Proceedings of the 2024 IEEE
International Conference on Robotics and Biomimetics
December 10-14, 2024, Bangkok, Thailand

Learning NEAT Emergent Behaviors in Robot Swarms

Pranav Rajbhandari[1] and Donald Sofge[2]

Abstract— **When researching robot swarms, many studies observe complex group behavior emerging from the individual agents' simple local actions. However, the task of learning an individual policy to produce a desired group behavior remains a challenging problem. We present a method of training distributed robotic swarm algorithms to produce emergent behavior. Inspired by the biological evolution of emergent behavior in animals, we use an evolutionary algorithm to train a 'population' of individual behaviors to produce a desired group behavior. We perform experiments using simulations of the Georgia Tech Miniature Autonomous Blimps (GT-MABs) aerial robotics platforms conducted in the CoppeliaSim simulator. Additionally, we test on simulations of Anki Vector robots to display our algorithm's effectiveness on various modes of actuation. We evaluate our algorithm on various tasks where a somewhat complex group behavior is required for success. These tasks include an Area Coverage task and a Wall Climb task. We compare behaviors evolved using our algorithm against *designed policies*, which we create in order to exhibit the emergent behaviors we desire.**

I. Introduction

A. Emergent Behavior

Emergent behavior is a phenomenon observed in swarms of agents. It is generally defined as a complex swarm behavior which occurs as a consequence of each individual agent following a relatively simple control scheme [16], [18]. Examples of this can be found in the behavior of groups of animals, such as how some species of fire ants have been observed to create rafts out of their bodies to survive flooding [14]. Emergent behavior is also exhibited in migrating swarms of some species of caterpillars, which walk on top of each other in order to create a 'rolling swarm' [22]. This allows the caterpillars to migrate quicker than simply walking. Similar instances of complex emergent behavior have been observed in the field of swarm robotics.

B. Swarm Robotics

Though there are various definitions of swarm robotics, it is generally agreed that robot swarms are multi-agent systems where each agent is autonomous and has low complexity [3], [10], [15]. They are used for various tasks, including exploration and surveillance. They are characterized by using locally communicating distributed systems as opposed to a central controller. Due to this, they remain functional as the swarm size is increased. However, they do not have a central controller, so careful fine tuning of the individual policies is required to achieve a desired swarm behavior.

It is well known that emergent behavior can arise in robot swarms, and past studies have explored this. Pagello et al. [19] study how to make a robot swarm perform a cooperative task by creating emergent behaviors. They implement this by dynamically assigning predefined roles to each robot. A function Q is learned for each agent, which takes in local information and outputs the best role for the agent to adopt. After refining their Q functions, they observe cooperative emergent behavior in their chosen task, robot soccer.

In a more recent study, Oliveri et al. [17] use a Monte Carlo scheme to continuously refine the behavior of individual agents in a swarm. They tested their method on robots connected in a line, which were each able to push away from their neighbors. Their study concluded that training each agent to optimize its individual velocity resulted in the entire group of connected robots crawls forward.

C. Evolutionary Algorithms/NEAT

Evolutionary algorithms are algorithms inspired by biological evolution, where a user defined *fitness function* is optimized by repeatedly transforming a 'population' of solutions by spawning new members similar to members with the best fitness. We inspect NeuroEvolution (NE), a type of evolutionary algorithm that evolves neural networks. In early NE algorithms, the topology[1] of the neural network is fixed, and the mutations take place in the weights of the connections [24]. However, using the Neuroevolution of Augmenting Topologies (NEAT) algorithm, the topology of the neural network can evolve as well, theoretically allowing generalization to problems of arbitrary complexity [25].

D. Organization

The rest of the paper is organized as follows: Section II discusses related work. Section III details our proposed learning algorithm, as well as common sensing and output schemes we use. We present our experiments in Section IV, and discuss the results in Section V. The conclusions we draw are in Section VI.

II. Related Work

Much research has focused on the goal of learning individual behaviors for a robot swarm [2], [3], [5], [6], [7], [17], [19]. Behjat et al. [2] explore how to learn tactical swarm behavior through a combination of various techniques. These

[1]Pranav Rajbhandari is the corresponding author and with Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA, prajbhan@alumni.cmu.edu. They completed this work under NREIP at Naval Research Laboratory, Washington D.C., USA.

