Zespol: A Lightweight Environment for Training Swarming Agents

Shay Snyder, Kevin Zhu, Ricardo Vega, Cameron Nowzari, Maryam Parsa
George Mason University
Fairfax, VA, USA
{ssnyde9,kzhu4,rvega7,cnowzari,mparsa}@gmu.edu

ABSTRACT

Agent-based modeling (ABM) and simulation have emerged as important tools for studying emergent behaviors, especially in the context of swarming algorithms for robotic systems. Despite significant research in this area, there is a lack of standardized simulation environments, which hinders the development and deployment of real-world robotic swarms. To address this issue, we present Zespol, a modular, Python-based simulation environment that enables the development and testing of multi-agent control algorithms. Zespol provides a flexible and extensible sandbox for initial research, with the potential for scaling to real-world applications. We provide a topological overview of the system and detailed descriptions of its plug-and-play elements. We demonstrate the fidelity of Zespol in simulated and real-word robotics by replicating existing works highlighting the simulation to real gap with the milling behavior. We plan to leverage Zespol's plug-and-play feature for neuromorphic computing in swarming scenarios, which involves using the modules in Zespol to simulate the behavior of neurons and their connections as synapses. This will enable optimizing and studying the emergent behavior of swarm systems in complex environments. Our goal is to gain a better understanding of the interplay between environmental factors and neural-like computations in swarming systems.

CCS CONCEPTS

• **Computing methodologies** → **Modeling and simulation**; **Multi-agent systems**.

KEYWORDS

multi-agent systems, swarm intelligence, modeling and simulation, applied neuromorphic computing

ACM Reference Format:
Shay Snyder, Kevin Zhu, Ricardo Vega, Cameron Nowzari, Maryam Parsa. 2023. Zespol: A Lightweight Environment for Training Swarming Agents. In *International Conference on Neuromorphic Systems (ICONS '23), August 1–3, 2023, Santa Fe, NM, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3589737.3606002

This work is licensed under a Creative Commons Attribution International 4.0 License.
ICONS '23, August 1–3, 2023, Santa Fe, NM, USA
© 2023 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0175-7/23/08.
https://doi.org/10.1145/3589737.3606002

1 INTRODUCTION

Despite having limited individual capabilities, species such as bees [16] and ants [14], combine their abilities collectively to achieve impressive feats, such as honeybee house hunting [14] and weaver ant nest construction [14]. Researchers have drawn inspiration from this natural phenomenon and developed weak agents capable of working together to solve unique challenges, including clustering [9, 34], classification [34], self-triggered coordination [26], and asynchronous cloud access [27]. However, there are several issues hindering the advancement of emergent behaviors in low-powered, disposable robotic swarms. The first issue is the lack of standardization in simulation, development, and evaluation, resulting in challenging verification and extension of results [8, 37]. The second issue involves domain adaptation problems when trying to recreate simulated emergent behaviors physically, which leads to significant performance reductions [15, 36, 37]. Finally, the increasing complexity of robotic control algorithms [19] present a significant obstacle to simulating multiple physical models required for swarming robotics. Although robotics research has often focused on developing individual capabilities [1], simulating swarming robotics requires the simulation of multiple physical models. However, current examples of distributed robotic simulation environments are often designed for larger-scale manufacturing and reinforcement learning problems [20], have complicated C and C++ interfaces [20], or are specialized for particular applications [32].

We developed Zespol, a Python-based simulation environment, to overcome the challenges of engineering swarms of robots with emergent behaviors. Zespol aims to facilitate research into the application of nature-inspired computing algorithms for swarming robotics. It provides a lightweight means of standardization for engineering swarms of robotics to emerge at collective behaviors before transitioning over to real-world experiments or higher fidelity simulations. Additionally, this framework establishes a direct connection to other simulation environments, minimizing domain adaptation penalties when switching from simulation to physical robotic systems. Zespol has native support for distributed parallelization and provides a modular, extensible, and well-documented Python-based interface compatible with neuromorphic computing platforms.

