applied sciences

MDPI

Article

Learning a Swarm Foraging Behavior with Microscopic Fuzzy Controllers Using Deep Reinforcement Learning

Fidel Aznar *,†, Mar Pujol † and Ramón Rizo †

Department of Computer Science and Artificial Intelligence, University of Alicante, 03690 San Vicente del Raspeig, Alicante, Spain; mar.pujol@ua.es (M.P.); ramon.rizo@ua.es (R.R.)
* Correspondence: fidel@ua.es
† These authors contributed equally to this work.

Abstract: This article presents a macroscopic swarm foraging behavior obtained using deep reinforcement learning. The selected behavior is a complex task in which a group of simple agents must be directed towards an object to move it to a target position without the use of special gripping mechanisms, using only their own bodies. Our system has been designed to use and combine basic fuzzy behaviors to control obstacle avoidance and the low-level rendezvous processes needed for the foraging task. We use a realistically modeled swarm based on differential robots equipped with light detection and ranging (LiDAR) sensors. It is important to highlight that the obtained macroscopic behavior, in contrast to that of end-to-end systems, combines existing microscopic tasks, which allows us to apply these learning techniques even with the dimensionality and complexity of the problem in a realistic robotic swarm system. The presented behavior is capable of correctly developing the macroscopic foraging task in a robust and scalable way, even in situations that have not been seen in the training phase. An exhaustive analysis of the obtained behavior is carried out, where both the movement of the swarm while performing the task and the swarm scalability are analyzed.

Keywords: swarm robotics; foraging behavior; fuzzy controllers; deep reinforcement learning

check for updates

Citation: Aznar, F.; Pujol, M.; Rizo, R. Learning a Swarm Foraging Behavior with Microscopic Fuzzy Controllers Using Deep Reinforcement Learning. *Appl. Sci.* **2021**, *11*, 2856. https://doi.org/10.3390/app11062856

Academic Editors: Álvaro Gutiérrez and Giancarlo Mauri

Received: 8 February 2021
Accepted: 19 March 2021
Published: 23 March 2021

1. Introduction and State of the Art

Swarm robotics aims to produce robust, scalable and flexible self-organizing behaviors through local interactions among a large number of simple robots. In swarm robotics systems, complex swarm behaviors emerge from simple interaction rules. However, designing relatively simple individual rules to produce a large set of complex swarm behaviors is extremely difficult. Programming emerging swarm behaviors still has no formal or precise architecture in swarm robotics [1].

In swarm robotics, foraging is important for several reasons. It is a metaphor for a broad class of problems integrating exploration, navigation, object identification, manipulation and transport [2]. Swarm foraging behaviors are part of the set of functions of swarm robotics that are inspired by social insects. As in a foraging ecosystem, robots search for and collect food items in a shared environment [1]. Foraging can be described as a search for provisions (food). Several computational problems can be viewed as foraging problems; for example, searching on the web, routing in a network, path planning and so on [3]. The foraging task is one of the widely used testbeds in the study of swarm robotics, as it has the closest ties to biological swarms [2].

This task is challenging because the dimensionality of the problem grows exponentially with the number of agents and the agents can only partially observe a global state. Therefore, it is difficult to model the agents and to develop a control method based on the model. A promising approach to these complicated systems could be reinforcement learning, in which agents learn through trial and error, because reinforcement learning does not require prior knowledge of the environment [4].

However, we find that few previous works have attempted to solve the foraging problem using reinforcement learning (RL), among which we highlight [2,3,5]. We also see that many of these papers addressed low noise or simple environments or even modeled unrealistic robots.

Since the state space in robot navigation is large, the learning time is long. Moreover, at the beginning of the training phase, a robot does not have any knowledge; thus, the number of failures (punishments) will be high. Applying supervised learning to robot navigation faces serious challenges such as inconsistent and noisy data, difficulties in gathering the training data and many errors in the training data. RL algorithms are time consuming and suffer from a high failure rate in the training phase [5,6].

Given the special features of swarm systems and the difficulties faced by RL algorithms in dealing with them, we propose a hybrid strategy that uses RL to integrate basic microscopic behaviors. This strategy is not new and is often defined in this kind of system. As noted in [7], a swarm robotic system design should be based on the number of abstraction layers.

Thus, we propose the use of several behaviors at the robot level (microscopic behaviors) based on fuzzy logic. Fuzzy controllers are efficient and interpretable system controllers for continuous state and action spaces. The problems addressed by classical fuzzy control theory—i.e., stability, fault detection and robustness—make them well suited for serving as low-level system controllers [8]. Furthermore, the basic tasks required by robot agents (obstacle avoidance and rendezvous) are easy to model and test with this type of system.

Once the microscopic system has been developed, the high-level foraging task is learned through reinforcement learning. We achieve several advantages over the use of end-to-end strategies:

- The search space of the RL problem is substantially simplified, making it possible to apply this learning method to a realistically modeled robotic swarm with differential actuator and range sensors.
- This strategy allows experts to adjust the robotic level behaviors without changing the macroscopic behavior simply by subsequent adjustments and maintenance in a real environment.
- The learning of degenerate behaviors (such as robots moving backwards, which offers no real advantage to the search task) is avoided, simplifying the design of the reward function and even relaxing many problems encountered when using RL end-to-end systems.

The main objective of this work is to learn a foraging behavior using a robotic swarm (modeled with realistic physics) combining fuzzy logic (microscopic system) and deep reinforcement learning (macroscopic system). Although this approach is not the first to combine both strategies [6,8], this paper is one of the first to apply this approach to a swarm foraging problem in a functional way using a physically modeled differential robot with realistic sensors and actuators.

2. Problem Specification

In this article, we develop a macroscopic foraging behavior for finding and moving an object collaboratively towards a goal point. In this case, we use two microscopic behaviors for displacement and collision avoidance. These behaviors are defined using fuzzy logic due to its versatility and ease of use for this type of robotic system.

From these fuzzy behaviors, we want to obtain a macroscopic controller that must be able to achieve the final desired behavior: the movement of the swarm towards the object to be displaced, which is then pushed in a coherent way until a goal is reached.

We define the foraging task as follows. Consider a robot swarm with R robots that move in an environment E with objects called food F (represented as boxes in our environment) that need to be gathered at the collection area P_{nest}, also called the goal. The nesting place is a known area on the surface E, in which it is necessary to collect food in the environment. The goal of the swarm is to be able to locate and approach the food F and push it to P_{nest} using only their own bodies. There are no grippers or special robot shapes usedto facilitate this movement. The swarm must learn how to do this push process effectively. For this task, we assume that there exists a sensor (camera or range sensor) or a heterogeneous agent (for example, with zenith vision) able to detect the food position relative to R. There are no other objects in the environment, other than the individuals in the swarm, that could hinder the process of pushing the food towards P_{nest}.

Therefore, we assume that the swarm knows the location of the object to be moved and the goal towards which it must be pushed. Furthermore, for this approximation, the individual agents must receive their coordinates relative to the object and their goal. Finally, the macroscopic task controller must also know the end goal that the pushable object must achieve.