[2]Donald Sofge is with the Naval Research Laboratory, Washington D.C., USA, donald.a.sofge.civ@us.navy.mil.

[1]The topology of a neural network refers to the graph structure of the nodes and the connections between them

2024 IEEE International Conference on Robotics and Biomimetics (ROBIO) | 979-8-3315-0964-4/24/$31.00 ©2024 IEEE | DOI: 10.1109/ROBIO64047.2024.10907512

Authorized licensed use limited to: University of New South Wales. Downloaded on August 01,2025 at 01:50:23 UTC from IEEE Xplore. Restrictions apply.

techniques include learning neural network based robot policies, dynamically organizing the swarm into groups, and Pareto filtering of points of interest to reduce the problem dimensionality.

Fan et al. [5] compare different swarm intelligence algorithms against each other to evaluate their relative performance in obstacle performance under different circumstances. The algorithms they evaluate are the bat algorithm (BA), particle swarm optimization (PSO), and the grey wolf optimizer (GWO). They find that PSO outperforms BA, which outperforms GWO in general. However, GWO performs better than the other two algorithms in the case of large swarms and large communication ranges.

The AutoMoDe algorithm, designed by Francesca et al., defines an agent policy as a finite state machine whose states are preexisting *constituent behaviors* [6]. The optimization process applies the F-race algorithm [11] to learn transitions between constituent behaviors based on features observed from the environment. In their initial paper, they successfully train robots to perform an aggregation task.

Dorigo et al. [7] explore how to create communication protocols between agents in a swarm that best help methods such as deep reinforcement learning create good decentralized control policies.

In addition to the studies by Oliveri and Pagello which focus on the learning aspect (described in Section I-B), there has also been research on helping the stability of emergent behaviors in swarms. In 2022, Chen and Ng explore secure communications between robots in a swarm [3]. They model these communications as a series of random graphs, and use a method involving hash chains to identify 'rogue robots' with high probability. This allows them to ensure that the emergent behavior exhibited by the swarm is protected. In their paper, they also create a system to distinguish different classes of robot swarms by identifying differences in the robot homogeneity, the interactions between robots, and the interactions with a central control.

Our work is closely related to research done by Trianni et al. in 2003 [28]. They use an evolutionary algorithm to evolve an 'aggregation' behavior in a swarm of robots, inspired by the self-organized aggregation behavior observed in the cellular slime mold *Dictyostelium discoideum* [29]. Trianni's evolutionary algorithm uses a neural network with no hidden layers, and evolves by mutating the weights. They use a fitness function that seeks to minimize the mean distance of each agent from the swarm's center of mass. Later research by Bahceci explores this further by varying the parameters of the evolutionary algorithm to see how it affects the evolution of aggregation behaviors [1].

We expand on Trianni's idea by generalizing the algorithm to allow user-defined fitness functions. Additionally, we use the NEAT algorithm as opposed to simply evolving the weights of a set network. These extensions allow our algorithm to theoretically evolve a network to arbitrary complexity in order to create a desired emergent behavior.

We use the GT-MABs as the robots for our experiments [27]. The control system for their actuation was designed based on the dynamic model created in [26], [4].

III. Methodology

A. Evolutionary Algorithm

We propose NEAT as a candidate for learning emergent behavior in robot swarms. We will evolve a population of neural networks, evaluated through the performance of a homogeneous[2] robot swarm in one episode.

Our algorithm requires as input a user-defined fitness function. This function acts on a full episode of a robotic swarm's behavior, and returns a real number evaluating the swarm behavior.

The proposed algorithm does the following in a loop:

- For each network x in a population of neural networks, a robot swarm is initialized such that each member contains a copy of x for control.
- A full episode is run, and the output of the fitness function is used as the fitness of x.
- After fitnesses are collected, the next generation of networks is evaluated with NEAT.

We implement this on model robot swarms in a CoppeliaSim simulator [21]. For learning, we use the NEAT-Python package [13]. We use ROS2 Foxy to handle transferring sensing and actuation messages between CoppeliaSim and Python [12]. Our full implementation is available on Github [20].

1) Drawbacks/Justification: The main drawback of this method is each full episode generates one fitness score for evaluation. This is quite wasteful in comparison to alternative methods like Reinforcement Learning (RL), which use each timestep as a training example.

However, using RL would require a user-defined evaluation function to assign a value to each action of each agent based on its benefit to the swarm behavior. This seems like quite a strong restriction, and the difficulty of defining this function would greatly limit the applications of RL to this problem.

