SALSA: Swarm Algorithm Simulator

Joel Beedle
Department of Computer Science
University of York
York, United Kingdom
jjb577@york.ac.uk

Calum Imrie
Department of Computer Science
University of York
York, United Kingdom
calum.imrie@york.ac.uk

Radu Calinescu
Department of Computer Science
University of York
York, United Kingdom
radu.calinescu@york.ac.uk

***Abstract*—Swarm algorithms are being increasingly investigated as potential solutions for addressing distributed, complex problems across various domains. However, developing and testing these algorithms remains challenging due to the lack of robust and flexible testbeds. Moreover, efficiently tuning the parameters of swarm algorithms to suit specific situations is a significant challenge. This artifact paper presents SALSA, a comprehensive and extensible framework designed to streamline the development and evaluation of swarm algorithms - designed with ease of use in mind. Our testbed enables users to define custom swarm algorithms, drone types, targets to detect, and agent interaction processes. It also allows for dynamic parameter updates, providing instant feedback to optimize algorithm performance. Additionally, the testbed supports both user-defined and automated data collection, ensuring that users can gather relevant data efficiently. Overall, SALSA enhances research effectiveness by reducing the time and effort required to set up and test swarm algorithms.**

***Index Terms*—Swarms, Aerial Swarms, Multi-Agent Systems, Self-Organizing Systems, Simulation, Testbed**

I. Introduction

In recent years, drone swarms have received significant attention due to their promising results and unique approaches when tackling a range of complex challenges in extensive fields. Examples of such applications are in wildfire monitoring [1]–[3], environmental monitoring [4], [5], search and rescue operations [6]–[8], among many more [9]–[11]. This evolution of swarms from simple object-avoiding, flocking [12] systems to drones working collectively to locate targets is a process that current simulation software lacks coverage of. Swarms have shown how effective a collection of agents can be when working towards a common goal. Consecutively, despite the increased attention that these systems have received, the rapid iteration, parameter tuning and development of these algorithms remain difficult tasks, partially due to the absence of an adequate, versatile, and quick to use testing environments. There are very few options for frameworks that exist when wanting to have a platform to quickly iterate new ideas specifically in swarm related systems, and existing systems were often not created to work this way. SALSA distinguishes itself by offering extreme configurability, modularity, and ease of use. It allows users to actively modify parameters and collect data, facilitating rapid prototyping and collecting large amounts of data from multiple tests - an essential feature that is not fully supported by existing platforms like SwarmLab. These factors place SALSA in a unique position within the current landscape, effectively addressing the limitations of existing simulators and filling an important gap in current simulation technologies. SALSA is a product of research that utilized aerial swarms for investigating trees for forest health maintenance and this artifact has the following key components:

1) A lightweight and flexible simulator, allowing users to visually see their simulations, and also allowing real time changes to parameters with the GUI; SALSA can also be adapted with respect to the individual agent's properties and the problem itself.
2) Multiple inbuilt swarm algorithms a user can already experiment with, or use for comparison against their own developed algorithms; these are flocking, pheromone avoidance, dynamic space partitioning, and random walking.
3) A testbed which can facilitate complex experiments automatically, and store a variety of data that can be used for further analysis, which includes testing over a variety of algorithms for performance comparative purposes.

In this paper we show the capabilities and features of SALSA, as well as demonstrate the key components with the forest health maintenance scenario.

II. Related Work

A plethora of robotic simulators exists, and most can be used for the modeling of single and multi agent systems.

Robotic simulators: Gazebo [13]–[15], CoppeliaSim [16], and WeBots [17] are commonly used 3D robot simulators that are predominantly designed for ground robots and complex robotic interactions in 3D space. There are specialized simulators for investigating aerial drones, such as AirSim [18] and is specifically designed for drones and autonomous vehicles. It provides real-world testing through its hardware-in-the-loop capabilities. ARGroHBotS [19] and USARSim [20] are examples of specialized simulators for specific aerial drone research, including the interactions between aerial and ground robots, and providing a realistic simulation environment for urban search and rescue operations.