2 BACKGROUND & MOTIVATION

There have been a variety of previous works aimed at creating simulation environments for robotics and swarming applications. In [6], the Vectorized Multi-Agent Simulator (VMAS) is introduced as a framework designed for efficient multi-agent reinforcement learning applications. Their physics engine and the underlying control logic is written in PyTorch. VMAS has shown incredible

performance in allowing parallel environments to run on GPUs. While this focus on CUDA accelerated hardware is beneficial for GPU-compatible workloads, it struggles when the simulation environment needs to expand to multiple heterogeneous compute nodes. A core assumption of the VMAS platform is holonomicity. We believe emergent behaviors can be decoupled from the sensing and control problem. Zespol does not force this trade-off of simulation fidelity for speed, as this would compromise the connection between simulation and non-holonomic robotics.

Swarm-Sim as a 2D & 3D simulation core for swarm agents stands as a framework for the implementation and evaluation of swarming agents [11]. The major limitations of this framework are the lack of direct support for cluster-level parallelization, and the discrete grid coordinate system, which precludes emergent behaviors that depend on agents having a continuous state, such as milling with ground robots.

Introduced in [33], SwarmLab is a MATLAB [24] based drone swarm simulator. The simulator is designed around a Drone class that supports quadcopter and fixed-wing aircraft dynamics. This makes their framework unsuitable for use with any non-drone agents without extensive modifications to the code. Interfacing MATLAB with modern learning and neuromorphic processing frameworks presents additional difficulties. There are also notable performance issues associated with this framework that make running large-scale simulations a computationally expensive task.

MASON [22] is an agent-based simulation library designed from the ground-up to support custom Java-based [2] simulations. There are many similarities between MASON and Zespol such as the inherent separation between the environment and visualization systems along with the compartmentalized nature of individual simulations. Both MASON and Zespol allow agents to be given arbitrary dynamics. The major limitation of MASON is a consequence of using very advanced Java where the barrier to entry for new users can be high. This issue is only compounded when we consider the lack of a mature and low-barrier system for distributing these simulations among heterogeneous computing systems. Addressing these issues is one of the major goals of Zespol.

OpenAI Gym, introduced in 2016, was a pioneering platform in the field of single-agent reinforcement learning [7]. Out of the box, they support a wide variety of applications for classic control problems such as Box2D [10], and Atari [4]. Compared to Zespol, Gym has two major limitations in that it is primarily designed for reinforcement learning and the programmatic architecture around Gym is focused purely on single agent simulations which severely limits its applicability to multi-agent robotics [23, 37].

NetLogo [38] is another multi-agent simulation environment. It is primarily designed to be used in educational environments, as evidenced by its integrated IDE with a drag-and-drop GUI. This makes programming behaviors easy, but the NetLogo language is limited. It is possible to run Python and R code from within NetLogo, as well as invoke a NetLogo simulation from a Java environment, but the interfaces are clunky and limited; thus NetLogo is largely incompatible with current means of distributing computation and simulation environments among heterogeneous computing systems and modern learning frameworks. NetLogo's simulation speed is, at best, equal to that of MASON [22], but struggles with anything higher than two-dimensional environments.

In [37], they conducted an interactive simulation in the design loop where simulated experiments where tightly coupled with real-world experiments. This study was broken up into four distinct portions to minimize the simulation to reality gap: 1) Characterizing the salient capabilities of the real robot, 2) Building a minimally viable simulation environment that characterizes the measured capabilities of physical robots, 3) Developing and exploring potential emergent behaviors in simulation, and 4) Deploying real robots based on simulation-driven-hypothesis and evaluating the performance penalties associated with the domain shift. They used a binary controller [5] for the salient capabilities of real robots and created stable milling behaviors in NetLogo that also performed the same behavior on physical robots. Despite their ability to minimize the simulation to reality gap, we are interested in deploying low-power and scalable neuromorphic computing platforms and explore novel methods of arriving at emergent behaviors. Zespol is designed as a simulation framework compatible with existing neuromorphic frameworks [29–31] and hardware [25].