There are several approaches that can provide the necessary data in both indoor and outdoor environments with low-cost sensors and that are compatible with our proposal, such as the use of a zenith camera in environments that support it, the use of You Only Look Once (YOLO) networks for short distances (https://github.com/leggedrobotics/darknet_ros accessed on 22 March 2021), the use of light detection and ranging (LiDAR) or RGBD cameras, ultra-wideband (UWB) for indoor positioning (https://www.iotforall.com/ultra-wideband-uwb-indoor-positioning accessed on 22 March 2021) or GPS/4G devices for long distances.

Next, we present the details of the robotic platform used as well as the characteristics of the physical simulation utilized in the experimentation. In addition, the microscopic fuzzy systems to be used by the macroscopic model are presented. Finally, the reinforcement learning problem is formalized using the Markov decision process (MDP).

2.1. Robotic Platform

For the development of this work, a robotic simulator that allows realistic simulations of multiple environments was implemented. Some ideas from the Argos simulator (https://www.argos-sim.info/ accessed on 22 March 2021) —a simulator oriented towards large swarms mainly based on realistic physical engines—were taken as a basis. Specifically, our simulator uses the realistic Chipmunk (https://chipmunk-physics.net/ accessed on 22 March 2021) physical engine, which is one of the fastest on the market. It is designed to execute hundreds of instances of simulations to be able to parallelize the RL algorithms. This is a basic requirement to use a robotic swarm simulator for reinforcement learning tasks.

A holonomic robot with two active wheels and one for support was modeled. The shape of the robot was approximated to a 10 cm radius circle with 650 grams of weight. A low-level controller was used to alter the robot speed to move each agent. On this basis, our simulator provided a differential controller that allowed the speeds of the wheels (vl, vr) to be calculated given the required rotational and translational speeds.

A differential drive consists of two drive wheels mounted on a common axis, where each wheel can be independently driven either forward or backward. With this configuration, a robot must rotate (at a rate of ω) about a point that lies along its common left and right wheel axes, called the instantaneous center of curvature (R_{ICC}). By varying the velocities of the two wheels, we were able to vary the trajectories that each robot took. There are three interesting cases with these kinds of drives: if $vl = vr$, then the robot develops a forward linear motion along a straight line. If $vl = -vr$, then $_{R} = 0$, and we have rotation about the midpoint of the wheel axis. If $vl = 0$, then we have rotation about the left wheel (which is symmetric with vr). At time t, we can predict the next position of the robot $t + \delta t$:

$$\begin{bmatrix} R'_x \\ R'_y \\ R'_\alpha \end{bmatrix} = \begin{bmatrix} \cos(\omega\delta t) & -\sin(\omega\delta t) & 0 \\ \sin(\omega\delta t) & \cos(\omega\delta t) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} R_x - R_{ICCx} \\ R_y - R_{ICCy} \\ R_\alpha \end{bmatrix} + \begin{bmatrix} R_{ICCx} \\ R_{ICCy} \\ \omega\delta t \end{bmatrix} \quad (1)$$

We can describe the position of any robot capable of moving in a particular direction $\alpha(t)$ at a given velocity $V(t)$ as

$$
\begin{aligned}
R_x(t) &= \int_0^t V(t)\cos[\alpha(t)]dt \\
R_y(t) &= \int_0^t V(t)\sin[\alpha(t)]dt \\
R_\alpha(t) &= \int_0^t \omega(t)dt
\end{aligned}
\tag{2}
$$

Unfortunately, a differential drive robot imposes nonholonomic constraints on the establishment of its position; thus, answering the question of how to control the robot to reach a given configuration is not trivial (we cannot simply specify a robot pose and find the velocities needed to move to this position). This consideration motivates the classical strategy of moving the robot along a straight line, rotating in place to perform a turn, and then moving straight again, repeating this process until the goal is reached.

Finally, it is worth mentioning that all robots had a laser sensor based on the SLAMTEC RPLIDAR A1 device, set to a detection range of 0 to 4 m of 360°. We discretized this information to use 13 angular readings to develop this behavior.

Both the actuator and the robot range sensor were realistically modeled, taking noise models into account. Specifically, the models used were based on the perceptual and actuator models presented in [9], adjusted to our physical robots.

2.2. Low-Level Controller Design

We used two fuzzy systems to define the microscopic controllers of the system.

The first system was in charge of avoiding obstacles and direct collisions. It used the information from 13 angular readings (see Equation (3)) obtained by the LiDAR sensor, grouping them into three zones ($left = [0..3]$, $right = [9..12]$, $front = [4..8]$). From this information, it emitted the translational and rotational speeds of the agent to avoid collisions.

$$
\begin{cases}
0: & -105 \\
1: & -90 \\
2: & -75 \\
3: & -55 \\
4: & -35 \\
5: & -15 \\
6: & 0 \\
7: & 15 \\
\ldots & \\
12: & 105
\end{cases}
\tag{3}
$$

The second microscopic system (rendezvous) directed the agent towards a stated goal. To do this, it obtained the position of the robot relative to the goal and emitted the required rotational and translational speeds. Both systems are presented in Table 1.

In Figure 1, the fuzzy terms and sets for both microscopic systems are presented. The Takagi–Sugeno inference was used to extract knowledge from both behaviors, using the weighted average Takagi–Sugeno decoder. As a conjunction rule, we used the algebraic product, while for the disjunction rule, we used the algebraic sum.

The final microscopic controller was required to combine the output of both systems. In our case, we limited the rules of the fuzzy avoidance system to those areas far from the target goal since the task of pushing the box required collision. This limitation could lead to random, low-speed collisions between robots if there were an active target, but this is unavoidable if more advanced information (for example, knowing the kind of objects that are detected by the range sensor in addition to their distance) is not available. Specifically, the final microscopic system—i.e., the one used in our experiments—is presented in Table 1.

