International Journal of Electrical and Computer Engineering (IJECE)
Vol. 13, No. 5, October 2023, pp. 5707∼5716
ISSN: 2088-8708, DOI: 10.11591/ijece.v13i5.pp5707-5716

A deep reinforcement learning strategy for autonomous robot flocking

Fredy Martínez, Holman Montiel, Luis Wanumen
Facultad Tecnológica, Universidad Distrital Francisco José de Caldas, Bogotá D.C, Colombia

Article Info

Article history:

Received Dec 17, 2022
Revised Mar 10, 2023
Accepted Mar 28, 2023

Keywords:

Deep deterministic policy gradient
Deep reinforcement learning
Multi-agent learning
Self-organised flocking
Swarm robotics

ABSTRACT

Social behaviors in animals such as bees, ants, and birds have shown high levels of intelligence from a multi-agent system perspective. They present viable solutions to real-world problems, particularly in navigating constrained environments with simple robotic platforms. Among these behaviors is swarm flocking, which has been extensively studied for this purpose. Flocking algorithms have been developed from basic behavioral rules, which often require parameter tuning for specific applications. However, the lack of a general formulation for tuning has made these strategies difficult to implement in various real conditions, and even to replicate laboratory behaviors. In this paper, we propose a flocking scheme for small autonomous robots that can self-learn in dynamic environments, derived from a deep reinforcement learning process. Our approach achieves flocking independently of population size and environmental characteristics, with minimal external intervention. Our multi-agent system model considers each agent's action as a linear function dynamically adjusting the motion according to interactions with other agents and the environment. Our strategy is an important contribution toward real-world flocking implementation. We demonstrate that our approach allows for autonomous flocking in the system without requiring specific parameter tuning, making it ideal for applications where there is a need for simple robotic platforms to navigate in dynamic environments.

Corresponding Author:

Fredy Martínez
Facultad Tecnológica, Universidad Distrital Francisco José de Caldas
Carrera 7 No 40B-53, Bogotá D.C., Colombia
Email: fhmartinezs@udistrital.edu.co

1. INTRODUCTION

Traditional robotic application strategies in motion planning with a single robot have serious application drawbacks on many real problems [1]. These strategies rely on single hardware with powerful sensors, processors, and communication units, which, apart from being expensive and unfeasible in some cases, are inadequate in real practical situations where the environment restricts the use of sensors (indoor environments, with high flows or other characteristics that prevent their use) or where it is complex to calculate a model of the application that allows the theoretical design of the system [2]. These problems, therefore, present open challenges and fields of research with the active participation of the community [3], [4]. Some of the solutions proposed for these cases are based on swarm robotics [5]. Under this approach, the robot is replaced by a group of agents of much simpler and cheaper design, and with limited processing and sensing capabilities, but with the possibility of self-organizing into a multi-agent system, inspired by behaviors of nature, whose organization allows it to adapt to the conditions of the environment and the task. The Flocking Behavior is one of these dynamics derived from nature, in particular from birds in migratory processes, and has been modeled from some basic behaviors that allow functional replication of these systems [6]. Several experiments have shown the ability to flock dynamics to solve problems in complex environments, as in the case of tasks in aquatic and aerial environments [7], [8].

The basic rules of behavior that achieve flocking of a multi-agent system were postulated by Reynolds in 1986 [9]. These basic rules establish criteria for the motion strategy of each agent in the system to avoid the collision, perform velocity matching according to the motion of its neighbors, and define a flocking axis [10]. The performance of each of these three basic behaviors is measured by metrics on the area of the group and its polarization [11]. It should be clarified, however, that this dynamic corresponds to the cluster flocking strategy, which prioritizes the distance between agents, so it organizes the system along an area and has a center of movement that defines the navigation, but there is another strategy such as line flocking in which the agents move in V-shape as geese do when migrating. Although the behavior of the system is different, it has been shown that the same basic rules of cluster flocking can give rise to line flocking, so it can be considered a special case in which the shape of the group is more important than the distance between agents [12]. Other approaches that reflect flocking behavior without strictly adhering to the basic Reynolds behavior rules, such as the use of local readings unrelated to the agents in the system to define the individual motion strategy [13], or the use of the anisotropy of the angle between neighbors as a parameter to estimate the flock structure, have been developed [14].

In any of the approaches adopted to achieve flocking behavior, but in particular, in the required adjustment of the parameters of each basic behavior defined by Reynolds, it is necessary to perform a manual calibration so that the desired collective behavior emerges in the system [15]. To date, these schemes require manual adjustment to pseudo-optimal parameters whose tuning has not been generalized from the design, architecture, and desired behavior of the system [16], [17]. This research addresses this problem by formulating a scheme that achieves automatic parameter tuning for specific flocking conditions, to be able to tune the parameters of a flocking behavior without human intervention [18].