Thus, the sample inefficiency of using an evolutionary algorithm can be justified by how applicable the algorithm is.

B. Network Inputs

The primary method for control of robot swarms is to have each agent act independently on local information. This method allows generalization of learned policies to varied swarm sizes. Thus, we generally use local observations of the environment and of nearby agents for inputs to each agent's policy.

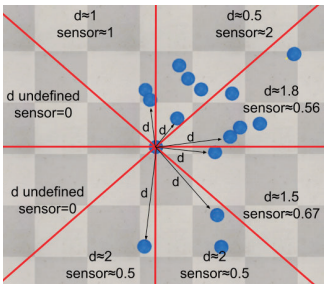
Since the position of other agents in the swarm is usually vital to training a good policy, we specify two sensing schemes that we use for our experiments.

We define *k-tant* sensing, where we split the environment into k regions based on direction relative to the agent that is sensing. We allow the agent to either sense the distance to the nearest neighbor[3] or the number of neighbors in each

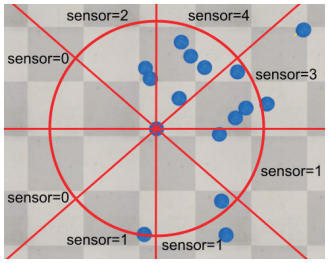
[2]Each agent is controlled with a copy of the same neural network

[3]We invert the distance so that an empty region corresponds to a sensor value of 0

Authorized licensed use limited to: University of New South Wales. Downloaded on August 01,2025 at 01:50:23 UTC from IEEE Xplore. Restrictions apply.



(a) Distance Sense



(b) Neighbor Sense

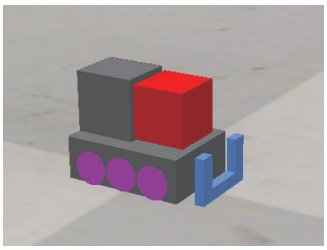
Fig. 1. k-tant sensing scheme with $k = 8$

region, as displayed in Fig. 1. Although we only use 2-D experiments, this sensing scheme very easily generalizes to 3-D (and N-D) sensing by discretizing the relative angles to each neighboring agent.

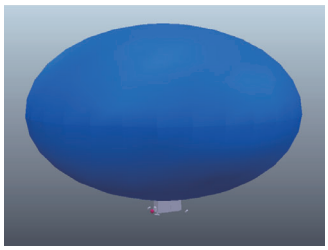
C. Network Outputs/Agent Actuation

Since the NEAT algorithm outputs vectors of any specified dimension, this algorithm can very easily be fed into a velocity controller, or even into a low-level motor controller. In our experiments, we use both methods to verify the robustness of our algorithm to output scheme.

IV. Experiments



(a) Anki Vector



(b) GT-MAB

Fig. 2. CoppeliaSim models

To test our algorithm, we use swarms of GT-MAB robots [27] and Anki Vector robots [9], simulated in CoppeliaSim (Fig. 2). We feed the neural network outputs into a velocity controller for the GT-MABs and directly into wheel velocity controllers for the Anki robots. We define various tasks for the robot swarms to complete, and train our evolutionary algorithm for 50 generations in CoppeliaSim. We terminate each episode after 60 seconds, and compare the best evolved policy's performance against a *designed policy* created to solve each task.

In our experiments, we mostly use NEAT's default hyperparameter values for the OpenAI Lunar Lander test, which was similar in complexity to our experiments. Since hyperparameters of NEAT (e.g. population size, mutation rates) tend to have a large effect on the performance of the algorithm, we plan to explore how they behave when applied to swarm control in future research.

We test in simulation so that we can verify the performance of our algorithm theoretically. We expect the transfer of these results onto physical robots will have complications, mainly due to the variability of physical sensors and actuators. In future research, we plan to evaluate this algorithm applied on the physical counterparts of our simulated robots.

A. Task: Area Coverage

In this task, we simulate a 'search and rescue' scenario. The environment we use is a square arena with the agents spawned randomly near the center (displayed in Fig. 5). For the fitness function, we adapt deployment entropy, a measure of how well distributed the agents become in the environment [8]. Deployment entropy is defined by discretizing the environment into a grid, and measuring the entropy of the distribution of agents in that grid. In the example of Fig. 3, we would calculate this value as $\sum_i -p_i \log(p_i) \approx 2.05$. Since entropy is maximized by a uniform distribution, we believe using this as a fitness function would encourage the agents to spread out in the environment, the desired behavior for a 'search and rescue' mission. We implement this task for the both the Anki robots and the GT-MABs.