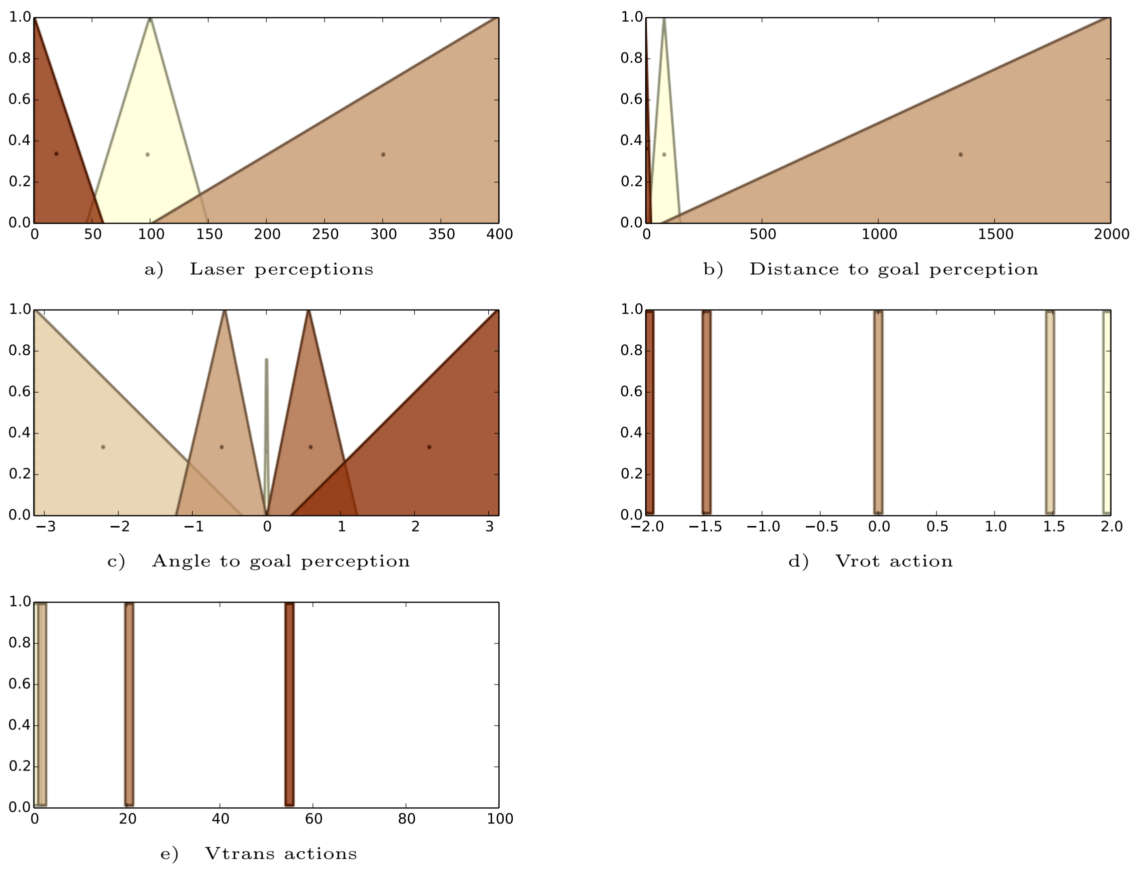

Figure 1. Fuzzy terms and sets for the microscopic controller. This controller was executed by each robot member of the swarm. (**a**) Fuzzy set used for the perception of lasers. The available readings were discretized into three groups (left, right, front), where each one contained three angle readings. The terms for this set are near, far, and emergency (emer). (**b**) Fuzzy set dist, which specifies the distance to the goal. The terms in this set are near, med and far. (**c**) Fuzzy set ang, which was used to represent the angle of the box with respect to the goal. The terms used are far left (farl), medium left (medl), near, medium right (medr) and far right (farr). (**d**) Fuzzy set that represents the rotational speed of the robot. The terms are high left (hleft), left, none, right and high right (hright). The Takagi–Sugeno weighted average was used as a defuzzifier, with a default value of 0. (**e**) Diffuse set, which represents the rotational speed of the robot. The terms are stop, low, medium and high. The Takagi–Sugeno weighted average was used as a defuzzifier, with a default value of 0.

Swarm Formation

Once the microscopic tasks of the system were defined, we combined them to develop a foraging task. This task used high-level data, leaving the low-level details to the fuzzy drivers. The two microscopic behaviors presented above allowed each individual in the swarm to be arranged in a certain position to avoid obstacles in their movement.

To develop the foraging task, a U-shaped swarm formation, obtained by combining the two previous behaviors, was proposed. An example of this robot formation is presented in Figure 2. This training was used in the subsequent learning phase to move the swarm, scale it or rotate it to solve this problem.

It is important to note that the existence of swarm coordination was assumed. This coordination can occur exogenously [10] or endogenously, as in this case, where the agents needed to be visible to a zenith agent controller, which guided the swarm towards the required goal.

In this way, it should be noted that the microscopic behavior of each individual in the swarm was responsible for moving its robot so that it could be placed in the position required by this formation. Therefore, the next step to be carried out was to learn how to move this robotic formation to achieve the required foraging task.

Table 1. Rules of each fuzzy microscopic behavior along with the corresponding weights. The combination of the microscopic avoidance system with the microscopic rendezvous system is also shown. The algebraic product is used as the conjunction rule. The algebraic sum is used as the disjunction rule.

Behavior	N.	Antecedent	Consequent
Avoidance	A1	left is emer and right is emer and front is emer	vrot is right
	A2	left is emer and right is emer and front is far	vrot is none
	A3	left is emer and right is far	vrot is hright
	A4	right is emer and left is far	vrot is hleft
	A5	left is near and right is far	vrot is right
	A6	right is near and left is far	vrot is left
Rendezvous	F1	ang is near	vrot is none
	F2	ang is medl	vrot is right
	F3	ang is medr	vrot is left
	F4	ang is farl	vrot is hright
	F5	ang is farr	vrot is hleft
	F6	dist is far	vtrans is high
	F7	dist is med	vtrans is medium
	F8	dist is near	vtrans is stop
	F9	front is emer	vtrans is medium
	F10	front is near	vtrans is medium
Combination	C1..C6	C$_n$: (dist is far or dist is med) and + A$_n$	Same as base behavior
	C7..C17	C7..17: F$_{n-6}$	

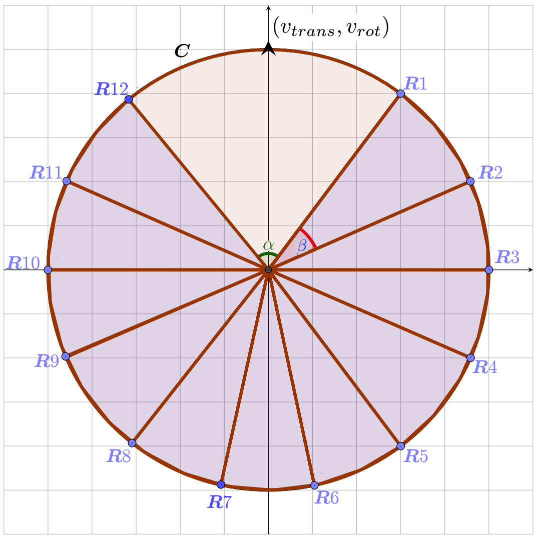

Figure 2. Example of robot formation R_n and associated parameters. A circumference of radius C is shown, on which 12 robots are placed. The α angle limits the area not occupied by robots to generate the U-shaped formation. The β angle is obtained such that $\beta = \frac{2\pi-\alpha}{n}$ and is the same between correlative pairs of R. The macroscopic formation can be moved and rotated with translational and rotational speed. This formation can also vary the radius C. The diffuse microscopic behaviors are used to calculate the local forces to be applied to each robot to achieve this final formation.

2.3. *Reinforcement Learning Architecture*

Next, the problem was formalized as an MDP—a necessary process for reinforcement learning.

The robotic swarm was the main component that interacted with the environment, which was unknown in our case. In each step t, the swarm observed its state $s_t \in S$.