We present a learning algorithm that combines deep learning and reinforcement learning, which achieves flocking self-organization of a multi-agent system in a continuous space learning process without the need for manual manipulation of parameters [19], [20]. To train a swarm model of robots, we use the deep deterministic policy gradient (DDPG) algorithm, which optimizes a reward function to correct the position of each robot until the system shows cluster flocking, a dynamic that maintains its stability [21]. Our deep reinforcement learning (DRL) scheme enables system agents to learn to perform actions directly in the environment and maximize a reward function that considers both the navigation environment and the agent itself through continuous interaction [22]. We derive the agent's actions from a policy that maps the state to action using (1).

$$\pi : \; s_t \longrightarrow a_t \tag{1}$$

Where s_t corresponds to the observation made at instant t by the agent in the environment based on its sensors, and from which an action a_t is produced. This interaction generates a reward r_{t+1}, as well as a new observation s_{t+1}. The observations do not correspond to the state of the environment, since in general, the agent is unable to fully sense the environment [23]. The observations correspond to a subset of the state, which may be constrained by the capabilities of the sensors. The objective of the algorithm is to find the optimal policy $\pi*$ maximizing the cumulative reward over time (2) [24].

$$\underset{\pi \in \prod}{max} \, \mathbb{E} \left\{ \sum_{t=0}^{T} R\left(s_t, x^{\pi}\left(s_t\right)\right) \mid s_0 \right\} \tag{2}$$

To solve (2) it is necessary to perform either a policy iteration, a value iteration, or even an interaction process that combines the two. This is where the deep learning strategy of our algorithm steps in to avoid this iterative process [25], [26]. The goal is to identify a state-action pair of the form $Q(s, a)$ (Q-value) instead of finding a value for each state according to (2). The assignment of action to a state is done using a deep neural network, which results in a strategy suitable for real problems in which this relationship is complex. The algorithm capable of performing this task is known as deep Q-network (DQN) [27].

The DDPG algorithm is a reinforcement learning technique that combines DQN with policy gradients [21], a learning technique based on increasing the probabilities of actions that lead to higher performance while reducing the probabilities of actions that lead to lower performance until an optimal policy is reached [28]. This combination generates a strategy that is capable of producing continuous actions from high-dimensional input signals, and with complex state-action relationships. This is the reason for its use as an auto-tuning strategy in our research.

2. PROBLEM STATEMENT

The multi-agent system is composed of n agents (robots in its physical implementation) in a closed environment $\mathcal{W} \subset \mathbb{R}^2$ and bounded by $\partial\mathcal{W}$, with finite obstacles that conform to a set $\mathcal{O}$ corresponding to areas inaccessible to the robots, and a free navigation space E defined by $\mathcal{W} - \mathcal{O}$. The behavior of the system emerges as a consequence of the behavior of each agent, which is determined by it autonomously (without a central control unit), as the combination of a finite set of sub behaviors designed to make the robot move to destination point, avoiding obstacles, collisions with other agents, and maintaining the formation of the system. The agent's behavior is defined by a vector that specifies the agent's velocity $\dot{\boldsymbol{x}}_i\,(t)$ and orientation $\theta_i\,(t)$. These parameters are tuned based on the system's behavioral policy and information sensed in the environment, which includes interaction with other agents.

As a dynamic interaction scheme in the multi-agent system, the strategy called wild motion [29] was selected, which is easy to implement in low-cost hardware with limited sensing, and natural to the behavior of agents (the interaction does not produce an exchange of orientation information, such as occurs with flocks of birds). Under this idea, agents move in the environment more or less in a straight line until they make contact with other neighbors, obstacles, and boundaries of the environment. When this happens, the agent randomly selects a forward direction to avoid the collision and continues moving in a straight line. This motion primitive is inspired by dynamic billiards and does not produce any exchange of orientation information between agents. Moreover, for its implementation, it requires only low-cost contact sensors.

In the model, the agents move at a velocity $\dot{\boldsymbol{x}}_i\,(t)$ that depends on their historical behavior and their position relative to nearby agents, as long as there is no collision. In the event of a collision, the agent stops and restarts its displacement at the same previous velocity, but in a new direction. If there is no collision, the velocity and orientation are defined from the distance in direct line to the nearby agents, which according to their geometric position also establishes the orientation of the agent. This strategy is shown in detail in [30] and requires a distance sensor that can be implemented with a laser distance sensor (LiDAR). This is a simple scheme oriented to maintain the structure of the system by prioritizing the distance between agents and producing a linear behavior in the movement of the agent concerning the distance to its neighbors. The strategy can be complemented with other movement criteria, or replaced if self-organization of the system is sought from non-geometrical criteria.