Some of the key differences between prior works and Zespol are summarized in Table 1. Zespol is the only simulator that is written in a user-friendly and well documented Python code, provides native capability for distributed (dist) simulation environments, and allows for arbitrary agent states and dynamics.

Table 1: Comparison of multi-agent simulators

Simulator	Lang	Dist	State	Dynamics
VMAS [6]	Python	No	Arbitrary	Holonomic
Swarm-Sim [11]	Python	No	Discrete	Holonomic
SwarmLab [33]	MATLAB	No	Continuous	Drone
MASON [22]	Java	No	Arbitrary	Arbitrary
NetLogo [38]	NetLogo	No	Arbitrary	Arbitrary
Gym [7]	Python	No	Arbitrary	Arbitrary
Zespol	**Python**	**Yes**	**Arbitrary**	**Arbitrary**

3 PROGRAMMATIC ARCHITECTURE

Zespol's underlying architecture is designed with modularity in mind where each fundamental building block has a plug and play interface. This design philosophy allows users to develop their own blocks such as sensor modules, controllers, and physical dynamics. All simulations are designed to minimize inter-object dependencies to reduce the chance of segmentation faults and minimize communication latency by only passing critical information between blocks. Each interface is thoroughly documented with the provided examples showing how users can extend the framework to support their needs. More formally, each building block is represented by two data structures that form an **object-state** relationship. We have provided three fundamental object-state pairs, Agent-AgentState, Swarm-SwarmState, and World-WorldState. A more detailed description of these pairs is given in the following.

Zespol objects are data structures responsible for containing all elements required for the object to function. For example, a robot object would contain the robot's current location, all of its sensor objects, controller objects, and control the interaction between these elements at every simulation time step.

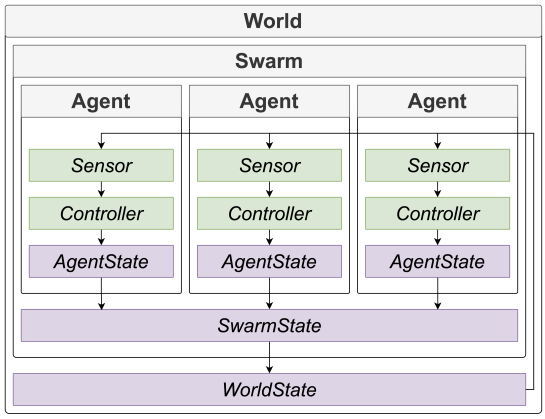

Figure 1: A flowchart presenting the critical programmatic flow within and between Zespol's main components.

The ***"Agent"*** object base class should be extended to support the specific requirements of a user's application. For example, the base class defines position and orientation vectors along three dimensions, a unique identifier, and the simulation time step fidelity. However, the *tick* method must be updated based on user requirements to control the interactions between sensors, controllers, and physical dynamics.

The ***"Swarm"*** class contains references to all Agents within the Swarm and controls the interactions between Agents at every simulation time step. This is where the distributed nature of Zespol is highlighted because the memory and process spaces for all Agents are separated, the processing of individual Agent updates at every time step can be distributed across heterogeneous compute clusters with tools such as Dask [13].

Bringing everything together, we have the ***"World"*** class that contains every object and actionable element within the simulation environment. Therefore, this object maintains references to all Swarms, visualization systems, and environmental objects such as world boundaries and obstacles. The last major responsibility of World objects is to manage the interactions between all Swarms and environmental objects to manage the programmatic flow at every simulation time step.

For every Agent, Swarm, and World object there are associated states that contains a holistic view of the object with the central idea being the establishment of a shareable data structure that only contains fundamental information. This avoids repeatedly passing redundant information between objects. For example, AgentStates contain an Agent's location and orientation but shouldn't contain a copy of the Agent's sensor or controller.