Robot swarm simulators: ARGoS [21] is a robot swarm simulator that stands out due to its scalable architecture, allowing the user to configure the environment and physics engines using dynamically loadable plugins; aiming to provide a good balance between extensibility and scalability. Though

the current variety of robotic models is limited. Breve [22] is another 3D simulator that focuses on multi-agent systems, with the aim to effectively research artificial life.

Lightweight simulators: Listed so far are simulators which targets to capture realism to a significant degree. Therefore these are intrinsically complex meaning that they have difficulty in scaling as well as needing significant effort to set up and execute experiments. Lightweight simulators are useful for performing extensive experiments. Stage [23], for example, efficiently supports large-scale 2D multi-agent systems, and is often used alongside the Player project, which provides an interface to control each simulated robot using real robot control algorithms. Enki [24] is geared towards 2D simulations of ground based, wheeled robot swarms. Though these simulators are not easily transferable to aerial systems currently.

Aerial swarm simulators: Less plentiful are aerial swarm simulators, but those that do exist also fall on the spectrum of realism, flexibility, and practicality. Examples of 3D simulators include robotsim [25], [26] and the simulator provided by D'Urso et al. [27] which supports basic robot modeling, environmental physics, and sensor simulations. These simulators align closer with real-world applications rather than rapid prototyping. SwarmLab [28] is a drone swarm simulator that can be considered on the other side of this spectrum. SwarmLab is a testbed written in Matlab that offers tools for simulating and analyzing collective behaviors. It focuses on simplicity and ease of use, and like SALSA is targeted towards allowing the prototyping of algorithms.

SALSA's contribution: SwarmLab is, to the best of our knowledge, the nearest in functionalities to SALSA, however the use cases are slightly different. SwarmLab offers movement in 3D space, whereas SALSA is just 2D. In comparison, SwarmLab, which mirrors the objectives of our proposal, contains crucial differences in operation and execution. Whilst it enables the incorporation of user defined novel swarm algorithms and drone specifications, it lacks the capabilities to conduct tailored experiments through the Test Queue system towards specific scenarios. A defining feature of SALSA is its flexibility in configuring swarms through dynamic, real time parameter adjustments, based on observation and analytics, a functionality that is not provided by SwarmLab. Moreover, SwarmLab's capabilities are predominately tied to its graphical user interface, limiting the configurability and modularity the system provides. In contrast, SALSA separates its testbed and simulation library, offering users the possibility to customize or extend the testbed according to their specific needs, within an established framework. Notably, drone swarms have demonstrated prowess in their ability to locate targets, which is a feature that is not present in SwarmLab. The immediate functionalities of SwarmLab do not entirely encompass what a user may require from a swarm simulation testbed, and therefore there is a gap in the simulation system environment which SALSA aims to fill.

Generally, the majority of the simulators detailed have the goal of providing realistic simulations, and typically towards single agent systems. They require at least some level of configuration for multi agent systems, making them less suited to rapid algorithm development and analysis. There are limited simulators for aerial swarms, though most were designed for a use case that is non-generic. These are difficult to adapt into the methods that are required for aerial simulations, or for a different problem. Additionally they lack the specific optimizations and features of SALSA such as extensible behaviors and dynamics of the agents, flexible problem setup, and the comprehensive testbed for extensive experimentation. This highlights a pressing need for a versatile, high-performance simulator tailored to the dynamic requirements of multi-agent systems. SALSA, along with information on user support and how to contribute, is available on GitHub: https://github.com/joelbeedle/salsa. A permanent link to SALSA is available on Zenodo: https://zenodo.org/doi/10.5281/zenodo.13151118

