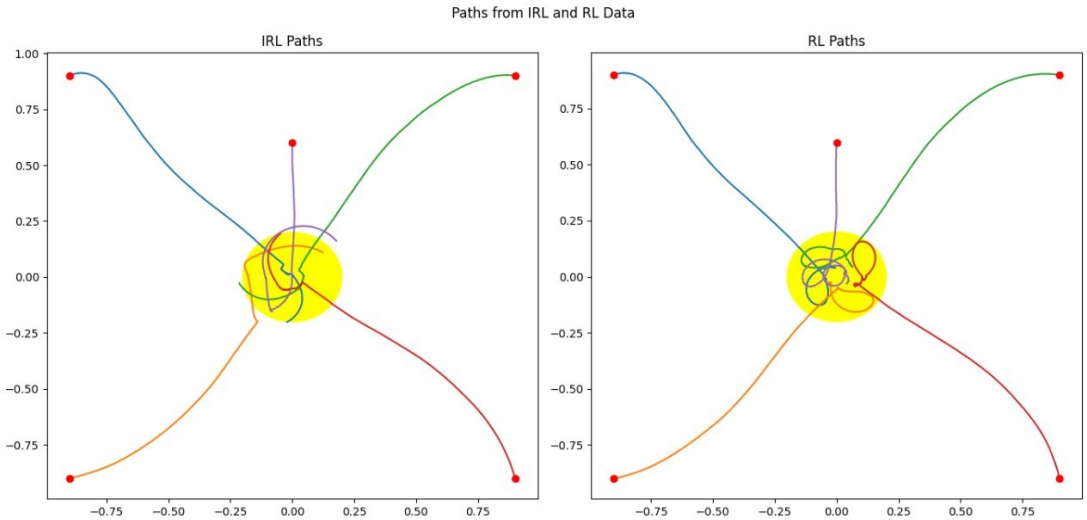

Figure 4.10. Robots' navigation paths.

4.3. Conclusion of Reward Methods in DRL for Swarm Robotics

This chapter demonstrates the crucial impact of well-structured reward methods on shaping SRs' behaviors and learning outcomes in various task environments. Applying diverse reward strategies such as sparse rewards, shaping rewards, and deep IRL has shown significant differences in how SRs adapt to and excel in their assigned tasks. In addition to integrating multi-objective rewards, curriculum learning and cooperative reward distribution within the reward systems can further refine the behavior and efficiency of swarms, especially in multi-task environments.

It highlighted how reward shaping facilitated more direct and efficient behaviors toward objectives, like reaching targets or navigating obstacles, compared to sparse rewards, which, while simplifying the reward design, required more rounds of learning to achieve similar outcomes. It introduces a relationship between penalty percentages and key performance metrics such as average time to reach the goal, number of collisions, and success rate. It showed that a balanced penalty rate, around 0.1 to 0.3, provided the best trade-off between rapid goal attainment and minimal collisions. This demonstrates that moderate penalties can significantly enhance safety and efficiency in navigation tasks.

Finally, the IRL-RL model that utilizes deep IRL to accurately infer the reward function from expert behavior demonstrations was introduced. Rather than directly learning behaviors, IRL aims to comprehend the underlying motivations for specific actions or strategies by estimating the rewards needed to accomplish tasks through generated behaviors. This approach eliminates the necessity for extensive manual adjustment of reward functions and enables more intuitive, demonstration-based learning. The proposed IRL-RL model can manage continuous state spaces and dynamic environments, addressing continuous RL challenges through a deep neural network to represent the reward function R. Additionally, it can recover the reward function using two types of data from the data loader: segmented and continuous features, catering to nuanced strategies. The model was evaluated in two tasks within a simulated swarm robotics environment: navigating to a predefined location and searching for specific items. It proved highly effective in inferring and adapting reward structures crucial for successfully directing autonomous robotic swarms to complete these tasks. Furthermore, The results underscore the model's generalization ability across various scenarios.

THESES – NEW SCIENTIFIC RESULTS

1. Swarm intelligence algorithms, particularly PSO and PPO, are widely applied in swarm robotics. While prior research has explored both methods individually, little attention has been given to a direct comparison of their impact on collective swarm behavior, adaptability, and coordination in decentralized robotics. Unlike studies that primarily integrate RL into PSO for parameter tuning and optimization, this research provides a comparative behavioral analysis of PSO and PPO, evaluating their individual strengths, limitations, and potential for structured hybridization. By examining their fundamental role in swarm formation, this study paves the way for more effective hierarchical, structured, and hybrid control strategies. Publications [k1] ,[k2].