"AgentsStates" are defined by a snapshot of the given Agent's current location and heading, the change in these values from the previous step, along with their unique identifier. ***"SwarmStates"*** are represented by a collection of states from all member Agents along with a variety of metrics such as angular momentum, center of mass, scatter, and radial variance. Lastly, the ***"WorldState"*** encompasses the states of all Swarms along with all polygons that define the boundaries of the environment.

Besides the three predefined object-state pairs, there are three other notable objects within the system: ***Sensors***, ***Controllers***, and ***Visualizers***. Each ***"Sensor"*** is representative of a real-world sensor such as an RGB camera or LIDAR scanner that uses information within the WorldState to recreate a synthetic version of the perspective an Agent would see from their location in the world. ***"Controllers"*** accept input from Sensors and modify the location, orientation, and heading of an agent based on their physical dynamics. These dynamics are arbitrary so they can be modified to fit a user's specific application. ***"Visualizers"*** in Zespol are separable, optional components of the simulator. They take a WorldState at every time step and generate visual output. We include a visualization system based around Matplotlib [17] to provide users with an example to follow when extending these utilities to support their specific application.

The overall algorithmic flow starts at (1) initializing all Swarms and Agents within the World. (2) The WorldState object is constructed by querying all Swarms and Agents for their SwarmStates and AgentStates, respectively. (3) For every Agent within every Swarm, an artificial sensory perception is calculated in the Agent's Sensor based on its location relative to all other elements in the environment. (4) This perception is then passed to the associated Controller where the AgentState is modified. (5) Once every Agent in every Swarm has calculated their new states, any visualizations and logs can be created. (6) Lastly, the newly accumulated WorldState is used to progress through the next simulation time-step. Figure 1 provides a visual representation for the algorithmic flow between the fundamental Zespol elements.

4 INITIAL RESULTS & DISCUSSION

Our initial use case for Zespol was recreating the circular milling behavior from [9, 37] where agents move in a uniform circle. Using knowledge gained from [37] and Zespol's modular framework, we set up a simulation environment consisting of 9 Flockbots [21] with each being equipped with a front-facing infrared proximity sensor and a differential drive system. An image of a real-world Flockbot can be seen in Figure 2.

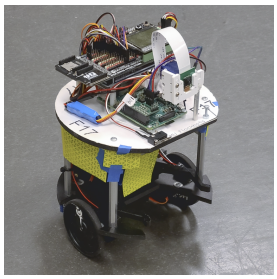

Figure 2: Image of a real-world Flockbot [21]

To fully implement this environment in Zespol, we extended the Agent class with the FlockbotAgent class, a BinarySensor class, and a DifferentialDriveController class. As shown in Figure 3, the entire process starts at the WorldState going into the BinarySenor where a synthetic binary output is calculated based on the Agent's current location and orientation with respect to the rest of the world. Next,

the binary sense is transferred to the DifferentialDriveController where the agent turns left if it senses something or turns right if it senses nothing.

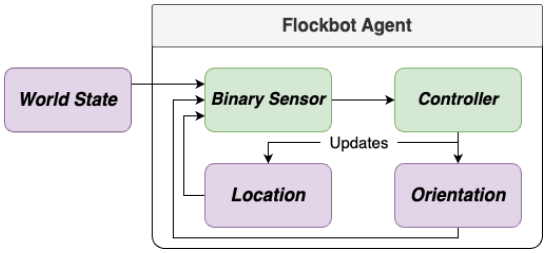

Figure 3: A flowchart presenting a detailed view of the inter-process communication within our example Zespol application using a Flockbot robot with a binary sensor

.

There are numerous parameters for the Flockbot milling behavior that we selected based on the results of [37] where the environment ticks at 30 ticks per second, the Swarm contains 9 Agents, and Sensors have a view distance of 3 meters and the same asymmetric field-of-view found in [37] with a left bound of 11.5 degrees left of center and a right bound of 4 degrees left of center.