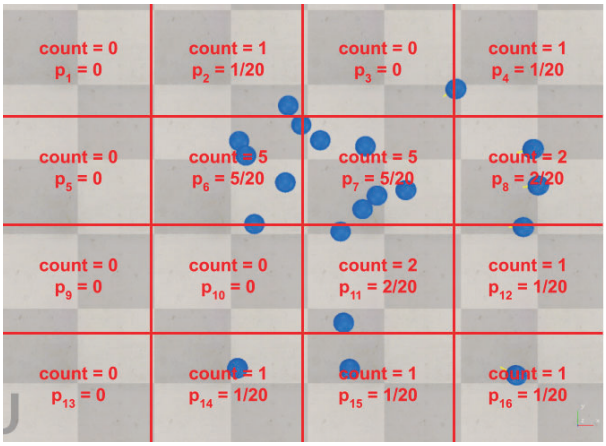


Fig. 3. Calculation of deployment entropy

For the GT-MABs, we use 20 agents and divide the environment into a grid with 16 units for calculating deployment entropy. We chose $\lfloor 20 \rfloor^2 = 16$ as this makes 1-2 agents per square optimal. For input, we use k-tant Distance Sensing with $k = 8$. We also allow the walls to be sensed on each k-tant, since this lets the agents avoid crashing into them.

For the Anki Vectors, we use 10 agents (as this number would comfortably fit in our environment) and divide the environment into $\lfloor \sqrt{10} \rfloor^2 = 9$ units for deployment entropy. For input, we use k-tant Distance Sensing ($k = 8$) along with of the Anki's onboard proximity sensor.

For both agents, we define a *designed policy* where each agent moves away from the closest neighbors. We expect this to cause the agents to distribute themselves evenly, producing the desired swarm behavior.

B. Task: Wall Climb

In previous research we established that a swarm of GT-MABs was able to climb a wall that was much taller than

Authorized licensed use limited to: University of New South Wales. Downloaded on August 01,2025 at 01:50:23 UTC from IEEE Xplore. Restrictions apply.

the altitude they were assigned to hold [23]. Upon closer inspection, we realized that that collisions were resulting in the agents registering each other as the floor, effectively stacking their desired heights. A stack of three agents would allow the top one to pass over the wall.

We also note that the 'wall climbing' behavior is susceptible to the sensing angle of the GT-MAB's ultrasound-based range sensor. With a large sensing angle, the GT-MABs are able to sense the wall itself when they are next to it, which results in a single agent being able to climb the wall on its own. To avoid this, we use a narrow ultrasound angle so that the agents must stack in order to climb the wall.

We design the Wall Climb task to replicate this scenario. The environment we use is an arena with a 3 meter tall wall on the y axis. A swarm of 20 GT-MABs spawn randomly on the right side, as shown in Fig. 8. For the fitness function, we use the number of agents that end the experiment on the other side of the wall. If no agents succeed, we subtract a penalty of the distance of the closest blimp to the wall, to speed up initial exploration. We believe using this as a fitness function will encourage the emergent 'aggregation' behavior, since this is the only way the agents can climb the wall.

For our input scheme, we experiment with both k-tant Distance Sensing and k-tant Neighbor Sensing, with $k = 8$. We compare the two experiments to display robustness of our algorithm to input scheme.

We define a *designed policy* where each agent moves towards either their closest neighbor or the direction with the most neighbors (depending on the input scheme), as well as slightly towards the direction of the wall. We expect this to cause the GT-MABs to flock and stack on top of one another, producing the desired swarm behavior of flocking over the wall. For our *designed policy*, we expect that Neighbor Sensing will perform slightly better, as this will encourage the agents to seek the largest group of neighbors.

V. Results

TABLE I

Fitness in Evolved[a] and Designed Behaviors

Experiment	Behavior	Mean	Stdev.
GT-MAB Area Coverage	Evolved	2.53	0.10
	Designed	2.62	0.12
Anki Area Coverage	Evolved	2.09	0.083
	Designed	1.87	0.20
Wall Climb (Distance Sense)	Evolved	16.70	1.12
	Designed	12.13	3.78
Wall Climb (Neighbor Sense)	Evolved	16.72	1.32
	Designed	15.33	1.58
Surround Target	Evolved	-2.21	.70
	Designed	-1.24	0.14

[a]: Using the best genome from the final generation.