According to this scheme, the maximum displacement velocity is achieved when the agent orientation coincides with the system displacement direction. Therefore, the ideal agent displacement condition requires that the orientation of each agent be as close as possible to the orientation of the system motion, which corresponds to the convergence criterion of the flocking strategy. How this convergence criterion is assumed is by defining an alignment parameter, called Alignment $\mathcal{A}$, which corresponds to the average alignment value of all the agents of the system, which is compared with the reference value defined by the navigation path of the system. The ideal configuration of the system should aim for each agent to be oriented as well as possible with this reference for the entire duration of the task. If this is not met, navigation takes much longer, which is used as a reward and penalty criterion.

The learning architecture proposed for our model contemplates the DDPG scheme as shown in Figure 1. This is a learning algorithm for continuous actions that combines ideas from deterministic policy gradient (DPG) and DQN slow learning objective networks. As in the case of the actor-critic method, the scheme has two loops, the actor that proposes an action a_t according to the state s_t, and the critic that defines whether the action is correct (positive value) or incorrect (negative value) from the state and the action. The DDPG scheme, unlike the original DQN, uses two target networks to improve training stability (estimated targets), and experience replay which allows it to learn from all previous experiences, not only from the most recent one. The basic steps are described in algorithm 1.

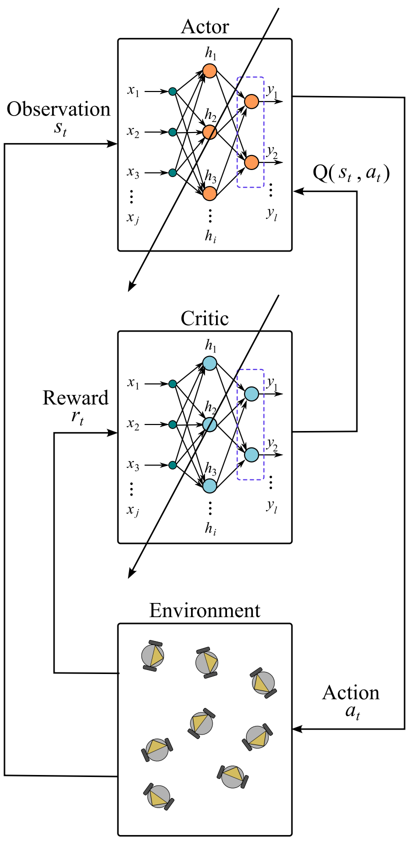

Figure 1. Architecture of the proposed learning scheme

Algorithm 1. Pseudo code for the proposed flocking strategy based on DDPG

```
Require: For each robot in the system:
  Initialize neural network for approximating action-value function $Q$ and policy function $\pi$
  Initialize experience replay buffer $R$
  Initialize target networks $Q$’ and $\pi$’
  for each iteration of the learning process: do
      for each robot in the system: do
          Observe current state $s$
          Select action $a$ using $\pi$ and exploration noise
          Execute action $a$ and observe resulting state $s$’ and reward $r$
          Store transition ($s$, $a$, $s$’, $r$) in replay buffer $R$
          Sample random mini-batch of transitions ($s$, $a$, $s$’, $r$) from $R$
          Update $Q$ and $\pi$ using the sampled transitions and the DDPG algorithm
          Update target networks $Q$’ and $\pi$’ using Polyak averaging
      end for
  end for
```

3. METHOD

Our categorization models for the actor and the critic use convolutional neural networks, therefore, the data were encoded mimicking the structure of color images (three matrices corresponding to the three color channels). The state space of the multi-agent system is then represented by three matrices, where each position corresponds to an agent of the system characterized by three values (two-dimensional pose and orientation, equivalent to the three color channels). With this design, the population size was restricted to values yielding square matrices, i.e., populations of n =1, 4, 9, 16, ..., k^2, where k corresponds to the number of rows and columns of each matrix. The three parameters of each agent are, the first channel for x-coordinate, the second channel for y-coordinate, and the third channel for θ angle. The tests were performed with a population size of 100 agents (matrices of 10×10).