Zespol successfully models the complex coordination between multiple agents that results in a stable milling behavior. A visualization of the resulting formation is shown in Figure 4. This highlights the ability of Zespol to recreate emergent behaviors from other simulated environments and experimental results that have been validated on real-world robotic systems.

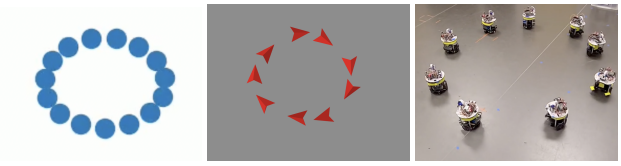

Figure 4: A swarm of Flockbot robots demonstrating a stable milling behavior in Zespol analogous to results found in NetLogo and real-world experiments

5 CONCLUSION AND FUTURE WORK

In conclusion, the field of agent-based modeling and simulation for studying emergent behaviors has witnessed substantial growth in parallel with the demand for robotic systems that can perform collective tasks. However, the lack of standardization in simulation environments makes it challenging to compare and contrast newfound research ideas with existing methods. The Zespol environment is introduced to serve as a lightweight and modular, Python-based simulation environment for developing multi-agent control algorithms. It offers ample opportunities for adoption and expansion by the broader research community. Moreover, the fidelity of Zespol is evaluated against previously published results in simulated and real-world robotics, demonstrating its ability to replicate existing swarming algorithms with the comparison between Zespol, NetLogo, and real robots conducting the milling behavior with Flockbots. With Zespol, users can develop and standardize swarming algorithms before transitioning over to real-world experiments or higher fidelity simulations. Zespol also provides native support for distributed parallelization across compute clusters and is compatible with neuromorphic computing platforms. As a result, it is a promising solution to issues slowing the advancement of emergent behaviors in robotic swarms of low-powered and individually incapable robotic systems.

Although Zespol is already demonstrating promising results, there is still room for improvement to make it a solid foundation for research on the application of neuromorphic computing in swarming robotics. Our plans include developing formal interfaces for common neuromorphic computing frameworks such as Lava [18] and Nengo [3]. We will also incorporate formal support for evolutionary algorithms [12] and Bayesian optimization learning schemes [28]. To simplify the distributed nature of Zespol, we will create a user-friendly interface that minimizes the hassle of dealing with Dask [13] and multiprocessing [35]. Additionally, we will incorporate a vectorized simulation module to run simulations on multiple GPUs across heterogeneous systems. Finally, we will leverage spiking controllers to discover novel swarming behaviors.

ACKNOWLEDGEMENT

This work was supported in part by the Department of the Navy, Office of Naval Research (ONR), under federal grant N00014-22-1-2207.