For each experiment, after training for 50 generations, we compare the fitness of the best genome in the final generation with the fitness of our *designed policy*. We run each for 60 trials in their respective experiments and collect the fitness mean and standard deviation. We report our results in Table I.

A. Area Coverage

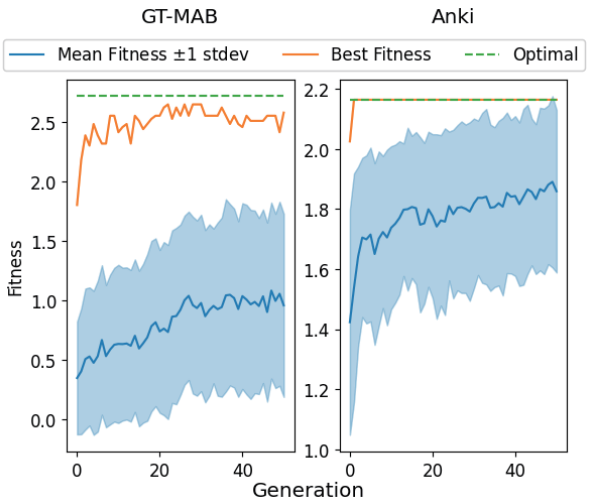
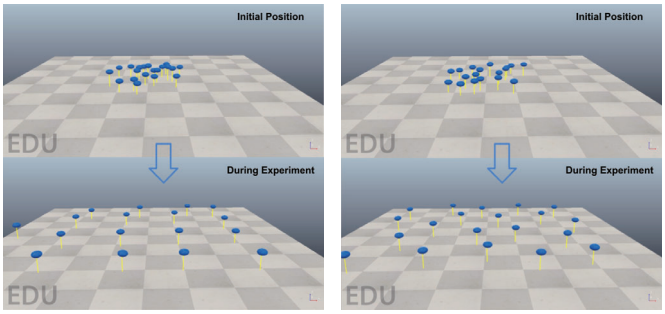
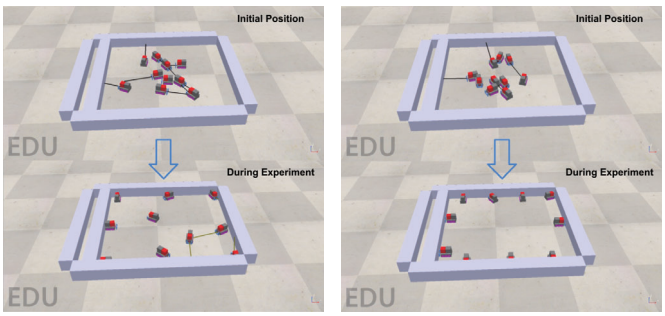


Fig. 4. Area Coverage population fitness across generations



(a) GT-MAB NEAT (b) GT-MAB *designed policy*



(c) Anki NEAT (d) Anki *designed policy*

Fig. 5. Area Coverage behaviors

After training in CoppeliaSim, we observe that the evolved behavior for both GT-MABs and Ankis seems to be for each agent to move in a direction away from its close neighbors. The GT-MABs accomplish this directly through the velocity controller, while the Ankis learned to spin in circles and reverse whenever their proximity sensor sensed an agent in front of them. For both experiments, the result seems to be a well distributed swarm (Fig. 5). From Fig. 4, we can see that both experiments generated agents that achieved close

Authorized licensed use limited to: University of New South Wales. Downloaded on August 01,2025 at 01:50:23 UTC from IEEE Xplore. Restrictions apply.

to the theoretic maximal entropy[4], with the Ankis achieving it almost immediately.

The result of the *designed policy* (Fig. 5(b, d)) also had the swarm distribute in the environment. Comparing these results in Table I, we can see that for GT-MABs, the *designed policy* outperforms the evolved behavior by about one standard deviation. For the Anki experiment, the opposite is true, with the evolved behavior outperforming the *designed policy* by about one standard deviation. This seems to be due to the *designed policy* favoring sending agents to the edges of the environment.