Once the environment was perceived, an action $a \in A$ was selected to make a transition between a state s_t and the following one s_{t+1}. For each transition, the swarm immediately received a reward, such that $r_t = r(s_t, a_t, s_{t+1}) \in \mathbb{R}$. This process was followed by the system to handle the swarm in each step t, providing actions that maximized the total reward of the behavior. We could assume that transitions between states satisfied the Markovian property:

$$P(s_{t+1}|s_t, a_t, s_{t-1}, a_{t-1}, ...) = P(s_{t+1}|s_t, a_t) \tag{4}$$

In this way, we were able to consider this process of finding the object target and pushing it towards the goal as a discrete-time Markov decision process (MDP). The agent's goal was to learn a policy π, determined stochastically as $\pi(a|s) = P(a|s)$ or deterministically as $a = \mu(s)$. This policy maximized the reward of all steps taken during the h_π history of the agent:

$$R(h_\pi) = \sum_{t-1}^{T} \gamma^{t-1} r(s_t, a_t, s_{t+1}) \tag{5}$$

Therefore, the optimal policy was that which maximizes

$$\pi^* = \arg\max_{\pi} R(h_\pi) \tag{6}$$

Markov Decision Process Design

Reinforcement learning has been widely applied in robotic tasks [4,5,11,12]. In [13], a detailed explanation of its operation and application to obtaining robotic control policies is shown. A reinforcement learning problem contains various integral components, such as observations, actions and rewards. Below, we describe the details of each of these parts.

Observation Space. As previously mentioned, the perception of the swarm at a given moment was directly related to the task to be carried out. It was assumed that the position of the object obj to be pushed and the position of the $goal$ are known. The swarm knew the rotation fr_{rot} and the scaling fr_{sca} of the formation used in this behavior. It is important to highlight that obj could not be unequivocally located by the sensors of the robots in the formation, which could perceive only the distances of the robots from the environment but not the type of objects that generated this perception. $S = (obj_x, obj_y, goal_x, goal_y, fr_{rot}, fr_{sca})$

Action Space. The actions to be carried out by the swarm were defined by the formation of the swarm. In this case, the swarm could move the formation relative to its current position. This could be achieved by moving it at linear (v_{tras}) and rotational (v_{rot}) speed, as well as scaling it by (v_{sca}). Therefore, only three continuous actions allowed the entire swarm to coordinate, regardless of the number of individuals that formed it; $A = (v_{tras}, v_{rot}, v_{sca})$.

Network Architecture. Given an observation entry at time t that characterized the state of the swarm S_t and its output A_t, we defined the network that developed the policy to generate A_t from S_t.

Thus, a three-layer network was designed (see Figure 3). There were six network inputs, corresponding to the distance and angle of the swarm to the box to be pushed, the distance and angle of the box with respect to the goal and the rotation and scale of the current formation. The hidden layers contained 64 neurons with tanh activation. The last layer was the one that connected directly with the actions of the swarm. These outputs defined the relative changes of the formation: its translation, rotation and scaling. Because we were working with a continuous policy, the output layer was linear, with one neuron for each output.

It is worth mentioning that several tests were carried out with different architectures and activation functions. The network presented worked best for the training process and subsequent operation through the algorithm; in addition, it was very contained in size. This aspect allowed us to train it not only by gradient descent algorithms but also using evolutionary techniques. For illustrative purposes, Figure 4 shows the average reward obtained in several training sessions depending on the training algorithm, architecture and activation function.

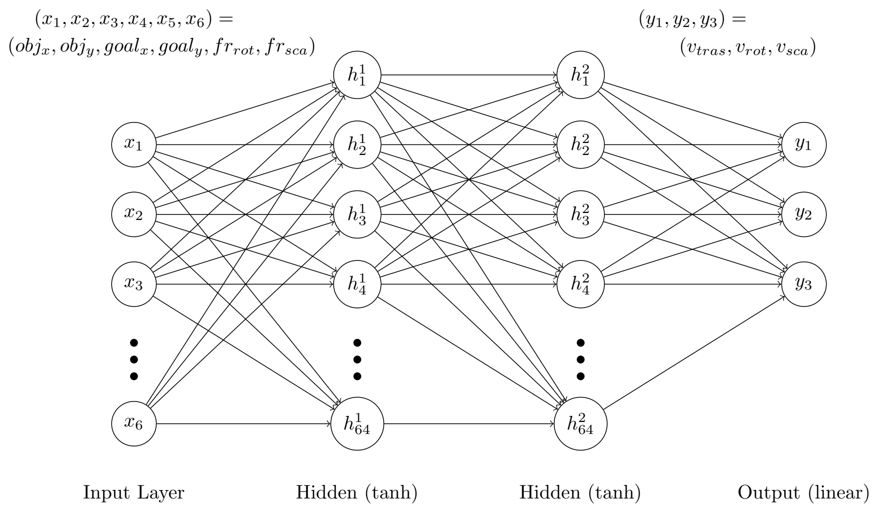

Figure 3. Deep neural network used as a model for reinforcement learning. The activation function of each neuron is shown in parentheses. The network uses the observations as inputs and the action space defined in the Markov decision process (MDP) as the output.

The foraging task defined in this article had several issues that made it difficult to learn. On the one hand, each execution of the swarm could be potentially different even under identical or similar conditions, given the noise of both the sensors and actuators. In addition, the swarm needed to learn to avoid pushing the object in a direction opposite to that of the goal and therefore often needed to surround the object to find the optimal push direction.

Reward is a fundamental part of the design of the MDP, since it is the assessment that guides the achievement of the policy. Therefore, this reward must appropriately contain both positive and negative feedback to avoid the previous problems and enable effective and rapid learning.

In this way, we designed a continuous reward that assessed the performance of the behavior at each step. In Equation (7), the used reward function is presented.

There were basically three positive rewards: if the distance from the swarm to the box was improved, the reward was 1; once the swarm reached the box, it was rewarded with 5; as soon as it reached the goal, a reward of 100 was given. Any other case was valued with a negative reward of -1 to prioritize behaviors that solved the task in the shortest possible way.

It is worth mentioning that, to prevent the swarm from heading straight towards the box in the initial phases, a potential function U was introduced, which is discussed below.

$$reward = \begin{cases} 100 & \text{if } box \text{ is in } goal \\ 1 & \text{if } swarm \text{ does not arrive at } box \\ & \text{ and improve } dist(swarm, box) \\ & \text{ and } |U|^{\beta} < 0.2 \\ 5 & \text{if } swarm \text{ arrives at } box \\ & \text{ and improves } dist(box, goal) \\ -1 & \text{other} \end{cases} \tag{7}$$

The learning of this behavior required two parts; i.e., going to the target and pushing it (many times in the direction opposite that of the swarm heading when moving towards the target), with the first part of the movement likely hindering the second. Therefore, to prevent the pushing behavior from being carried out in the wrong orientation, a potential function $U(\boldsymbol{x})$ was designed. This function was added to the reward function to be able to use an information gradient in the navigation plan. Many studies and different alternatives have been presented that have aimed to define potential functions that behave desirably as a feedback motion plan. We used a potential function U that is quadratic with the distance.

$$U(\boldsymbol{x}) = \alpha \sum_{i=1}^{N} q \frac{\boldsymbol{x} - \boldsymbol{x}_g}{\|\boldsymbol{x} - \boldsymbol{x}_g\|} \tag{8}$$

Specifically, we used $U(\boldsymbol{x})$ to reward the good positioning of the swarm when pushing and to avoid the situation in which, in the approach phase, the box was pushed unexpectedly. For this, two charges were introduced in U: a repulsive one, located in the central position of the box, with intensity -1, and an attractive one with intensity 1 in the position of the box opposite the goal. In the reward function, we specifically used the normalized value of $|U|^{\beta}$, where β was between 1 and 100 to prioritize avoidance in high potential areas. In Figure 5, we present an example of the potential function for a given environment.