The convolutional models for the actor and the critic have a similar structure to the LeNet network (Figures 2 and 3). We use two 2D convolution layers with rectified linear unit (ReLU) activation function followed each by a 2D Max pooling layer, bringing the dimensionality to a (2,2,50) array. This is followed by a dense multilayer network of 500 hidden neurons. The last layer of these networks uses a Sigmoid activation function to guarantee symmetry in the output actions. The Actor network produces motion actions for each agent defined as the velocity components on the x-axis and on the y-axis, therefore the network has a total of 2 output nodes (200 nodes in the evaluated system, for a total of 227,270 parameters to be set). On the other hand, the output of the critic network has only one output node corresponding to the Q value, which generates a total of 125,571 parameters.

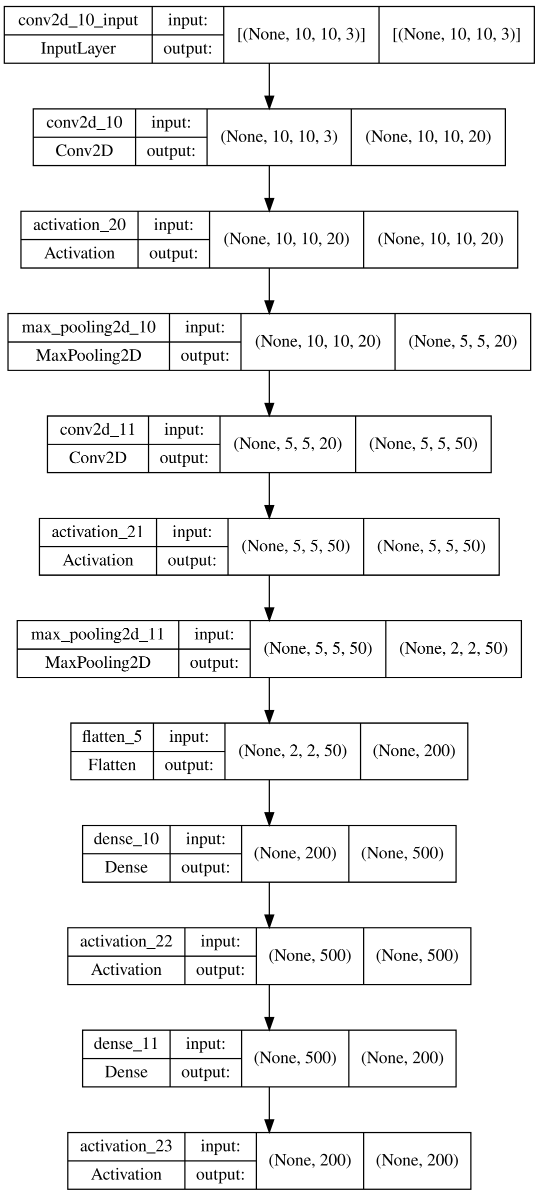

Figure 2. Architecture for the Actor convolutional network

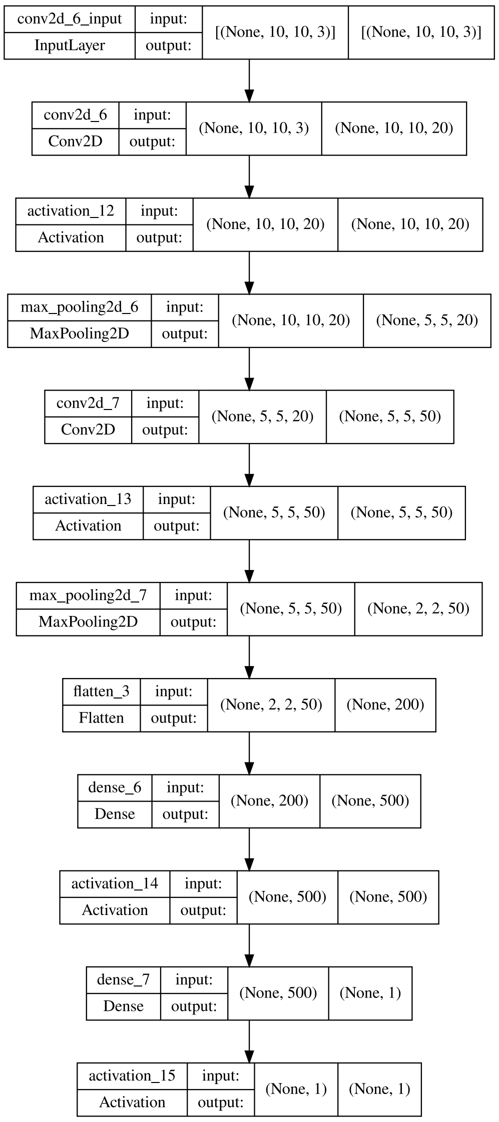
conv2d_6_input | input: | [(None, 10, 10, 3)] | [(None, 10, 10, 3)]
InputLayer | output:
conv2d_6 | input: | (None, 10, 10, 3) | (None, 10, 10, 20)
Conv2D | output:
activation_12 | input: | (None, 10, 10, 20) | (None, 10, 10, 20)
Activation | output:
max_pooling2d_6 | input: | (None, 10, 10, 20) | (None, 5, 5, 20)
MaxPooling2D | output:
conv2d_7 | input: | (None, 5, 5, 20) | (None, 5, 5, 50)
Conv2D | output:
activation_13 | input: | (None, 5, 5, 50) | (None, 5, 5, 50)
Activation | output:
max_pooling2d_7 | input: | (None, 5, 5, 50) | (None, 2, 2, 50)
MaxPooling2D | output:
flatten_3 | input: | (None, 2, 2, 50) | (None, 200)
Flatten | output:
dense_6 | input: | (None, 200) | (None, 500)
Dense | output:
activation_14 | input: | (None, 500) | (None, 500)
Activation | output:
dense_7 | input: | (None, 500) | (None, 1)
Dense | output:
activation_15 | input: | (None, 1) | (None, 1)
Activation | output:

Figure 3. Architecture for the critic convolutional network

For the reward value R, a function was designed to evaluate the alignment of the agents concerning the navigation of the system, while considering the actions of the Actor 3). This is achieved by incorporating into the reward a term that moves in the opposite direction to the alignment error, and a term that makes an average contribution from the Actor's actions. The idea of the learning process is to maximize the value of this reward, which means finding values for the set of parameters so that the system moves together as a flock along the path defined in E in the shortest possible time. The interaction dynamics guarantee the structure of the system while learning yields the behavioral values that minimize the total navigation time.

$$R = R_0 - \frac{\alpha}{\mathcal{A}_r - \mathcal{A}} + \frac{\beta}{n}\sum_{i=1}^{n} a_i \tag{3}$$

4. RESULT AND DISCUSSION

The system used for performance evaluation has a population of $n = 100$ agents, and the TurtleBot 3 Burger robot from ROBOTIS was used as a reference model due to its availability in the laboratory. The training of the scheme was performed for 50 epochs, each of which had a duration of 1,000 steps. For the two convolutional networks, a learning rate of 3×10^{-4}, and a batch size of 32 were used as the Adam optimization function. The α and β constants of (3) were adjusted by trial and error over several simulations to favor convergence and reward values. The code was developed in Python 3.7.13 with support for PyTorch 1.12.1, python-opengl 3.1.0, xvfb 1.19.6, pyvirtualdisplay 3.0, piglet 1.0.0, and gym 0.17.3, on an Intel Core i7-7700HQ 64-bit machine with Linux MX-21.1_x64 kernel 5.18.0.

Figures 4 and 5 show the behavior of the system throughout the training. Figure 4 shows the reward function, which at the end of training reaches a maximum value of 200 (maximum reward set in the strategy). It is observed not only that it reaches this maximum value, but also that it maintains it in a stationary way (i.e., the algorithm converges). This maximum value is reached on several previous occasions, but in these cases the value was not maintained, falling according to the dynamics of the system. It is important to note that the reward function contemplates an initial constant value R_0 assigned to avoid saturations at very low values. Many of the simulations involved different values for R_0, and in some cases, momentary reward values were lower than R_0 (as shown in Figure 4), yet in all cases, convergence to a stationary value was reached (usually after 30 epochs). These behaviors indicate that the algorithm always converges, and that convergence is independent of the initial reward value.