Overall, we show that in this task, our algorithm learns a local behavior that closely approximated the desired 'search and rescue' emergent swarm behavior. The evolved behavior greatly outperforms the *designed policy*, which shows that our evolutionary algorithm performs comparably to designing the behavior with knowledge of the task.

B. Wall Climb

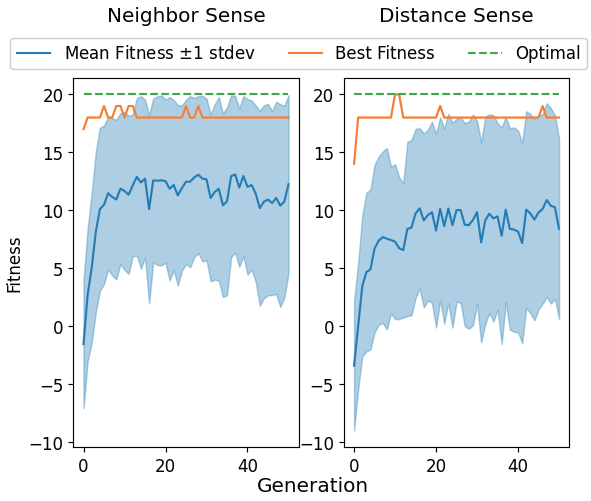
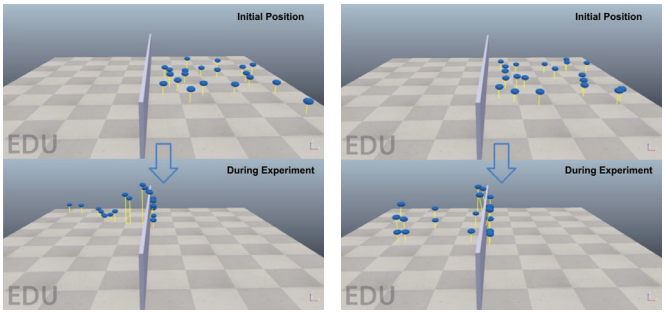


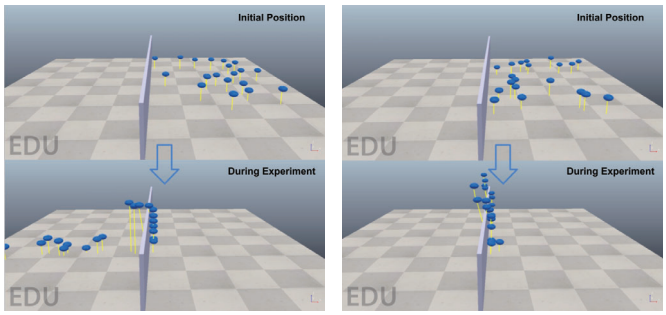
Fig. 6. Wall Climb population fitness across generations



(a) Evolved behavior (b) *Designed policy* behavior

Fig. 7. Distance Sensing Wall Climb behaviors

[4]2.718 for GT-MABs from placing one agent in 12 of the squares, and two agents in 4 of the squares; 2.16 for Ankis by placing one agent in 8 of the squares and two agents in 1 square



(a) Evolved behavior (b) *Designed policy* behavior

Fig. 8. Neighbor Sensing Wall Climb behaviors

After training the GT-MAB models in CoppeliaSim, we observe that the evolved behavior for both experiments seems to be for each agent to move in a direction towards its closest neighbors, in addition to moving in the direction to climb the wall. The result of this behavior does seem to be a flock of agents stacked on top of each other, climbing the wall. From Fig. 6, we can see that after a few generations, the best genome of each generation achieved having about 18 GT-MABs make it over the wall.

We noticed that both sensing methods achieved similar population fitness values across generations Fig. 6. From Table I, we see there is no statistically significant difference between the two fitness results.[5]

The result of the *designed policy* in both of these experiments similarly causes the agents to flock together and climb the wall. However, in this case we do notice a difference in the two modes of sensing. In the Distance Sensing experiments, the *designed policy* seems to be more likely to cause the agents to form several smaller groups as opposed to one large one. This results in a lower fitness due to more agents being left behind, as shown in Table I. Comparing this with our evolved behavior, we can see in both experiments, the evolved behavior outperforms the best *designed policy* by at least one standard deviation.

Overall, we show that in this task, our algorithm learns a local behavior that closely approximated the desired 'flocking' emergent swarm behavior. We can further see that although the different sensing modes has an effect on our *designed policy*, the evolutionary algorithm is robust to these variations.