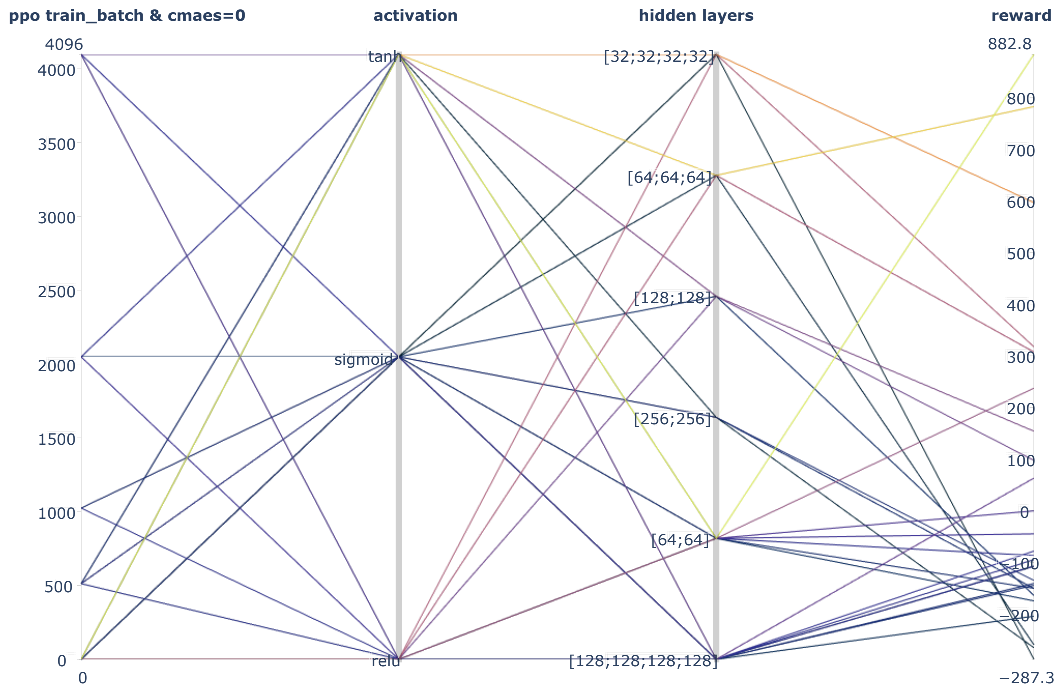

Figure 4. Parallel coordinate plot showing the average reward obtained in various training sessions as a function of the training algorithm, network architecture, training batch size (Proximal Policy Optimization (PPO)) and activation function. First, several PPO training sessions are presented, with the following main hyperparameters: $\gamma = 0.995$, $\lambda = 0.95$, $lr = 3 \times 10^{-4}$ (obtained with a stochastic hyperparameter search). Secondly, training results of the covariance matrix adaptation evolution strategy (CMA-ES) algorithm are shown, with the following parameters ($pop_{size} = 8$, $\sigma_{init} = 0.1$). It should be noted that the first axis on the left of the graph represents both the use of PPO algorithm and its training batch size (value greater than 0) and the use of the CMA-ES algorithm (value equal to 0). We found that the PPO algorithm often misses promising policies, preventing us from obtaining the desired behavior and thus having a lower average reward. The result of this hyperparameter optimization phase determines the number of neurons, layer structure and activation function used for our final network.

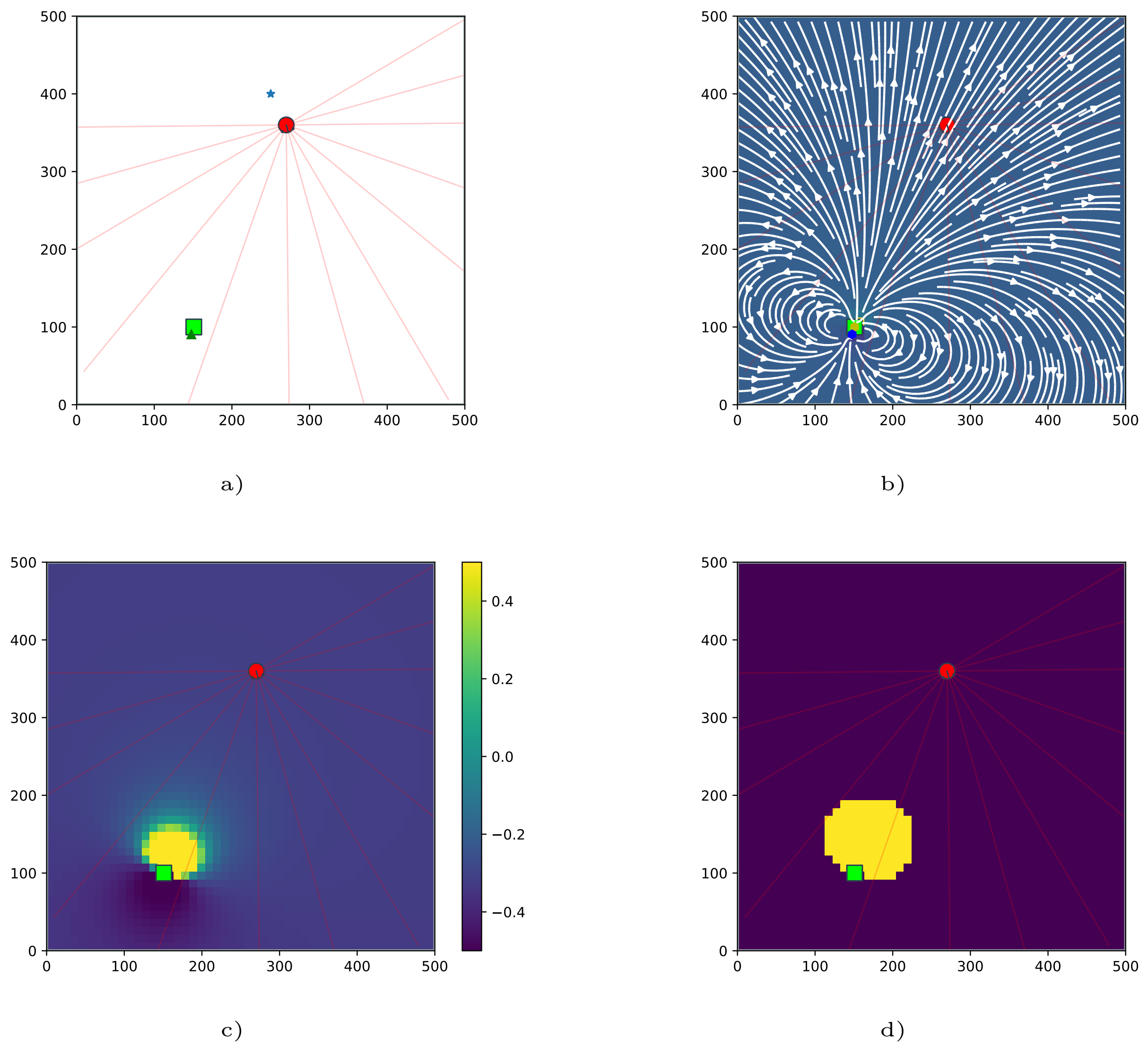

Figure 5. Example of the calculation of forces used for the reward function. Environment units are in cm. (**a**) Initial distribution of the robots. The orientation of each robot and the distribution of its sensors are observed. The target (star) and the box to be moved are also displayed. (**b**) Stream diagram generated by the two forces (repulsive (yellow points) and attractive (blue points)) located in the vicinity of the box for this environment. (**c**) $|U|^{\beta}$ for $\beta = 60$. (**d**) Discretized use of the function, where the threshold values are represented in yellow and $\epsilon = 0.2$, such that $|U|^{\beta} > \epsilon$.