2. This study presents a method for optimizing mobile robot navigation using DRL by enhancing the PPO algorithm with curriculum learning. The research demonstrates improved convergence efficiency and adaptability. A comparative analysis between the modified PPO, original PPO, and other algorithms highlights the superior performance of the curriculum-augmented PPO, particularly in handling complex, dynamic environments. Additionally, the study investigates swarm robot training, revealing that curriculum learning significantly enhances success rates, collision avoidance, and generalization capabilities in novel scenarios [k3] ,[k4].

3. It introduces a hybrid approach combining automatic design methods like DRL or PSO within a modular design to tackle the foraging problem in swarm robotics. The system, implemented in a 3D environment using Webots, involves 8 E-Puck robots equipped with light sensors to search for and transport dynamically moving resources. The modular architecture enhances system manageability and reduces computational demands, making it easier to address complex, non-static foraging tasks. The simulations show that the RL-based model outperforms PSO regarding task efficiency, resource collection, and adaptability to dynamic environments. RL-equipped robots demonstrate superior individual learning and autonomy, contributing to more effective collective swarm intelligence, while PSO relies more on the collective knowledge of the swarm [k5].

4. The study systematically examines how reward functions can be structured to guide robots in tasks such as efficient resource collection, adaptive navigation, and decentralized decision-making. A key aspect of this research is the balancing of penalties and rewards, ensuring that learning is neither hindered by excessive punishment nor misdirected by overly generous rewards, which could lead to suboptimal behaviors. A major contribution of this thesis is the introduction of a Deep Inverse Reinforcement Learning (RL-IRL) model designed to discover optimal reward structures for guiding swarm behavior in complex and unpredictable environments. Unlike traditional RL methods, which rely on manually defined rewards, IRL extracts implicit reward functions by learning from expert swarm demonstrations. This method is particularly effective in handling continuous state and action spaces, allowing the swarm to develop adaptive collective behaviors based on specific task objectives [k5] ,[k6] ,[k7] ,[k8].

ACKNOWLEDGMENTS

First and foremost, I would like to express my deepest gratitude to the University of Miskolc for allowing me to pursue my doctoral studies. I am profoundly thankful to my supervisor, Dr. Béla Kovács, for his unwavering support, guidance, and belief in my work. Your mentorship has been instrumental in helping me navigate the complexities of my research.

I would also like to extend my heartfelt thanks to Ali Hammoud for his collaboration and dedication in developing the RL-IRL and foraging models. Working alongside you has been an enriching experience that significantly contributed to the success of this research.

To my beloved parents, words cannot fully express the depth of my gratitude. To my father, whose wisdom and strength have always been my guiding light, thank you for your endless sacrifices and for believing in me, even when I doubted myself. To my mother, whose love and warmth have been a constant source of comfort, thank you for your unwavering support and always being there, no matter how far away I was.
To my dear sisters, Sawsan and Hanadi, you have been my pillars of strength and greatest cheerleaders. Your love, encouragement, and understanding have meant the world to me, and I am forever grateful to have you by my side. I want to express my heartfelt gratitude to my wonderful nieces, Zainab, Bushra, Rama, and Tala, for their endless joy and love and to Yhia and Samer for their constant support and encouragement.
To Hla, Assi, and Farah: thank you for your friendship, support, and the joy you bring into my life.

REFERENCES

[1] Cheraghi AR., Shahzad S., Graffi K. ”Past, present, and future of swarm robotics.” In Intelligent Systems and Applications: Proceedings of the 2021 Intelligent Systems Conference (IntelliSys), v. 3, pp. 190-233, 2022. Springer International Publishing. https://doi.org/10.1007/978-3-030-82199-9_13.

[2] Debie E., Kasmarik K., Garratt M. ” Swarm robotics: A Survey from a Multi-tasking Perspective.” ACM Computing Surveys,v. 56, no. 2, pp. 1-38, 2023. https://doi.org/10.1145/3611652.