VI. Conclusion

In this paper, we present a novel extension of the NEAT algorithm designed to learn emergent behaviors in robot swarms. The algorithm we present is robust to robot swarms with various modes of sensing and actuation. Results from

[5]We perform a two sample t-test with unequal variance on the fitnesses obtained from the evolved Neighbor Sensing and Distance Sensing experiments. Using their respective means and standard deviations of (16.70,1.12) and (16.72,1.32) with a sample size of 60 for both, we arrive at a p-value of $p > .9$, which is much larger than .05, the accepted value for statistical significance. This shows that with our sample size, there is no significant difference between the fitnesses obtained from the different sensing methods.

Authorized licensed use limited to: University of New South Wales. Downloaded on August 01,2025 at 01:50:23 UTC from IEEE Xplore. Restrictions apply.

simulations show that individual agent behaviors evolved using this method are comparable to hand designed policies at producing desired complex emergent behaviors.

We compare our algorithm against *designed policies* since the tasks we evaluate are simple and have a reasonable hard-coded approximate solution. In future work, we plan to increase the complexity of our tasks and compare against AutoMoDe or a similar algorithm. This will allow us to better evaluate the effectiveness of our method.

In future research, we plan to test our algorithm's performance on the physical GT-MABs and Anki Vector robots. We also plan to evaluate our algorithm on a more complex set of tasks. We also may explore the fine tuning of NEAT parameters to try improving our results. Additionally, the accuracy of the CoppeliaSim simulator seems to be dependant on the performance of the computer, as this determines the speed that each agent can respond to stimuli. In our experiments, we choose to speed up our experiments by running multiple simulations in parallel, which potentially caused delays in agent response time. In future experiments, less parallelization, a more powerful computer, or a different simulator could improve the stability of our algorithm.

REFERENCES

[1] Erkin Bahceci and Erol Sahin. Evolving aggregation behaviors for swarm robotic systems: a systematic case study. In *Proceedings 2005 IEEE Swarm Intelligence Symposium, 2005. SIS 2005.*, pages 333 – 340, 07 2005.

[2] Amir Behjat, Hemanth Manjunatha, Prajit Krisshna Kumar, Apurv Jani, Leighton Collins, Payam Ghassemi, Joseph Distefano, David Doermann, Karthik Dantu, Ehsan Esfahani, and Souma Chowdhury. Learning robot swarm tactics over complex adversarial environments. In *2021 International Symposium on Multi-Robot and Multi-Agent Systems (MRS)*, pages 83–91, Nov 2021.

[3] Liqun Chen and Siaw-Lynn Ng. Securing emergent behaviour in swarm robotics. *Journal of Information Security and Applications*, 64:103047, 2022.

[4] Sungjin Cho, Vivek Mishra, Qiuyang Tao, Paul Vamell, Matt King-Smith, Aneri Muni, Weston Smallwood, and Fumin Zhang. Autopilot design for a class of miniature autonomous blimps. In *2017 IEEE Conference on Control Technology and Applications (CCTA)*, pages 841–846, 2017.

[5] Jiaqi Fan, Mengqi Hu, Xianghua Chu, and Dong Yang. A comparison analysis of swarm intelligence algorithms for robot swarm learning. In *2017 Winter Simulation Conference (WSC)*, pages 3042–3053, Dec 2017.

[6] Gianpiero Francesca, Manuele Brambilla, Arne Brutschy, Vito Trianni, and Mauro Birattari. AutoMoDe: A novel approach to the automatic design of control software for robot swarms. *Swarm Intell*, 8:1–24, 06 2014.

[7] Maximilian Hüttenrauch, Adrian Šošić, and Gerhard Neumann. Local communication protocols for learning complex swarm behaviors with deep reinforcement learning. In *Swarm Intelligence*, pages 71–83, Cham, 2018. Springer International Publishing.

[8] John D. Kelly, Daniel M. Lofaro, and Donald Sofge. Persistent area coverage for swarms utilizing deployment entropy with potential fields. In *2020 17th International Conference on Ubiquitous Robots (UR)*, pages 479–486, 2020.

[9] Digital Dream Labs. Vector 2.0 ai robot companion. `https://ddlbots.com/products/vector-robot`, 2023.