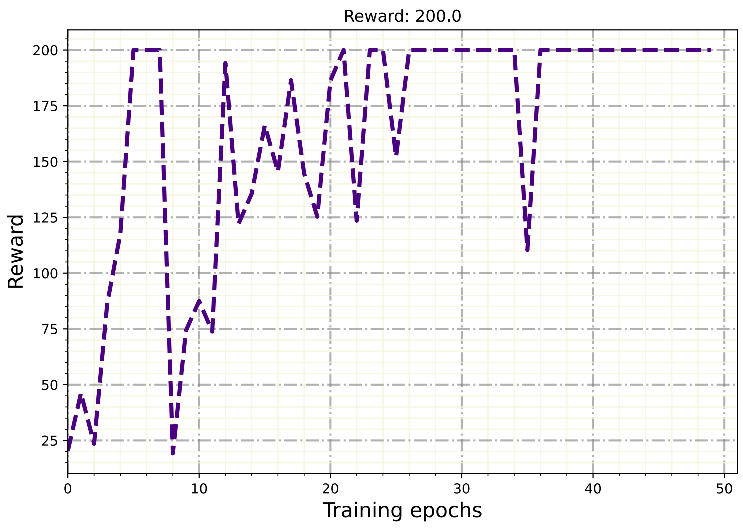

Figure 4. Rewards during the learning process

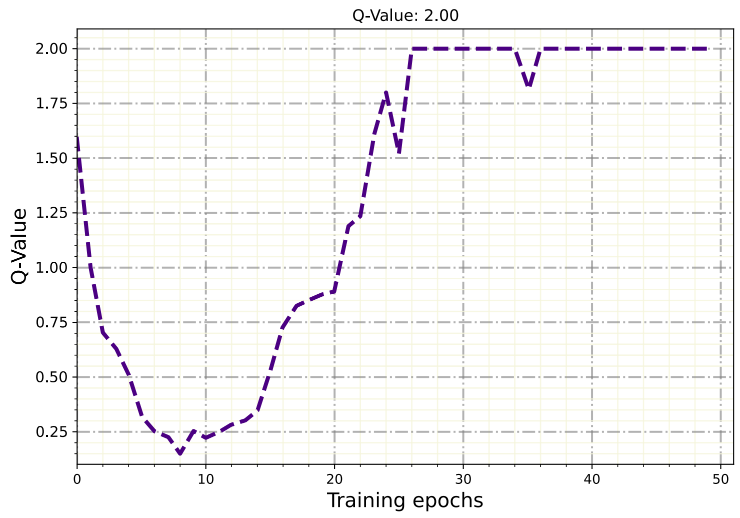

Figure 5. Q-Value during the learning process

The behavior of the Q-Value shown in Figure 5 is consistent with the reward curve above. This value represents the utility of the applied action and should drive the value of the future reward. During the early training epochs, this value drops and remains low during erratic reward behavior. However, typically after the first 20 epochs, this value consistently increases to its maximum value (set at 2.0), which at the same time stabilizes the reward value. These behaviors shift over time according to the values of the α and β constants. The weighted value of the actions in the reward equation was introduced to prevent the Actor from producing a very small action that would make convergence difficult. It was observed that this term causes the reward value to oscillate as the Q-Value decreases, so it is assumed that very strong inputs make it difficult to learn the algorithm. In our tests we evaluated values of β between 0.01 and 0.5, the best values were close to 0.1.

5. CONCLUSION

In this paper, we propose a general training strategy to achieve the flocking of a swarm of small autonomous robots under the principle of decentralized autonomous control. This strategy assumes the behavior of the swarm as a single system over which a global navigation strategy is defined along the free space of an environment, and learning capability for the robots using deep deep forcement learning (RL), which dynamically learn their motion strategy from the interaction. Under these conditions, the strategy turns out to apply to any multi-agent system regardless of its population size, as well as the characteristics of the environment. The algorithm uses the DDPG scheme for learning, which is an improvement over the Actor-Critic model since it incorporates DQN with Policy gradients to improve its stability and performance. DDPG is a reinforcement learning technique that uses two loops, the Actor loop (loop that takes the state as input and emits the action as output) and the Critic loop (loop that takes the state and the action as input and produces a Q-value of the action utility) that produces a continuous action, and in which the Actor and the Critic are implemented with convolutional networks. We use a similar architecture for these two blocks with five depth layers (two of them 2D convolutional) and continuous output via the sigmoid activation function. The system environment consisted of 100 robots with characteristics derived from the popular TurtleBot 3 Burger platform. As motion dynamics, a geometric interaction strategy was established under which robots avoid obstacles and other robots during collisions, and search for a relative position within the swarm based on the location of other neighboring robots. The reward function was constructed from the alignment error of each agent concerning the motion of the system, also considering the cumulative behavior of the actions. The test results demonstrated that the RL scheme can be successfully used to optimize the system parameters while optimizing the resource consumption derived from task time and distance traveled. It is also shown that RL schemes help to tune complex multi-agent schemes in a quasi-optimal way.

ACKNOWLEDGEMENT

This work was supported by the Universidad Distrital Francisco José de Caldas, in part through CIDC, and partly by the Facultad Tecnológica. The views expressed in this paper are not necessarily endorsed by Universidad Distrital. The authors thank the research group ARMOS for the evaluation carried out on prototypes of ideas and strategies.

REFERENCES

[1] B. Aminof, A. Murano, S. Rubin, and F. Zuleger, “Verification of agent navigation in partially-known environments,” *Artificial Intelligence*, vol. 308, no. 103724, Jul. 2022, doi: 10.1016/j.artint.2022.103724.