3. Experiments

Both the training phase enacted to obtain the desired behavior and the experiments carried out are described below. This behavior was analyzed starting from a swarm of three individuals. Next, several tests with different numbers of robots were developed to verify the scalability of this behavior.

3.1. Training Phase

Once the reinforcement learning problem was defined, we needed to train the neural network presented in our MDP. To accomplish this task, we tried two different training approaches. On the one hand, one of the state-of-the-art algorithms for this type of learning was used: Proximal Policy Optimization (PPO) [14]. This is an algorithm from the family of policy gradient methods for reinforcement learning that alternates between sampling data obtained through interactions with the environment and optimizes a surrogate objective function using stochastic gradient ascent. However, we found that PPO often loses promising policies, thus hindering us from obtaining the desired behavior. Although we applied the reward shaping techniques presented above and performed stochastic hyperparameter searches, we found that the existence of local minima and the intrinsic randomness of the actuators and sensors of the robots and of the environment itself encouraged the PPO algorithm to fail and not find promising policies. This problem occurred mainly after the swarm located the box, making it unable to proceed; i.e., to learn in a general way how to push the box towards the goal.

Given the results, we opted to use a non-gradient-dependent learning method; i.e., the covariance matrix adaptation evolution strategy (CMA-ES) [15,16], which represents the population by a full-covariance multivariate Gaussian. The CMA-ES has been extremely successful in solving optimization problems in low to medium dimensions [17].

The behaviors learned with the CMA-ES were much more stable than those obtained through PPO learning, obtaining stable policies that were capable of solving the problem. Furthermore, the CMA-ES requires practically no hyperparameter adjustments or an initial search space. For our specific case, we carried out the training with a population size of 8 and $\sigma_{init} = 0.1$. Figure 4 shows several results of the training sessions performed with both PPO and CMA-ES algorithms.

Each episode began with three robots at P_{nest} and contained a random food box F. Episodes consisted of a maximum of 700 simulation steps (each simulation step was equal to 2/3 seconds, so an episode lasted approximately 8 minutes), after which the episode was considered finished (and not completed if the goal was not reached by the box). An episode ended in fewer steps if a box of food F reached P_{nest}. Each behavior was simulated three times to improve the evaluation of the policy. Therefore, each iteration corresponded to the execution of 24 episodes and a maximum of 16,800 simulation steps.

The best training result, using the CMA-ES algorithm, is presented in Figure 6. We observed that the swarm was able to move towards the box in only 50 iterations. From that point on, it pushed the box until it reached the goal (this was achieved within 80 iterations). The following iterations involved refinements of the learned behavior.

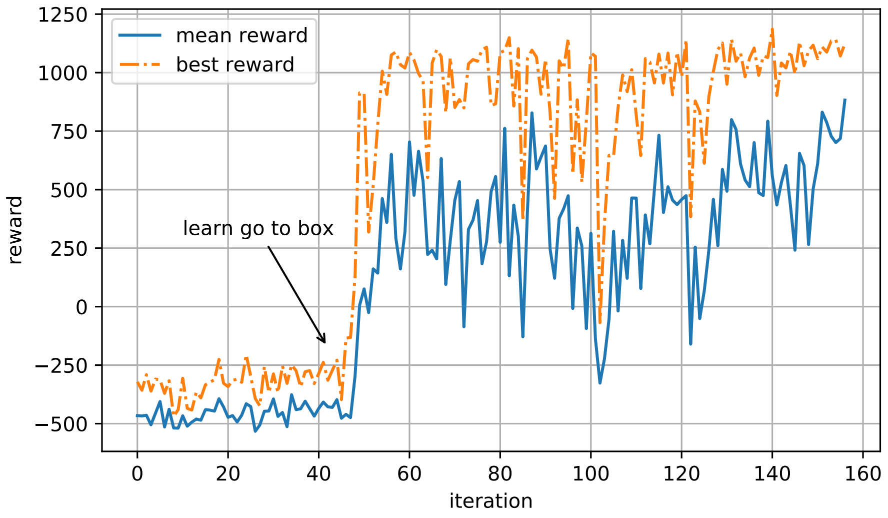

Figure 6. Policy training using the CMA-ES. The average and maximum reward of the learned behavior for each iteration are shown. Each iteration includes 16,800 simulation steps. The point at which the swarm has learned to move the food box correctly is marked.

3.2. Evaluation Phase

Then, several tests were performed to thoroughly examine the learned policy. On the one hand, an example of the learned policy execution (for three robots in an environment with three boxes) is presented. On the other hand, an exhaustive study of this behavior was carried out with a total of 800 tests with random starting positions of the box to be transported. Within this set of tests, we verified how the size of the box and its weight affected the learned behavior.

Figures 7 and 8 show the simulator captures and several traces of how each of the swarm robots moved to the box and pushed it to its goal. Several questions are answered by these figures. Initially, we see how the policy allowed the swarm to learn to move towards the box appropriately. We observe that the robots did not go directly towards the box, which caused them to collide with it in a direction other than that required for the push process. Instead, they accessed it from the side, which greatly facilitated the arrangement of the robots in the subsequent process. Furthermore, we see how the formation of the robots was important as the three robots collaborated, given the disposition and scale learned in the behavior, to complete the pushing process. Finally, we also observed that this behavior was capable of adjusting to possible losses and the slipping of the box, since the rounded shape of the robots did not facilitate the box pushing behavior.