[3] Dias PG., Silva MC., Rocha Filho GP., Vargas PA., Cota LP., Pessin G. ”Swarm robotics: A perspective on the latest reviewed concepts and applications.” Sensors. vol. 21, no. 6, 2021. https://doi.org/10.3390/s21062062

[4] Brambilla M., Ferrante E., Birattari M., Dorigo M.” Swarm robotics: a review from the swarm engineering perspective.” Swarm Intelligence. vol. 7, pp. 1-41, 2013. https://doi.org/10.1007/s11721-012-0075-2

[5] Olaronke, Iroju, Ikono R., Ishaya G., O. A. Ojerinde, and Olaleke J. ”A systematic review of swarm robots.” Drones, v. 7, no. 4, pp. 269, 2023. https://doi.org/10.3390/drones7040269

[6] Reynolds CW.” Flocks, herds, and schools: A distributed behavioral model.” In Proceedings of the 14th annual conference on Computer graphics and interactive techniques, pp. 25-34, 1987. https://doi.org/10.1145/37401.37406

[7] Kennedy J., Eberhart R. "Particle swarm optimization." In Proceedings of ICNN'95 - International Conference on Neural Networks, v. 4, pp. 1942-1948, 1995. IEEE. https://doi.org/10.1109/ICNN.1995.488968

[8] Dorigo M. "Ant colony optimization." Scholarpedia, v. 2, no. 3, pp. 1461, 2007. https://doi.org/10.4249/scholarpedia.1461.

[9] Czirók A., Vicsek T. "Collective behavior of interacting self-propelled particles." Physica A: Statistical Mechanics and its Applications, v. 281, no. 1-4, pp. 17-29, 2000. https://doi.org/10.1016/S0378-4371(00)00013-3

[10] Brooks R. "A robust layered control system for a mobile robot." IEEE Journal on Robotics and Automation, v. 2, no. 1, pp. 14-23, 1986. https://doi.org/10.1109/JRA.1986.1087032

[11] Devi K.V., Smitha B.S., Lakhanpal S., Kalra R., Sethi V.A., Thajil S.K. "A review: Swarm robotics: Cooperative control in multi-agent systems." In E3S Web of Conferences, v. 505, p. 03013, 2024. EDP Sciences. https://doi.org/10.1051/e3sconf/202450503013

[12] Nazarova A.V., Zhai M. "Distributed solution of problems in multi-agent robotic systems." Smart Electromechanical Systems: Group Interaction, pp. 107-124, 2019. https://doi.org/10.1007/978-3-319-99759-9_9

[13] Hüttenrauch M., Šošić A., Neumann G. "Deep reinforcement learning for swarm systems." Journal of Machine Learning Research, v. 20, no. 54, pp. 1-31, 2019.

[14] Orr J., Dutta A. "Multi-agent deep reinforcement learning for multi-robot applications: A survey." Sensors, v. 23, no. 7, pp. 3625, 2023. https://doi.org/10.3390/s23073625

[15] Bayındır L. "A review of swarm robotics tasks." Neurocomputing, v. 172, pp. 292-321, 2016.. https://doi.org/10.1016/j.neucom.2015.05.116