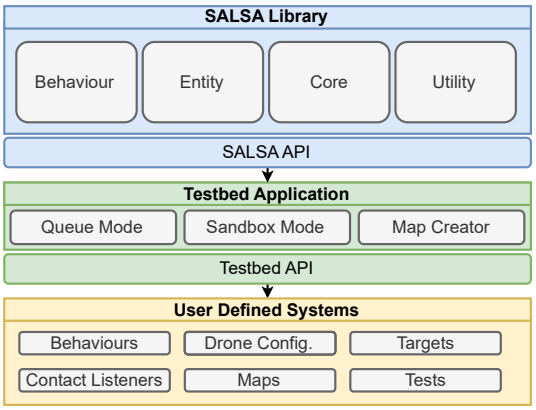

Fig. 1. Architecture Model and Implementation Diagram

III. Architecture and Implementation

SALSA is structured into two primary components: the *Library* and the *Testbed*. Fig. 1 contains an overview of the architecture. This structure is tailored for modularity and ease of use, enabling users to interface, modify, and extend functionalities as per their individual requirements. The core of SALSA is the Swarm Simulator Library, designed for modularity through Object Oriented Programming design, alongside the use of common Design Patterns. SALSA is built upon the *Box2D* physics engine (https://box2d.org/), which comes with a testbed already that SALSA extended. The Testbed acts as a practical application layer/wrapper that utilizes the Library. It adds GUI capabilities and rendering functionalities, providing an interactive and visually responsive platform for simulations. In essence, the Testbed is a direct wrapper around the *Simulation* class, enabling interaction and modification of the Simulations.

The Library defines a *Simulation* as a collective structure containing drones, an active swarm behavior, targets for the drones to seek (if any), a map, and a data collection system. The key components of the Library can therefore be identified as: the *Simulation* class, the *Drone* and *Target* classes, and the *Behavior* class. Registries are used extensively, such as in the creation of behaviors, contact listeners, targets, and maps, in

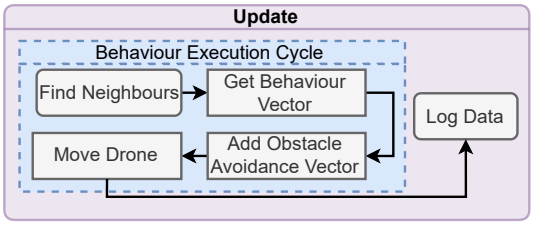

Fig. 2. Overview of the update simulation step. After logging data, a new world step must be called externally.

order to ensure modularity: modules interface with each other through these registries.

A. Simulation

The most important feature of the Library is the Simulation class, which collates all other elements, such as Drones, Targets, and Behaviors, to create a simulation platform. Notably, the Simulation does not invoke a 'Step' in the world, allowing freedom for users to define when the world takes a step compared to the simulation. Fig. 2 shows an overview of this simulation cycle. The central control point of the Simulation is the *update* function, which first calls each Drone's *update* function, which uses the *execute* Behavior method to control movement (this is completely defined by the individual algorithm), manages targets (updating an internal list of those found), and controls data logging. Data can be captured every step, or at step intervals decided by the user. The simulation uses asynchronous logging, flushing to the output file every 3 seconds. The simulation employs the use of observers. Drone observers (by default, the data collection logger) are notified of drone positions and velocities, followed by notifying the simulation observer to log general simulation data, such as the total number of targets found at that time-step. A Simulation can either be instantiated with all parameters set via a Test Configuration, or with minimal parameters: initial drone and target counts, and a map. This design ensures efficiency, as only the current Simulation being rendered is instantiated, with future simulations' parameters stored in the test queue. During the simulation, any parameters can be changed, including the number of drones (they will need to re-spawn), swarm algorithm parameters, drone parameters, the map, and more. The Contact Listener monitors collisions, executing user-defined Collision Handlers, when collisions between defined types (Drones, user Targets) are detected.

The Testbed application seen in Fig. 1 provides a pre-built interface to interact with and control a Simulation, operating with a GUI or headless. Headless mode allows simulations to execute at maximum speed without frame rendering delays. The Simulator modes in the Testbed provide a comprehensive way to interact with the simulation. In sandbox mode, users can test and debug algorithms, visualizing issues instantly. Even in visual modes, simulations can run faster than real-time, either accurately (calling multiple entire steps per frame) or inaccurately (increasing the size of a time step), facilitating rapid development. Accurate faster-than-real-time running is

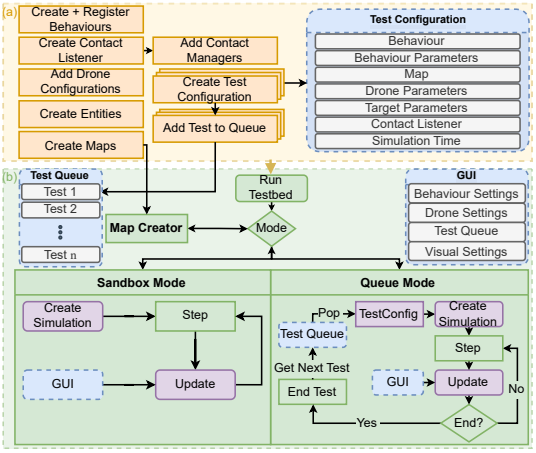

Fig. 3. **SALSA Testbed** Modes can be switched instantly. GUI and Test Queue are controlled by the Testbed, and Simulation Library code is in purple.