[2] T. Alam, L. Bobadilla, and D. A. Shell, “Minimalist robot navigation and coverage using a dynamical system approach,” in *2017 First IEEE International Conference on Robotic Computing (IRC)*, Apr. 2017, pp. 249–256, doi: 10.1109/IRC.2017.23.

[3] S. He, R. Xu, Z. Zhao, and T. Zou, “Vision-based neural formation tracking control of multiple autonomous vehicles with visibility and performance constraints,” *Neurocomputing*, vol. 492, no. 1, pp. 651–663, Jul. 2022, doi: 10.1016/j.neucom.2021.12.056.

[4] A. Ugenti, F. Vulpi, R. Domínguez, F. Cordes, A. Milella, and G. Reina, “On the role of feature and signal selection for terrain learning in planetary exploration robots,” *Journal of Field Robotics*, vol. 39, no. 4, pp. 355–370, Jun. 2022, doi: 10.1002/rob.22054.

[5] R. K. Ramachandran, P. Pierpaoli, M. Egerstedt, and G. S. Ssukhatme, “Resilient monitoring in heterogeneous multi-

robot systems through network reconfiguration," *IEEE Transactions on Robotics*, vol. 38, no. 1, pp. 126–138, 2022, doi: 10.1109/TRO.2021.3128313.

[6] F. Martínez, "Review of flocking organization strategies for robot swarms," *Tekhnê*, vol. 18, no. 1, pp. 13–20, 2021.

[7] R. Bailon-Ruiz, A. Bit-Monnot, and S. Lacroix, "Real-time wildfire monitoring with a fleet of UAVs," *Robotics and Autonomous Systems*, vol. 152, no. 1, Jun. 2022, doi: 10.1016/j.robot.2022.104071.

[8] T. Elmokadem and A. V. Savkin, "A method for autonomous collision-free navigation of a quadrotor UAV in unknown tunnel-like environments," *Robotica*, vol. 40, no. 4, pp. 835–861, Apr. 2022, doi: 10.1017/S0263574721000849.

[9] C. W. Reynolds, "Flocks, herds and schools: A distributed behavioral model," *ACM SIGGRAPH Computer Graphics*, vol. 21, no. 4, pp. 25–34, Aug. 1987, doi: 10.1145/37402.37406.

[10] A. A. Paranjape, S. J. Chung, K. Kim, and D. H. Shim, "Robotic herding of a flock of birds using an unmanned aerial vehicle," *IEEE Transactions on Robotics*, vol. 34, no. 4, pp. 901–915, 2018, doi: 10.1109/TRO.2018.2853610.

[11] F. H. Martínez, "TurtleBot3 robot operation for navigation applications using ROS," *Tekhnê*, vol. 18, no. 2, pp. 19–24, 2021.

[12] J. Yang, R. Grosu, S. A. Smolka, and A. Tiwari, "Love thy neighbor: V-formation as a problem of model predictive control," in *Leibniz International Proceedings in Informatics*, 2016, vol. 59, doi: 10.4230/LIPIcs.CONCUR.2016.4.

[13] R. Vatankhah, S. Etemadi, M. Honarvar, A. Alasty, M. Boroushaki, and G. Vossoughi, "Online velocity optimization of robotic swarm flocking using particle swarm optimization (PSO) method," in *2009 6th International Symposium on Mechatronics and its Applications*, Mar. 2009, pp. 1–6, doi: 10.1109/ISMA.2009.5164776.

[14] M. Makiguchi and J. I. Inoue, "Anisotropy measurement in artificial flockings: how does one design the optimal BOIDS by evolutionary computation?," in *SCIS and ISIS 2010 - Joint 5th International Conference on Soft Computing and Intelligent Systems and 11th International Symposium on Advanced Intelligent Systems*, 2010, pp. 1003–1008.

[15] J.-G. Dong, "Avoiding collisions and pattern formation in flocks," *SIAM Journal on Applied Mathematics*, vol. 81, no. 5, pp. 2111–2129, Jan. 2021, doi: 10.1137/21M1390141.

[16] D. N. M. Hoang, D. M. Tran, T.-S. Tran, and H.-A. Pham, "An adaptive weighting mechanism for Reynolds rules-based flocking control scheme," *PeerJ Computer Science*, vol. 7, no. 2021, Feb. 2021, doi: 10.7717/peerj-cs.388.

[17] T. Ibuki, S. Wilson, J. Yamauchi, M. Fujita, and M. Egerstedt, "Optimization-based distributed flocking control for multiple rigid bodies," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 1891–1898, 2020, doi: 10.1109/LRA.2020.2969950.