REFERENCES

[1] Amit Adate and BK Tripathy. 2022. A survey on deep learning methodologies of recent applications. *Deep learning in data analytics: Recent techniques, practices and applications* (2022), 145–170.
[2] Ken Arnold, James Gosling, and David Holmes. 2005. *The Java programming language.* Addison Wesley Professional.
[3] Trevor Bekolay, James Bergstra, Eric Hunsberger, Travis DeWolf, Terrence Stewart, Daniel Rasmussen, Xuan Choo, Aaron Voelker, and Chris Eliasmith. 2014. Nengo: a Python tool for building large-scale functional brain models. *Frontiers in Neuroinformatics* 7, 48 (2014), 1–13. https://doi.org/10.3389/fninf.2013.00048
[4] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. 2013. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research* 47 (2013), 253–279.
[5] F. Berlinger, M. Gauci, and R. Nagpal. 2021. Implicit coordination for 3D underwater collective behaviors in a fish-inspired robot swarm. *Science Robotics* 6, 50 (2021).
[6] Matteo Bettini, Ryan Kortvelesy, Jan Blumenkamp, and Amanda Prorok. 2022. VMAS: A Vectorized Multi-Agent Simulator for Collective Robot Learning. *arXiv preprint arXiv:2207.03530* (2022).
[7] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. OpenAI Gym. *CoRR* abs/1606.01540 (2016). arXiv:1606.01540 http://arxiv.org/abs/1606.01540
[8] Daniel S. Brown, Ryan Turner, Oliver Hennigh, and Steven Loscalzo. 2016. Discovery and Exploration of Novel Swarm Behaviors Given Limited Robot Capabilities. In *International Symposium on Distributed Autonomous Robotic Systems.*
[9] Daniel S Brown, Ryan Turner, Oliver Hennigh, and Steven Loscalzo. 2018. Discovery and exploration of novel swarm behaviors given limited robot capabilities. In *Distributed Autonomous Robotic Systems.* Springer, 447–460.
[10] Erin Catto. 2023. Box2d: A 2d physics engine for games. *URL: http://www. box2d. org* (2023).
[11] Ahmad Reza Cheraghi, Karol Actun, Sahdia Shahzad, and Kalman Graffi. 2020. Swarm-sim: A 2d & 3d simulation core for swarm agents. In *2020 3rd International Conference on Intelligent Robotic and Control Engineering (IRCE).* IEEE, 1–10.
[12] Mark A Coletti, Eric O Scott, and Jeffrey K Bassett. 2020. Library for evolutionary algorithms in Python (LEAP). In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion.* 1571–1579.
[13] Dask Development Team. 2016. *Dask: Library for dynamic task scheduling.* https://dask.org

[14] Simon Garnier, Jacques Gautrais, and Guy Theraulaz. 2007. The biological principles of swarm intelligence. *Swarm intelligence* 1, 1 (2007), 3–31.

[15] H. Hamann, T. Aust, and A. Reina. 2020. Guerrilla performance analysis for robot swarms: Degrees of collaboration and chains of interference events. In *International Conference on Swarm Intelligence*. Springer, 134–147.

[16] Rasika M Harshey. 1994. Bees aren't the only ones: swarming in Gram-negative bacteria. *Molecular microbiology* 13, 3 (1994), 389–394.

[17] J. D. Hunter. 2007. Matplotlib: A 2D graphics environment. *Computing in Science & Engineering* 9, 3 (2007), 90–95. https://doi.org/10.1109/MCSE.2007.55

[18] Intel Neuromorphic Computing Lab (NCL). 2022. *Lava: A software framework for neuromorphic computing*. Intel, Santa Clara, California, USA. https://lava-nc.org/

[19] Artúr István Károly, Péter Galambos, József Kuti, and Imre J. Rudas. 2021. Deep Learning in Robotics: Survey on Model Structures and Training Strategies. *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 51, 1 (2021), 266–279. https://doi.org/10.1109/TSMC.2020.3018325

[20] Jacky Liang, Viktor Makoviychuk, Ankur Handa, Nuttapong Chentanez, Miles Macklin, and Dieter Fox. 2018. GPU-Accelerated Robotic Simulation for Distributed Reinforcement Learning. *CoRR* abs/1810.05762 (2018). arXiv:1810.05762 http://arxiv.org/abs/1810.05762

[21] S. Luke, K. Andrea, M. Bowen, D. Fleming, K. Sullivan, B. Hrolenok, C. Vo, A. Bovill, R. Steck, B. Davidson, et al. 2014. The Flockbots. *Open specification. Available at http://cs. gmu. edu/ eclab/projects/robots/flockbots* (2014).

[22] Sean Luke, Claudio Cioffi-Revilla, Liviu Panait, Keith Sullivan, and Gabriel Balan. 2005. Mason: A multiagent simulation environment. *Simulation* 81, 7 (2005), 517–527.

[23] MHA Majid, MR Arshad, and RM Mokhtar. 2022. Swarm robotics behaviors and tasks: a technical review. *Control Engineering in Robotics and Industrial Automation: Malaysian Society for Automatic Control Engineers (MACE) Technical Series 2018* (2022), 99–167.