[16] Sutton R.S., Barto A.G. Reinforcement Learning: An Introduction. MIT Press, 2018.
[17] Watkins C.J.C.H., Dayan P. "Q-learning." Machine Learning, v. 8, no. 3-4, pp. 279-292, 1992. https://doi.org/10.1007/BF00992698
[18] Rummery G.A., Niranjan M. "On-line Q-learning using connectionist systems." Technical Report CUED/F-INFENG/TR 166, University of Cambridge, Department of Engineering, 1994.
[19] S.Y., Liu Y., Wang G., Zhang H. "Deep learning for plant identification in natural environment." Computational Intelligence and Neuroscience, v. 2017, pp. 1-10, 2017. https://doi.org/10.1155/2017/7361042
[20] Haarnoja T., Zhou A., Abbeel P., Levine S. "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor." Proceedings of the 35th International Conference on Machine Learning (ICML), v. 80, pp. 1861-1870, 2018. https://doi.org/10.48550/arXiv.1801.01290
[21] S.J., Levine S., Abbeel P., Jordan M., Moritz P. "Trust region policy optimization." In International Conference on Machine Learning, pp. 1889-1897, PMLR, 2015.. https://doi.org/10.48550/arXiv.1502.05477
[22] C.N. "Deep deterministic policy gradient for urban traffic light control." arXiv preprint, arXiv:1703.09035, 2017.https://doi.org/10.48550/arXiv.1703.09035
[23] S.J., W.F., Dhariwal P., Radford A., Klimov O. "Proximal policy optimization algorithms." arXiv preprint, arXiv:1707.06347, 2017. https://doi.org/10.48550/arXiv.1707.06347
[24] M.W., Park B., Nengroo S.H., Kim T., Har D. "Path planning of cleaning robot with reinforcement learning." In 2022 IEEE International Symposium on Robotic and Sensors Environments (ROSE), pp. 1-7, IEEE, 2022. https://doi.org/10.48550/arXiv.2208.08211
[25] Y.L., Bi J., Yuan H. "Dynamic Path Planning for Mobile Robots with Deep Reinforcement Learning." IFAC-PapersOnLine, v. 55, no. 11, pp. 19-24, 2022. https://doi.org/10.1016/j.ifacol.2022.08.042
[26] Y.X., Wang P., Zhang Z. "Learning-based end-to-end path planning for lunar rovers with safety constraints." Sensors, v. 21, no. 3, p. 796, 2021. https://doi.org/10.3390/s21030796
[27] J.X., Wang Z. "Proximal policy optimization based dynamic path planning algorithm for mobile robots." Electronics Letters, v. 58, no. 1, pp. 13-15, 2022. https://doi.org/10.1049/ell2.12342
[28] Tan Z., Karaköse M. "Proximal policy based deep reinforcement learning approach for swarm robots." In 2021 Zooming Innovation in Consumer Technologies Conference (ZINC), pp. 166-170, IEEE, 2021. https://doi.org/10.1109/ICRA.2021.9562035
[29] Wu Z., Yu C., Ye D., Zhang J., Zhuo H.H. "Coordinated proximal policy optimization." Advances in Neural Information Processing Systems, v. 34, pp. 26437-26448, 2021.
[30] Sadhukhan P., Selmic R.R. "Multi-agent formation control with obstacle avoidance using proximal policy optimization." In 2021 IEEE International Conference on Systems, Man, and Cybernetics (SMC), pp. 2694-2699, IEEE, 2021. https://doi.org/10.1109/SMC52423.2021.9658635
[31] Khaldi B., Cherif F. "An overview of swarm robotics: Swarm intelligence applied to multi-robotics." International Journal of Computer Applications, v. 126, no. 2, 2015.
[32] Blum C., Groß R. "Swarm intelligence in optimization and robotics." Springer Handbook of Computational Intelligence, pp. 1291-1309, 2015. https://doi.org/10.1007/978-3-662-43505-2_66

[33] Francesca G., Birattari M. "Automatic design of robot swarms: achievements and challenges." Frontiers in Robotics and AI, v. 3, p. 29, 2016. https://doi.org/10.3389/frobt.2016.00029

[34] Iskandar A., Kovács B. "A survey on automatic design methods for swarm robotics systems." Carpathian Journal of Electronic & Computer Engineering, v. 14, no. 2, 2021. https://doi.org/10.2478/cjece-2021-0006

[35] Mehta D., Sharma A., Ravichandran R. "A review on robotic swarm optimization techniques." Authorea Preprints, 2023. https://doi.org/10.36227/techrxiv.23675199.v1

[36] Ab Aziz N.A., Ibrahim Z. "Asynchronous particle swarm optimization for swarm robotics." Procedia Engineering, v. 41, pp. 951-957, 2012. doi:https://doi.org/10.1016/j.proeng.2012.07.268

[37] Rossides G., Metcalfe B., Hunter A. "Particle swarm optimization—an adaptation for the control of robotic swarms." Robotics, v. 10, no. 2, p. 58, 2021. https://doi.org/10.3390/robotics10020058

[38] Hamami M.G.M., Ismail Z.H. "A systematic review on particle swarm optimization towards target search in the swarm robotics domain." Archives of Computational Methods in Engineering, pp. 1-20, 2022. https://doi.org/10.1007/s11831-022-09819-3

[39] Blais M.-A., Akhloufi M.A. "Reinforcement learning for swarm robotics: An overview of applications, algorithms and simulators." Cognitive Robotics, 2023. https://doi.org/10.1016/j.cogr.2023.07.004

[40] Jin B., Liang Y., Han Z., Ohkura K. "Generating collective foraging behavior for robotic swarm using deep reinforcement learning." Artificial Life and Robotics, v. 25, pp. 588-595, 2020. https://doi.org/10.1007/s10015-020- 00642-2