[10] Thalia May Laing, Siaw-Lynn Ng, Allan Tomlinson, and Keith M Martin. Security in swarm robotics. *Handbook of Research on Design, Control, and Modeling of Swarm Robotics*, page 42–66, 2016.

[11] Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Mauro Birattari, and Thomas Stützle. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58, 2016.

[12] Steven Macenski, Tully Foote, Brian Gerkey, Chris Lalancette, and William Woodall. Robot operating system 2: Design, architecture, and uses in the wild. *Science Robotics*, 7(66):eabm6074, 2022.

[13] Alan McIntyre, Matt Kallada, Cesar G. Miguel, Carolina Feher de Silva, and Marcio Lobo Netto. neat-python. `https://github.com/CodeReclaimers/neat-python`, 2008.

[14] Nathan J. Mlot, Craig A. Tovey, and David L. Hu. Fire ants self-assemble into waterproof rafts to survive floods. *Proceedings of the National Academy of Sciences*, 108(19):7669–7673, 2011.

[15] Iñaki Navarro and Fernando Matia. An introduction to swarm robotics. *ISRN Robotics*, 2013, 01 2013.

[16] Richard E. Neapolitan and Xia Jiang. Chapter 5 - decision analysis fundamentals. In *Probabilistic Methods for Financial and Marketing Informatics*, pages 177–228. Morgan Kaufmann, Burlington, 2007.

[17] Giorgio Oliveri, Lucas C. van Laake, Cesare Carissimo, Clara Miette, and Johannes T. B. Overvelde. Continuous learning of emergent behavior in robotic matter. *Proceedings of the National Academy of Sciences*, 118(21):e2017015118, 2021.

[18] Timothy O'Connor. Emergent Properties. In *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Winter 2021 edition, 2021.

[19] E. Pagello, A. D'Angelo, C. Ferrari, R. Polesel, R. Rosati, and A. Speranzon. Emergent behaviors of a robot team performing cooperative tasks. *Advanced Robotics*, 17(1):3–19, 2003.

[20] Pranav Rajbhandari. Swarm Coppeliasim. `https://github.com/pranavraj575/swarm_coppeliasim`, July 2023.

[21] E. Rohmer, S. P. N. Singh, and M. Freese. V-rep: A versatile and scalable robot simulation framework. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1321–1326, 2013.

[22] Destin Sandlin. Gregarious caterpillar locomotion - raw video for analysis. `https://www.youtube.com/watch?v=YehR0wSUioY`, Jun 2013.

[23] Tristan Schuler, Cameron Kabacinski, Daniel M. Lofaro, Dhawal Bhanderi, Jennifer Nguyen, and Donald Sofge. Wall climbing emergent behavior in a swarm of real-world miniature autonomous blimps. In *Proceedings of the 15th International Conference on Agents and Artificial Intelligence, ICAART 2023, Volume 1, Lisbon, Portugal, February 22-24, 2023*, pages 225–232. SCITEPRESS, 2023.

[24] Kenneth Stanley, Jeff Clune, Joel Lehman, and Risto Miikkulainen. Designing neural networks through neuroevolution. *Nature Machine Intelligence*, 1, 01 2019.

[25] Kenneth O. Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2):99–127, 2002.

[26] Qiuyang Tao, Jaeseok Cha, Mengxue Hou, and Fumin Zhang. Parameter identification of blimp dynamics through swinging motion. In *2018 15th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, pages 1186–1191, 2018.

[27] Qiuyang Tao, Junkai Wang, Zheyuan Xu, Tony X. Lin, Ye Yuan, and Fumin Zhang. Swing-reducing flight control system for an underactuated indoor miniature autonomous blimp. *IEEE/ASME Transactions on Mechatronics*, 26(4):1895–1904, 2021.

[28] Vito Trianni, Roderich Groß, Thomas H. Labella, Erol Şahin, and Marco Dorigo. Evolving aggregation behaviors in a swarm of robots. In *Advances in Artificial Life*, pages 865–874, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[29] Vanoss, C and Panfilov, Alexander and Hogeweg, P and Siegert, F and Weijer, CJ. Spatial pattern formation during aggregation of the slime mould Dictyostelium discoideum. *JOURNAL OF THEORETICAL BIOLOGY*, 181(3):203–213, 1996.

Authorized licensed use limited to: University of New South Wales. Downloaded on August 01,2025 at 01:50:23 UTC from IEEE Xplore. Restrictions apply.