[24] MATLAB. 2010. *version 7.10.0 (R2010a)*. The MathWorks Inc., Natick, Massachusetts.

[25] J. Parker Mitchell, Catherine D. Schuman, Robert M. Patton, and Thomas E. Potok. 2020. Caspian: A Neuromorphic Development Platform. In *Proceedings of the 2020 Annual Neuro-Inspired Computational Elements Workshop* (Heidelberg, Germany) *(NICE '20)*. Association for Computing Machinery, New York, NY, USA, Article 8, 6 pages. https://doi.org/10.1145/3381755.3381764

[26] C. Nowzari and J. Cortés. 2012. Self-triggered coordination of robotic networks for optimal deployment. *Automatica* 48, 6 (2012), 1077–1087. https://doi.org/10.1016/j.automatica.2012.03.009

[27] Cameron Nowzari and George J. Pappas. 2016. Multi-Agent Coordination with Asynchronous Cloud Access. In *2016 American Control Conference (ACC)* (2016-07). 4649–4654. https://doi.org/10.1109/ACC.2016.7526085

[28] Maryam Parsa, Shruti R Kulkarni, Mark Coletti, Jeffrey Bassett, J Parker Mitchell, and Catherine D Schuman. 2021. Multi-Objective Hyperparameter Optimization for Spiking Neural Network Neuroevolution. In *2021 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 1225–1232.

[29] James S. Plank, Catherine D. Schuman, Grant Bruer, Mark E. Dean, and Garrett S. Rose. 2018. The TENNLab Exploratory Neuromorphic Computing Framework. *IEEE Letters of the Computer Society* 1, 2 (2018), 17–20. https://doi.org/10.1109/LOCS.2018.2885976

[30] Catherine D. Schuman, Shruti R. Kulkarni, Maryam Parsa, J. Parker Mitchell, Prasanna Date, and Bill Kay. 2022. Opportunities for neuromorphic computing algorithms and applications. *Nature Computational Science* 2, 1 (Jan 2022), 10–19. https://doi.org/10.1038/s43588-021-00184-y

[31] Catherine D. Schuman, J. Parker Mitchell, Robert M. Patton, Thomas E. Potok, and James S. Plank. 2020. Evolutionary Optimization for Neuromorphic Systems. In *Proceedings of the 2020 Annual Neuro-Inspired Computational Elements Workshop* (Heidelberg, Germany) *(NICE '20)*. Association for Computing Machinery, New York, NY, USA, Article 2, 9 pages. https://doi.org/10.1145/3381755.3381758

[32] Enrica Soria, Fabrizio Schiano, and Dario Floreano. 2020. SwarmLab: A MATLAB drone swarm simulator. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 8005–8011.

[33] Enrica Soria, Fabrizio Schiano, and Dario Floreano. 2020. SwarmLab: a Matlab Drone Swarm Simulator. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 8005–8011. https://doi.org/10.1109/IROS45743.2020.9340854

[34] Alexandros Tzanetos and Georgios Dounias. 2020. A comprehensive survey on the applications of swarm intelligence and bio-inspired evolutionary strategies. *Machine Learning Paradigms* (2020), 337–378.

[35] Guido Van Rossum and Fred L. Drake. 2009. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA.

[36] G. Vásárhelyi, C. Virágh, G. Somorjai, T. Nepusz, A.E. Eiben, and T. Vicsek. 2018. Optimized flocking of autonomous drones in confined environments. *Science Robotics* 3, 20 (2018), eaat3536.

[37] Ricardo Vega, Kevin Zhu, Sean Luke, Maryam Parsa, and Cameron Nowzari. 2023. Simulate Less, Expect More: Bringing Robot Swarms to Life via Low-Fidelity Simulations. https://doi.org/10.48550/ARXIV.2301.09018

[38] U. Wilensky. 1999. *NetLogo*. Northwestern University, Evanston, IL, USA. http://ccl.northwestern.edu/netlogo/