[41] Jin B., Liang Y., Han Z., Hiraga M., Ohkura K. "A hierarchical training method of generating collective foraging behavior for a robotic swarm." Artificial Life and Robotics, pp. 1-5, Feb 2022. https://doi.org/10.1007/s10015-021-00714-x

[42] Wei Y., Nie X., Hiraga M., Ohkura K., Car Z. "Developing end-to-end control policies for robotic swarms using deep Q-learning." Journal of Advanced Computational Intelligence and Intelligent Informatics, v. 23, no. 5, pp. 920-927, 2019. https://doi.org/10.20965/jaciii.2019.p0920.

[43] Garaffa L.C., Basso M., Konzen A.A., de Freitas E.P. "Reinforcement learning for mobile robotics exploration: A survey." IEEE Transactions on Neural Networks and Learning Systems, 2021. https://doi.org/10.1109/TNNLS.2021.3124466.

[44] Di Mario E., Talebpour Z., Martinoli A. "A comparison of PSO and reinforcement learning for multi-robot obstacle avoidance." In 2013 IEEE Congress on Evolutionary Computation, pp. 149-156, IEEE, 2013. https://doi.org/10.1109/CEC.2013.6557565

[45] Fan J., Hu M., Chu X., Yang D. "A comparison analysis of swarm intelligence algorithms for robot swarm learning." In 2017 Winter Simulation Conference (WSC), pp. 3042-3053, IEEE, 2017. https://doi.org/10.1109/WSC.2017.8248025.

[46] Klein L., Zelinka I., Seidl D. "Optimizing parameters in swarm intelligence using reinforcement learning: An application of Proximal Policy Optimization to the iSOMA algorithm." Swarm and Evolutionary Computation, v. 85, p. 101487, 2024. https://doi.org/10.1016/j.swevo.2024.101487

[47] Wang F., Wang X., Sun S. "A reinforcement learning level-based particle swarm optimization algorithm for large-scale optimization." Information Sciences, v. 602, pp. 298-312, 2022. https://doi.org/10.1016/j.ins.2022.04.053

[48] Gad A.G. "Particle swarm optimization algorithm and its applications: A systematic review." Archives of Computational Methods in Engineering, v. 29, no. 5, pp. 2531-2561, 2022. https://doi.org/10.1007/s11831-021-09694-4

[49] Niknam T., Amiri B. "An efficient hybrid approach based on PSO, ACO, and k-means for cluster analysis." Applied Soft Computing, v. 10, no. 1, pp. 183-197, 2010. https://doi.org/10.1016/j.asoc.2009.07.001

[50] Janson S., Middendorf M. "A hierarchical particle swarm optimizer for dynamic optimization problems." Applications of Evolutionary Computing: EvoWorkshops 2004, EvoBIO, EvoCOMNET, EvoHOT, EvoISAP, EvoMUSART, and EvoSTOC, Coimbra, Portugal, April 5-7, 2004. Proceedings, pp. 513-524, 2004. https://doi.org/10.1007/978-3-540-24653-4_52

[51] Iskandar A., Hammoud A., Kovács B. " Swarm Robotics Navigation Task: A Comparative Study of Reinforcement Learning and Particle Swarm Optimization Methodologies " Mekhatronika, Avtomatizatsiya, Upravlenie, v. 25, no. 9, pp. 471-478. https://doi.org/10.17587/mau.25.471-478.

[52] Michel O. "Cyberbotics Ltd. Webots—: professional mobile robot simulation." International Journal of Advanced Robotic Systems, v. 1, no. 1, p. 5, 2004. doi:https://doi.org/10.5772/5618.

[53] Kirtas M., Tsampazis K., Passalis N., Tefas A. "Deepbots: A Webots-based deep reinforcement learning framework for robotics." In Artificial Intelligence Applications and Innovations: 16th IFIP WG 12.5 International Conference, AIAI 2020, Greece, Proceedings, Part II, v. 16, pp. 64-75, Springer, 2020. https://doi.org/10.1007/978-3-030-49186-4 6.