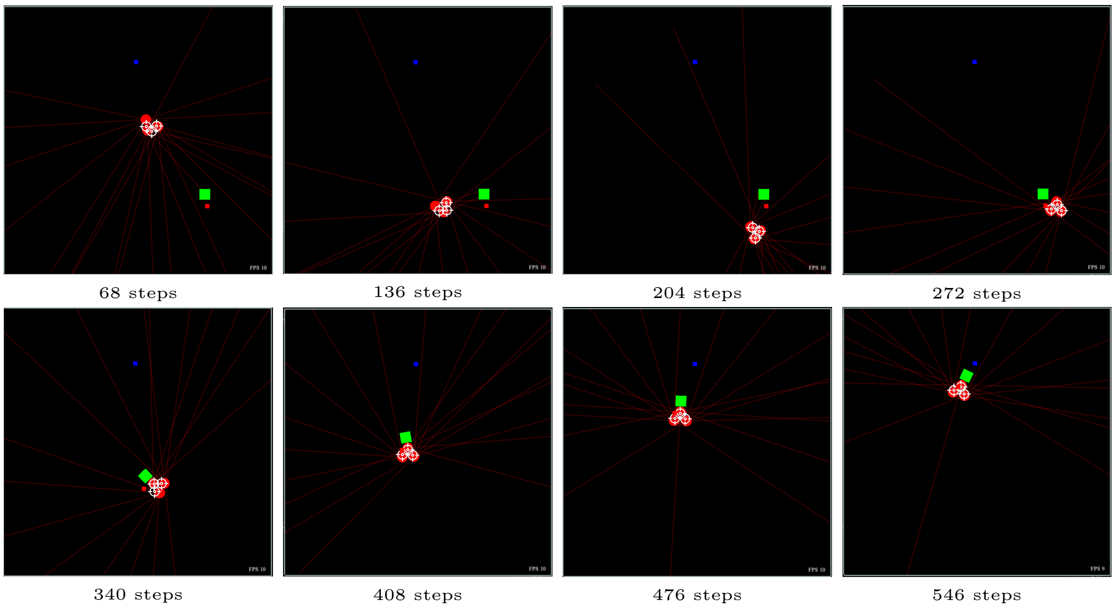

Figure 7. Environment of 500 $\times$ 500 cm with three robots (red circles) and one box (green) that needed to be located and transported to the home zone (blue). The figure presents several steps of our simulator with the learned policy. Each simulation step was equal to 2/3 seconds, so this simulation lasted 6 minutes. It is shown how the swarm was able to move towards the box and push it to the goal within 546 simulation steps.

As commented in Section 3.1, the training for this behavior used a swarm of three individuals with random positions of F. In addition, the object to be found and pushed had a box size of 20 centimeters and a weight of 1 kg. In Table 2, 400 executions of the behavior are shown for four different box weights. It can be seen that the behavior had a success rate higher than 80% even for box weights twice or half that used in training. In fact, it is notable that higher box weights corresponded to a higher success rate, possibly because of the reduction in inertia of the swarm pushing process. We also observed some variance in the final box–goal distance in the tests in which the box did not reach the goal. For the 5 m environment used in the simulation, the expected average error for failing behaviors was estimated to be approximately 2.5 m. However, we observe that many of the behaviors had a minor error with considerable variance, as many of them could have been solved if the 700 allowed iterations were expanded.

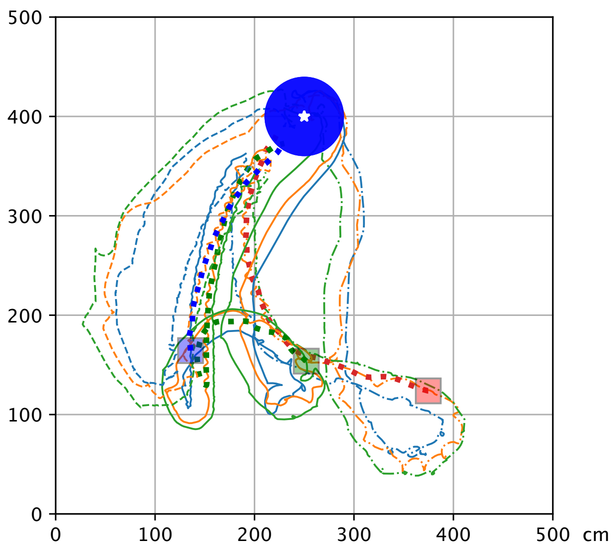

Figure 8. Environment of 500 × 500 cm with three robots and three boxes that needed to be located and transported to the home zone (goal). The figure presents the trace of each of the robots in the swarm. The origin of the swarm is indicated by the white point, surrounded by the goal zone (in blue). It is shown how the swarm was able to move towards the boxes and push them to the goal.

Table 2. Policy tests for boxes of different weights. For each weight, 100 executions of the behavior were carried out. If the box did not reach the goal in 700 iterations, then the test was marked as incorrect. For all incorrect tests of each weight, the box–goal mean distance and standard deviation are shown.

Box Mass (g)	Correct Executions	Fail Distance Mean (cm)	Fail Distance Std (cm)
500	81%	236.92	90.78
1000	80%	182.13	80.15
1500	83%	171.59	90.75
2000	85%	193.76	82.59

Finally, we intended to investigate the behavior under different box sizes. Although the training was developed using only a single size (20 cm), it was interesting to analyze its robustness against different box sizes that were not previously considered. In Table 3, we observe how this parameter directly affected the behavior success rate. From 10 cm to 30 cm, we see how this policy developed the correct foraging behavior an average of 70% of the time. However, a larger box size resulted in the swarm being unable to push the box. This outcome was expected, mainly because the reward potential function presented above was dependent on the box size; therefore, the controller could not learn to position the swarm correctly for larger boxes. However, it is easy to avoid this problem if it is necessary fo the system to work with different box sizes; to do this, we should train the robots with the sizes that will be used in the final platform.

Table 3. Policy tests for different square box sizes. For each size, 100 executions of the behavior were carried out. If the box did not reach the goal in 700 iterations, then the test was marked as incorrect. For all incorrect tests of each size, the box–goal mean distance and standard deviation are shown.

Box Size (cm)	Correct Executions	Fail Distance Mean (cm)	Fail Distance Std (cm)
10	70%	264.94	147.51
20	80%	182.13	80.15
30	68%	182.12	85.18
40	29%	167.12	65.69

In this section, we showed how the policy developed a correct behavior that allowed the swarm to perform both the location and pushing of the box to its goal. This is also a robust behavior, where alterations to both the weight and the size of the box (if they were maintained within parameters consistent with the training values) did not affect the foraging process.

3.3. Scalability Tests

Finally, we intended to check the swarm scalability of the learned policy. Although the training considered only three individuals, one of the expected advantages of the reinforcement-learned macroscopic behavior is the flexibility of its swarm formation and the microscopic policy combination. Thus, several behavior tests were carried out for swarm formations of 2 to 10 individuals.

Table 4 shows the result of 1000 executions of the behavior for different swarm sizes. Although the best success rates were close to the number of robots used in training, we observed several notable issues. On the one hand, we see that for 2 to 6 robots, the swarm carried out its push task with percentages of success of around 70%. This result indicated a reasonable scalability rate for the presented behavior, taking into account the fact that the training involved a swarm of a fixed size of 3. Higher numbers of robots in the swarm caused the success rate to decline, mainly due to the increase in collisions between them while moving.

More specifically, the tendency of the standard deviation was to increase as the mean distance between tests increased. To highlight this aspect, the calculation of the coefficient of variation (CV) was added. The higher the CV, the greater the dispersion in the variable. In our case, the higher the number of swarm agents in the same bounded space, the greater the disruptions or conflicts in the swarm movement. However, this did not limit the swarm's ability to solve the problem, although not as easily as with a smaller number of agents.

We believe that these two aspects increased the dispersion of the CV coefficient. This result seems to indicate that runs with more steps will increase the success rate of larger swarms.

Table 4. Scalability tests. For each swarm size, 100 executions of the behavior were carried out. If the box did not reach the goal in 700 iterations, then the test was marked as incorrect. For all incorrect tests of each size, the box–goal mean distance, the standard deviation and the coefficient of variation $(CV = (std/mean) \times 100)$ are shown.