[18] E. Jacinto, F. Martinez, and F. Martinez, "Navigation of autonomous vehicles using reinforcement learning with generalized advantage estimation," *International Journal of Advanced Computer Science and Applications*, vol. 14, no. 1, pp. 954–959, 2023, doi: 10.14569/IJACSA.2023.01401103.

[19] W. Wang, L. Wang, J. Wu, X. Tao, and H. Wu, "Oracle-guided deep reinforcement learning for large-scale multi-UAVs flocking and navigation," *IEEE Transactions on Vehicular Technology*, vol. 71, no. 10, pp. 10280–10292, 2022, doi: 10.1109/TVT.2022.3184043.

[20] G. Shen *et al.*, "Deep reinforcement learning for flocking motion of multi-UAV systems: learn from a digital twin," *IEEE Internet of Things Journal*, vol. 9, no. 13, pp. 11141–11153, Jul. 2022, doi: 10.1109/JIOT.2021.3127873.

[21] T. P. Lillicrap *et al.*, "Continuous control with deep reinforcement learning," in *4th International Conference on Learning Representations*, Sep. 2015.

[22] V. François-Lavet, P. Henderson, R. Islam, M. G. Bellemare, and J. Pineau, "An introduction to deep reinforcement learning," *Foundations and Trends® in Machine Learning*, vol. 11, no. 3–4, pp. 219–354, 2018, doi: 10.1561/2200000071.

[23] H. Bae, G. Kim, J. Kim, D. Qian, and S. Lee, "Multi-robot path planning method using reinforcement learning," *Applied Sciences*, vol. 9, no. 15, Jul. 2019, doi: 10.3390/app9153057.

[24] W. B. Powell, "A unified framework for stochastic optimization," *European Journal of Operational Research*, vol. 275, no. 3, pp. 795–821, Jun. 2019, doi: 10.1016/j.ejor.2018.07.014.

[25] A. Leite, M. Candadai, and E. J. Izquierdo, "Reinforcement learning beyond the Bellman equation: exploring critic objectives using evolution," in *The 2020 Conference on Artificial Life*, 2020, pp. 441–449, doi: 10.1162/isal_00338.

[26] H. Wiltzer, D. Meger, and M. G. Bellemare, "Distributional Hamilton-Jacobi-Bellman equations for continuous-time reinforcement learning," *arXiv:2205.12184*, May 2022.

[27] V. Mnih *et al.*, "Playing atari with deep reinforcement learning," *arXiv:1312.5602*, 2013.

[28] S. Dankwa and W. Zheng, "Twin-delayed DDPG," in *Proceedings of the 3rd International Conference on Vision, Image and Signal Processing*, Aug. 2019, pp. 1–5, doi: 10.1145/3387168.3387199.

[29] L. Bobadilla, F. Martinez, E. Gobst, K. Gossman, and S. M. Lavalle, "Controlling wild mobile robots using virtual gates and discrete transitions," in *Proceedings of the American Control Conference*, 2012, pp. 743–749, doi: 10.1109/acc.2012.6315569.

[30] F. Martínez, "Minimalistic control scheme for the development of search tasks with flocks of robots," *Journal of Physics: Conference Series*, vol. 1993, no. 1, Aug. 2021, doi: 10.1088/1742-6596/1993/1/012025.

BIOGRAPHIES OF AUTHORS

Fredy Martínez is a professor of control, intelligent systems, and robotics at the Universidad Distrital Francisco José de Caldas (Colombia) and director of the ARMOS research group (Modern Architectures for Power Systems). His research interests are control schemes for autonomous robots, mathematical modeling, electronic instrumentation, pattern recognition, and multi-agent systems. Martinez holds a Ph.D. in Computer and Systems Engineering from the Universidad Nacional de Colombia. He can be contacted at email: fhmartinezs@udistrital.edu.co.

Holman Montiel is a professor of algorithms, embedded systems, instrumentation, telecommunications, and computer security at the Universidad Distrital Francisco José de Caldas (Colombia) and a researcher in the ARMOS research group (Modern Architectures for Power Systems). His research interests are encryption schemes, embedded systems, electronic instrumentation, and telecommunications. Montiel holds a master's degree in computer security. He can be contacted at email: hmontiela@udistrital.edu.co.

Luis Wanumen is a professor of Software Engineering at the Universidad Distrital Francisco José de Caldas (Colombia) and a researcher in the METIS research group (Organizational Informatics Research Group). His research interests are database design, advanced programming and systems analysis. Mr. Wanumen holds a Master's degree in Computer and Systems Engineering. He can be contacted at email: lwanumen@udistrital.edu.co.