only provided by the headless mode. After developing algorithms, users can switch to queue mode, to set pre-defined tests or create permutations for algorithm parameter values. The test queue outputs data for each simulation, with data output controlled via the GUI. Plots are generated at the end of each simulation, for informative visualization. The GUI allows users to select a swarm behavior, dynamically change behavior parameters, control visual settings (e.g. drone sensor ranges, target visibility), and edit drone configuration settings (e.g. sensor ranges, maximum speeds, and forces). In queue mode, the simulation test queue is displayed, enabling users to add, remove, reorder, and edit tests.

B. Drones and Targets

Drones and Targets are extensions of the base *Entity* class. An Entity defines standard features for dynamic entities, including observer notifications, IDs, radius, and color. The Drone class builds on this by adding movement, behaviors, and drone sensors, while Target is a virtual class intended for further extension by the user (e.g. we used *Trees* as a subclass of Targets).

Drones are modeled as circles based on quad-copters, featuring omni-directional movement controlled by a swarm algorithm. Their movement is restricted by maximum speed, forces, mass, and radius. Drones have sensors for obstacle, target, and neighboring drone detection. Users can modify the drones' attributes such as size, ranges for sensors, maximum speed, and force, either through the GUI or with the Drone Configuration structure, set beforehand in the test queue. During simulation, each drone's pointer is set to the current Behavior instance, and the *execute* function of the behavior is called to determine the drone's movement for each step.

Targets are user-defined, as they vary greatly depending on the scenario. All targets must have a *world*, *position*, and *id*, with custom parameters defined as needed. Once complete, targets are registered in the Target Factory, with the user specifying any parameters they have added to their extension, and what these values should be set to.

C. Swarm Behaviors

The Behavior class provides the interface for users to create custom algorithms for Drones. It contains useful functions such as obstacle avoidance, steering, and drone avoidance. Users can add new algorithms by extending the Behavior class, implementing the *execute* function to update the Drone's movement, and registering it in the Behavior Registry. More details of this can be done are are provided in the SALSA Git repository. Inheriting from existing algorithms is also possible to reduce code reuse. The Testbed includes examples such as Flocking, Random Walking, Dynamic Space Partitioning (DSP), and Pheromone Avoidance, showcasing the simulator's flexibility - custom algorithms only need to override *execute*, allowing for diverse implementations. For instance, in Pheromone Avoidance, a drone can detect pheromones by accessing a shared list of points - this could have been done dynamically using AABB detection [29], illustrating the design freedom available. In DSP, the simulation space is divided into regions, assigning each drone a point to head towards, and then walk randomly around it upon arrival. Flocking implements Reynold's Boids [12], where each drone computes separation, cohesion, and alignment vectors, in order to determine acceleration. Random Walking involves drones moving randomly without communication. These example algorithms were developed in the Testbed using the same methods available to users.

D. Testbed Implementation

The Testbed, seen in Fig 3, is a versatile example of the extension and use of the SALSA Library, featuring a GUI for dynamic parameter changes, and multiple extensible modes to operate in. Sandbox and Queue modes interface directly with the SALSA API, while the Map Creation mode enables users to design maps for use in simulations. User code should be written in `user.cpp`, and custom behaviors and targets should be defined in their respective folders. Within user code, tests are created and added to the Test Queue, contact listeners are created and assigned contact managers to handle collision between types, and configurations are created. When the Testbed is ran, it executes user code followed by the Testbed code. Although it requires a full rebuild for any changes, this process is quick. In Sandbox mode, the Test Queue GUI supports creating and modifying tests and the queue. It allows for parameter exploration: users can create permutations of different parameters, and export them to a JSON file. Plots are automatically generated after each test run, with options for which to plot available in the GUI. The Testbed can also run headless, executing pre-defined tests from the Test Queue, and can be set to generate plots or to not. Plots are automatically generated using a Python script, which is an example of how a user might use the result logs.