Num. Robots	Correct Executions	Fail Distance Mean (cm)	Fail Distance Std (cm)	CV
2	65%	173.68	75.54	43.49
3	80%	182.13	80.15	44.00
4	83%	192.22	113.61	59.10
5	67%	221.05	114.64	51.86
6	64%	216.78	125.35	57.82
7	59%	223.71	130.62	58.38
8	40%	236.16	123.38	52.24
9	30%	258.02	137.79	53.40
10	37%	241.39	134.42	55.68

4. Discussion

Considering the results of the developed experiments, we can see that the separation of pre-designed microscopic behaviors together with a macroscopic behavior (which must be learned) is feasible. We also observe how the macroscopic system is capable of adapting to unseen changes in the environment, operating coherently with its initial goal.

We believe that the use of fuzzy logic in the microscopic system not only greatly simplifies the learning of the global system (as it significantly reduces the dimensionality of the problem to be solved) but could also provide a way to easily adjust these microscopic behaviors without significant macroscopic alterations. Thus, it also allows engineers to develop safety-critical systems (the reactive behavior of the robots; e.g., their speeds related to obstacle avoidance and local goals) in a simple way, with the required methodology. However, delegating these tasks to an end-to-end network could add an additional control phase to verify that the trained system is safe in its application domain.

End-to-end strategies have clear advantages, including the designer's lack of concern for microscopic tasks when dealing with only a global behavior. However, the dimensionality of the problem is critical in determining whether deep reinforcement learning can be used. Moreover, when dealing with robotic swarms, we have the added problem that this dimensionality can be much higher. Hybrid strategies such as those presented in this article can be of great help to allow the use of RL strategies in problems with high dimensionality and also reduce the difficulties derived from the existence of local minima and the intrinsic randomness of the actuators and sensors of the robots.

5. Conclusions and Future Works

This article presents a foraging behavior that is capable of directing a swarm of robots from an initial position P_{nest} to a resource F that must be moved to P_{nest}. The robots must use their own bodies for the pushing process, which makes the task especially difficult. Furthermore, both the robots and the F resources are simulated with realistic physics.

To achieve this behavior, we use a combination of two microscopic tasks based on fuzzy logic that develop the avoidance and the rendezvous required to go to a specific position. These tasks are used to control the entire swarm to develop the foraging behavior. Furthermore, the macroscopic foraging problem is modeled using an MDP and is learned via deep reinforcement learning using the CMA-ES algorithm.

A thorough behavioral analysis is performed with hundreds of tests. Very good results (with a behavior success rate higher than 80% in the environments used for the training process) are obtained for this task, even for previously untrained cases with different numbers of robots and with different sizes and weights of the F resource (60–80% of success for many of the environments with non-extreme modifications and always with a maximum limit of 700 iterations to reach the goal). This work offers us a glimpse into the advantages of combining RL systems with other types of systems: the reduction of the dimensionality of the search problem, custom design of the simple basic behaviors, and, ultimately, the easier adaptation, modification and understanding of the learned behavior.

We are currently assessing the use of the learned macroscopic policy by altering only the basic fuzzy microscopic behaviors to adjust it to the final deployment platform and to analyze how modifying only the microscopic behaviors can change the global behavior to other desired behaviors.

Author Contributions: The authors have contributed equally to this work. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the Ministerio de Ciencia, Innovación y Universidades (Spain), project RTI2018-096219-B-I00. Project co-financed with FEDER funds.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data contained within the article.

Conflicts of Interest: The authors declare no conflicts of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

References

1. Song, Y.; Fang, X.; Liu, B.; Li, C.; Li, Y.; Yang, S.X. A novel foraging algorithm for swarm robotics based on virtual pheromones and neural network. *Appl. Soft Comput.* **2020**, *90*, 106156. [CrossRef]

2. Yogeswaran, M.; Ponnambalam, S.; Kanagaraj, G. Reinforcement learning in swarm-robotics for multi-agent foraging-task domain. In Proceedings of the 2013 IEEE Symposium on Swarm Intelligence (SIS), Singapore, 16–19 April 2013; pp. 15–21. [CrossRef]
3. Barrios-Aranibar, D.; Goncalves, L.M.G. Learning to Collaborate from Delayed Rewards in Foraging Like Environments. *Jornadas Peruanas De Computación JPC* **2007**.
4. Iima, H.; Kuroe, Y. Swarm reinforcement learning methods improving certainty of learning for a multi-robot formation problem. In Proceedings of the 2015 IEEE Congress on Evolutionary Computation (CEC), Sendai, Japan, 25–28 May 2015; pp. 3026–3033. [CrossRef]
5. Bernstein, A.V.; Burnaev, E.V.; Kachan, O.N. Reinforcement Learning for Computer Vision and Robot Navigation. In *Machine Learning and Data Mining in Pattern Recognition*; Perner, P., Ed.; Springer: Cham, Switzerland, 2018; Volume 10935, pp. 258–272. [CrossRef]
6. Fathinezhad, F.; Derhami, V.; Rezaeian, M. Supervised fuzzy reinforcement learning for robot navigation. *Appl. Soft Comput.* **2016**, *40*, 33–41. [CrossRef]
7. Efremov, M.A.; Kholod, I.I. Swarm Robotics Foraging Approaches. In Proceedings of the 2020 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus), St. Petersburg and Moscow, Russia, 27–30 January 2020; pp. 299–304. [CrossRef]
8. Hein, D.; Hentschel, A.; Runkler, T.; Udluft, S. Particle swarm optimization for generating interpretable fuzzy reinforcement learning policies. *Eng. Appl. Artif. Intell.* **2017**, *65*, 87–98. [CrossRef]
9. Thrun, S.; Burgard, W.; Fox, D. *Probabilistic Robotics*, illustrated auflage ed.; The MIT Press: Cambridge, MA, USA, 2005.
10. Gebhardt, G.H.; Daun, K.; Schnaubelt, M.; Neumann, G. Learning Robust Policies for Object Manipulation with Robot Swarms. In Proceedings of the 2018 IEEE International Conference on Robotics and Automation (ICRA), Brisbane, Australia, 21–25 May 2018; pp. 7688–7695. [CrossRef]
11. Hüttenrauch, M.; Šošić, A.; Neumann, G. Guided Deep Reinforcement Learning for Swarm Systems. *arXiv*, **2017**, arXiv:1709.06011.
12. Hüttenrauch, M.; Šošić, A.; Neumann, G. Deep Reinforcement Learning for Swarm Systems. *arXiv* **2019**, arXiv:1807.06613.
13. Tai, L.; Zhang, J.; Liu, M.; Boedecker, J.; Burgard, W. A Survey of Deep Network Solutions for Learning Control in Robotics: From Reinforcement to Imitation. *arXiv* **2016**, arXiv:1612.07139.
14. Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; Klimov, O. Proximal Policy Optimization Algorithms. *arXiv* **2017**, arXiv:1707.06347.
15. Hansen, N.; Ostermeier, A. Completely Derandomized Self-Adaptation in Evolution Strategies. *Evolut. Comput.* **2001**, *9*, 159–195. [CrossRef] [PubMed]
16. Suttorp, T.; Hansen, N.; Igel, C. Efficient covariance matrix update for variable metric evolution strategies. *Mach. Learn.* **2009**, *75*, 167–197. [CrossRef]
17. Salimans, T.; Ho, J.; Chen, X.; Sidor, S.; Sutskever, I. Evolution Strategies as a Scalable Alternative to Reinforcement Learning. *arXiv* **2017**, arXiv:1703.03864.