[54] Mondada F., Bonani M., Raemy X., Pugh J., Cianci C., Klaptocz A., Magnenat S., Zufferey J.C., Floreano D., Martinoli A. "The e-puck, a robot designed for education in engineering." In Proceedings of the 9th Conference on Autonomous Robot Systems and Competitions, v. 1, no. 1, pp. 59-65, IPCB: Instituto Politécnico de Castelo Branco, 2009.

[55] Aznar F., Pujol M., Rizo R. "Learning a swarm foraging behavior with microscopic fuzzy controllers using deep reinforcement learning." Applied Sciences, v. 11, no. 6, p. 2856, 2021. https://doi.org/10.3390/app11062856

[56] Löffler R.C., Panizon E., Bechinger C. "Collective foraging of active particles trained by reinforcement learning." Scientific Reports, v. 13, no. 1, p. 17055, 2023. https://doi.org/10.1038/s41598-023-44268-3

[57] Iskandar A., Kovács B. "Curriculum learning for deep reinforcement learning in swarm robotic navigation task." Multidiszciplináris Tudományok, v. 13, no. 3, pp. 175-187, 2023. https://doi.org/10.35925/j.multi.2023.3.18

[58] Altshuler Y. "Recent developments in the theory and applicability of swarm search." Entropy, v. 25, no. 5, p. 710, 2023. https://doi.org/10.3390/e25050710

[59] Lee W., Vaughan N., Kim D. "Task allocation into a foraging task with a series of subtasks in swarm robotic system." IEEE Access, v. 8, pp. 107549-107561, 2020. https://doi.org/10.1109/ACCESS.2020.2999538

[60] Na S., Rouček T., Ulrich J., Pikman J., Krajník T., Lennox B., Arvin F. "Federated reinforcement learning for collective navigation of robotic swarms." IEEE Transactions on

Cognitive and Developmental Systems, v. 15, no. 4, pp. 2122-2131, 2023. https://doi.org/10.1109/TCDS.2023.3239815

[61] Adams S., Jarne Ornia D., Mazo Jr M. "A self-guided approach for navigation in a minimalistic foraging robotic swarm." Autonomous Robots, v. 47, no. 7, pp. 905-920,2023. https://doi.org/10.1007/s10514-023-10102-y

[62] Lee K.M., Kong F., Cannizzaro R., Palmer J.L., Johnson D., Yoo C., Fitch R. "An upper confidence bound for simultaneous exploration and exploitation in heterogeneous multi-robot systems." In 2021 IEEE International Conference on Robotics and Automation (ICRA), pp. 8685-8691, IEEE, 2021. https://doi.org/10.1109/ICRA48506.2021.9560822

[63] Talamali M.S., Bose T., Haire M., Xu X., Marshall J.A., Reina A. "Sophisticated collective foraging with minimalist agents: A swarm robotics test." Swarm Intelligence, v. 14, no. 1, pp. 25-56, Mar 2020. https://doi.org/10.1007/s11721-019-00176-9

[64] Wang X., Guo H. "Mobility-aware computation offloading for swarm robotics using deep reinforcement learning." In 2021 IEEE 18th Annual Consumer Communications & Networking Conference (CCNC), pp. 1-4, IEEE, 2021. https://doi.org/10.1109/CCNC49032.2021.9369456

[65] Soviany P., Ionescu R.T., Rota P., Sebe N. "Curriculum learning: A survey." International Journal of Computer Vision, v. 130, no. 6, pp. 1526-1565, Jun 2022. https://doi.org/10.1007/s11263-022-01611-x

[66] Ghebrechristos H., Alaghband G. "Deep curriculum learning optimization." SN Computer Science, v. 1, no. 5, p. 245, Sep 2020. https://doi.org/10.1007/s42979-020-00251-7

[67] B.K., Chakravarty P., Shrivastava S. "An A* curriculum approach to reinforcement learning for RGBD indoor robot navigation." arXiv preprint, arXiv:2101.01774, 2021. https://doi.org/10.48550/arXiv.2101.01774

[68] Jang P., Jang, S., Shin Y. "Indoor path planning for an unmanned aerial vehicle via curriculum learning." 2021 21st International Conference on Control, Automation and Systems (ICCAS). IEEE, 2021. https://doi.org/10.1109/ICTC55196.2022.9952572

[69] Sun M., Yang Z., Dai X., Nian X., Wang H., Xiong H. "Deep reinforcement learning based on curriculum learning for drone swarm area defense." International Conference on Autonomous Unmanned Systems, pp. 1119-1128, 2022. [Springer]. https://doi.org/10.1007/978-981-99-0479-2_101

[70] Hussein A., Petraki E., Elsawah S., Abbass H. A. "Autonomous swarm shepherding using curriculum-based reinforcement learning." Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems, pp. 633-641, 2022.