IV. Testbed Workflow

The Testbed workflow involves two distinct stages: configuration, and development. The configuration stage involves users creating systems and foundations for their simulations, such as creating custom maps and targets. Then, users define collision listeners to handle interactions between drones and their custom targets, configurations for drones, and, if desired, various instances of their targets. After this stage, the Testbed has the structure in place to run user simulations, and record data for analysis and evaluation purposes. Workflow then enters the second stage, development. Users begin a cycle, developing custom swarm algorithm, evaluating them, and then improving them. At first, this likely takes place in the Sandbox mode of the Testbed. Sandbox mode allows users to evaluate their algorithm visually across various maps, swarm sizes, drone configurations, and algorithm parameters, via dynamic updates using the GUI. Once an algorithm is perceived to be working as expected, users can switch the Testbed to Queue mode to extensively test their algorithm's performance in various scenarios. The Test Queue can be used to run multiple simulations with various parameters, collecting performance data. SALSA can also generate graphs for important performance data within the context of forest health maintenance, many of which are typical for evaluating swarm algorithms. Currently SALSA can plot drone speed and distance, heatmap and trace of drone positions, and targets detected over time for each simulation. Fig. 4 contains output from simulating an aerial swarm employing the flocking algorithm. Furthermore, the Test Queue can be used to permute algorithm parameters to ascertain the best settings for optimal performance. Once satisfied with the algorithm development, users can move it to a realistic simulator (examples listed in Sec. II), where using SALSA initially will have saved time implementing a realistic controller for an algorithm that has not been perfected or fully understood.

V. Evaluation

To demonstrate SALSA we evaluated various swarm algorithms tasked with continuous monitoring for forest health maintenance. We evaluated swarm techniques based on how many trees were successfully detected. For this we considered quad-copter drones, and the Drone configurations abstracted the different sensor packages required. We extended the Target class into Trees, and registered a contact manager that told the program whenever a drone sensor covered a tree, to consider

TABLE I
Testbed Performance

No. Drones	**No. Targets**	**Real Time Factor** (RTF)[a]
10	100	0.0017 ± 0.0003
	1000	0.0046 ± 0.0005
	$10,000$	0.0291 ± 0.0024
100	100	0.0250 ± 0.0025
	1000	0.0413 ± 0.0023
	$10,000$	0.0919 ± 0.0468
500	100	0.2139 ± 0.0238
	1000	0.2872 ± 0.0208
	$10,000$	0.5280 ± 0.0666
1000	100	0.5668 ± 0.0217
	1000	0.5134 ± 0.0336
	$10,000$	0.9392 ± 0.0864