[71] Iskandar A., Kovács B. "Investigating the impact of curriculum learning on reinforcement learning for improved navigational capabilities in mobile robots." Inteligencia Artificial, v. 27, no. 73, pp. 163-176, Mar 2024. https://doi.org/10.4114/intartif.vol27iss73pp163-176

[72] Hammoud A., Iskandar A., Kovács B. " Dynamic foraging in swarm robotics: a hybrid approach with modular design and deep reinforcement learning intelligence" Informatics and automation, v. 24, pp. 51, 2025. https://doi.org/10.15622/ia.24.1.3

[73] A. Iskandar, B. Kovács, "Analysis of the effects of reward structures in deep reinforcement learning on the path planning of mobile robots." in 5th international black sea modern scientific research congress, Rize, Turkiye. pp. 758, 2023.

[74] Iskandar A., Rostum H.M., Kovács B. "Using deep reinforcement learning to solve a navigation problem for a swarm robotics system." In 2023 24th International Carpathian

Control Conference (ICCC), pp. 185-189, IEEE, 2023. https://doi.org/10.1109/ICCC57093.2023.10178888

[75] Ziebart B.D., Maas A.L., Bagnell J.A., Dey A.K. "Maximum entropy inverse reinforcement learning." In AAAI, v. 8, pp. 1433-1438, 2008.

[76] Ng A.Y., Russell S. "Algorithms for inverse reinforcement learning." In ICML, v. 1, no. 2, p. 2, Jun 2000.

[77] Ho J., Ermon S. "Generative adversarial imitation learning." Advances in Neural Information Processing Systems, v. 29, 2016.

[78] Nauta J., Van Havermaet S., Simoens P., Khaluf Y. "Enhanced foraging in robot swarms using collective Lévy walks." In 24th European Conference on Artificial Intelligence (ECAI), v. 325, pp. 171-178, IOS, 2020. https://doi.org/10.3233/FAIA200090.

[79] Misir O., Gökrem L. "Flocking-based self-organized aggregation behavior method for swarm robotics." Iranian Journal of Science and Technology, Transactions of Electrical Engineering, v. 45, no. 4, pp. 1427-1444, 2021. https://doi.org/10.1007/s40998-021-00442-9

[80] Sadeghi A., Raoufi M., Turgut A.E. "A self-adaptive landmark-based aggregation method for robot swarms." Adaptive Behavior, v. 30, no. 3, pp. 223-236, 2022. https://doi.org/10.1177/1059712320985543

[81] Berlinger F., Gauci M., Nagpal R. "Implicit coordination for 3D underwater collective behaviors in a fish-inspired robot swarm." Science Robotics, v. 6, no. 50, eabd8668, 2021. https://doi.org/10.1126/scirobotics.abd8668.

[82] Zhang J., Lu Y., Che L., Zhou M. "Moving-distance-minimized PSO for mobile robot swarm." IEEE Transactions on Cybernetics, v. 52, no. 9, pp. 9871-9881, 2021. https://doi.org/10.1109/TCYB.2021.3079346.

[83] Parhi D.R., Sahu C., Kumar P.B. "Navigation of multiple humanoid robots using hybrid adaptive swarm-adaptive ant colony optimisation technique." Computer Animation and Virtual Worlds, v. 29, no. 2, 2018. https://doi.org/10.1002/cav.1802.

[84] Jiang L., Mo H., Tian P. "An adaptive decentralized control strategy for deployment and aggregation of swarm robots based on bacterial chemotaxis." Applied Intelligence, v. 53, no. 10, pp. 13018-13036, 2023. https://doi.org/10.1007/s10489-022-04128-5

[85] Hu C., Arvin F., Bellotto N., Yue S., Li H. "Swarm neuro-robots with the bio-inspired environmental perception." Frontiers in Neurorobotics, v. 18, p. 1386178, 2024. https://doi.org/10.3389/fnbot.2024.1386178.

[86] Hasselmann K., Ligot A., Birattari M. "Automatic modular design of robot swarms based on repertoires of behaviors generated via novelty search." Swarm and Evolutionary Computation, v. 83, p. 101395, 2023. https://doi.org/10.1016/j.swevo.2023.101395.

[87] Birattari M., Ligot A., Francesca G. "AutoMoDe: a modular approach to the automatic off-line design and fine-tuning of control software for robot swarms." In Automated Design of Machine Learning and Search Algorithms, pp. 73-90, 2021. https://doi.org/10.1007/978-3-030-72069-8_5

[88] Stolfi D.H., Danoy G. "Evolutionary swarm formation: From simulations to real-world robots." Engineering Applications of Artificial Intelligence, v. 128, p. 107501, 2024. https://doi.org/10.1016/j.engappai.2023.107501

[89] Arora S., Doshi P. "A survey of inverse reinforcement learning: Challenges, methods and progress." Artificial Intelligence, v. 297, p. 103500, 2021. https://doi.org/10.1016/j.artint.2021.103500

[90] Pinsler R., Maag M., Arenz O., Neumann G. "Inverse reinforcement learning of bird flocking behavior." In ICRA Swarms Workshop, 2018.

[91] Chen M., Zhang P. "Area coverage for swarm robots via inverse reinforcement learning." Available at SSRN 4592186. http://dx.doi.org/10.2139/ssrn.4592186

[92] Gharbi I., Kuckling J., Ramos D.G., Birattari M. "Show me what you want: Inverse reinforcement learning to automatically design robot swarms by demonstration." In 2023 IEEE International Conference on Robotics and Automation (ICRA), pp. 5063-5070, IEEE, 2023. https://doi.org/10.1109/ICRA48891.2023.10160947.

[93] Iskandar A., Hammoud A., Kovács B. " Implicit understanding: decoding swarm behaviors in robots through deep inverse reinforcement learning" Informatics and automation, v.23,p. 1485,2024. https://doi.org/10.15622/ia.23.5.8

LIST OF PUBLICATIONS RELATED TO THE TOPIC OF THE RESEARCH FIELD

(k1) Iskandar A., Kovács B. "A survey on automatic design methods for swarm robotics systems." Carpathian Journal of Electronic & Computer Engineering, v. 14, no. 2, 2021. https://doi.org/10.2478/cjece-2021-0006

(k2) Iskandar A., Hammoud A., Kovács B. " Swarm Robotics Navigation Task: A Comparative Study of Reinforcement Learning and Particle Swarm Optimization Methodologies " Mekhatronika, Avtomatizatsiya, Upravlenie. v. 25, no. 9, pp. 471-478. https://doi.org/10.17587/mau.25.471-478.(Scoups, Q3)

(k3) Iskandar A., Kovács B. "Curriculum learning for deep reinforcement learning in swarm robotic navigation task." Multidiszciplináris Tudományok, v. 13, no. 3, pp. 175-187, 2023. https://doi.org/10.35925/j.multi.2023.3.18.

(k4) Iskandar A., Kovács B. "Investigating the impact of curriculum learning on reinforcement learning for improved navigational capabilities in mobile robots." Inteligencia Artificial, v. 27, no. 73, pp. 163-176, Mar 2024. https://doi.org/10.4114/intartif.vol27iss73pp163-176. (Scoups, Q4)

(k5) Hammoud A., Iskandar A., Kovács B. " Dynamic foraging in swarm robotics: a hybrid approach with modular design and deep reinforcement learning intelligence" Informatics and automation, v. 24, pp. 51, 2025. https://doi.org/10.15622/ia.24.1.3 (Scoups, Q4).

(k6) A. Iskandar, B. Kovács, "Analysis of the effects of reward structures in deep reinforcement learning on the path planning of mobile robots." in 5th international black sea modern scientific research congress, Rize, Turkiye. pp. 758, 2023.

(k7) Iskandar A., Rostum H.M., Kovács B. "Using deep reinforcement learning to solve a navigation problem for a swarm robotics system." In 2023 24th International Carpathian Control Conference (ICCC), pp. 185-189, IEEE, 2023. https://doi.org/10.1109/ICCC57093.2023.10178888.(Scoups).

(k8) Iskandar A., Hammoud A., Kovács B. " Implicit understanding: decoding swarm behaviors in robots through deep inverse reinforcement learning" Informatics and automation, v.23, p. 1485,2024. https://doi.org/10.15622/ia.23.5.8 (Scoups, Q4)