[a] RTF $= \frac{\text{simulated time}}{\text{real time}}$

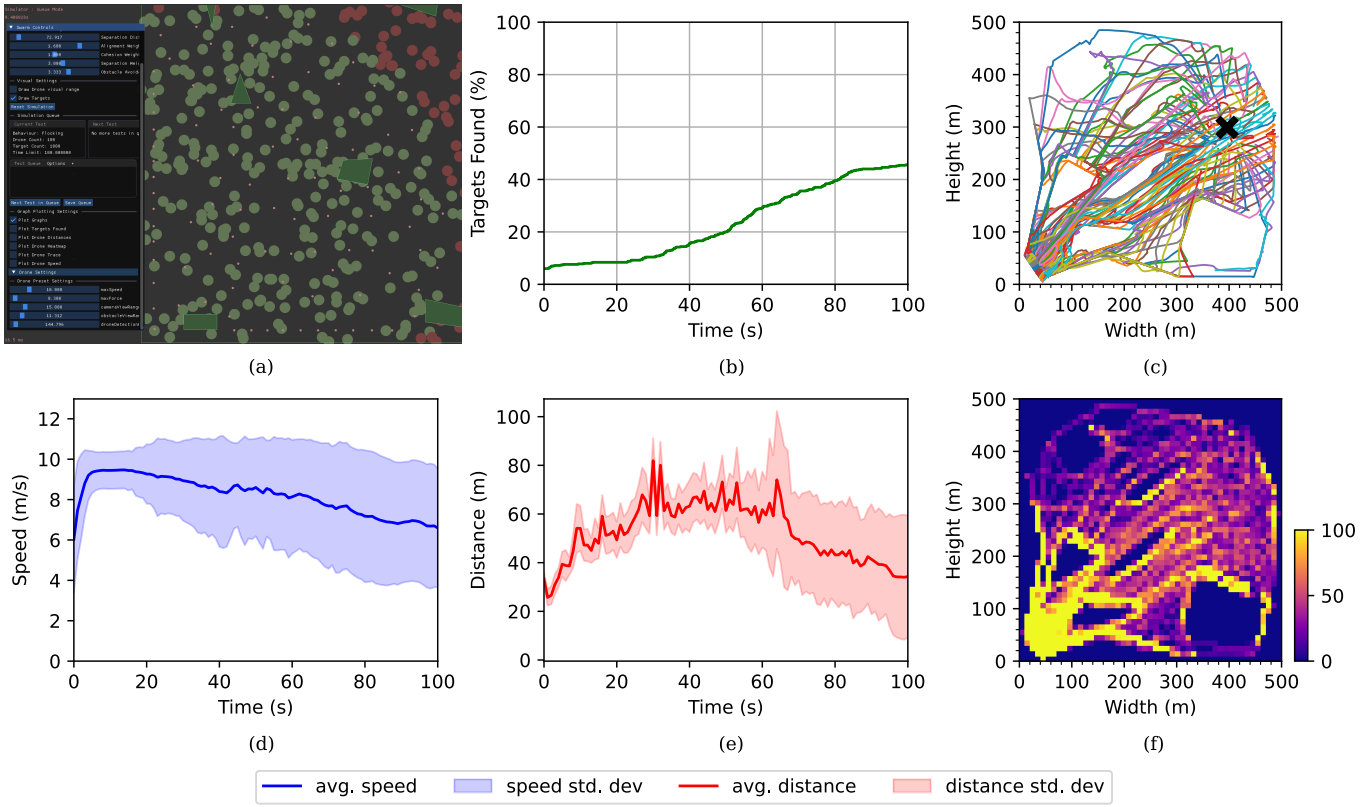

Fig. 4. **'Flocking' behavior test results**. The drones spawn at the top right (the black cross), then travel towards a location in the bottom left. (a) Example view from the simulator: targets are green if found, red otherwise. Drones are pink circles, and obstacles are green shaded areas. (b) Percentage of targets found over time (c) Paths of the drones over time through the entire test (c) Drone speeds over time (e) Drone Distances to nearest neighbors over time (f) A heat-map showing the concentrations of drone positions in each section for the entire simulation, normalized from 0 to 100. A video is available on the SALSA repo demonstrating the simulator.

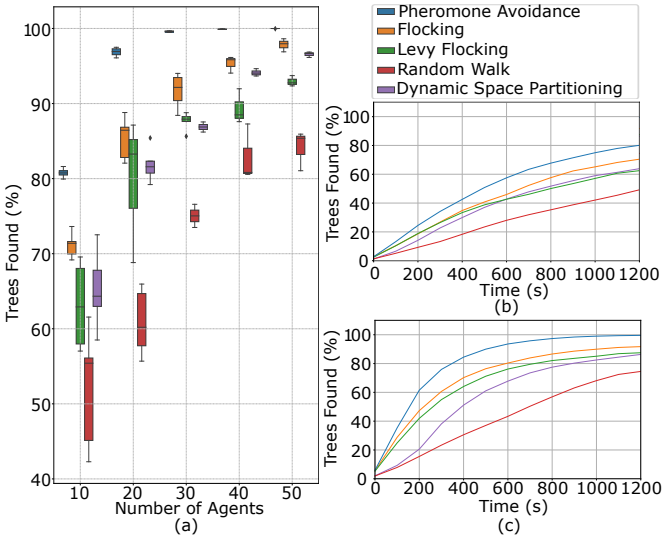

Fig. 5. **Experimental results, created using the test queue** (a) Box plot showing the number of trees found per behavior for varying swarm sizes. Each algorithm and swarm size were tested 5 times, automated using the test queue. (b) Trees found for a swarm size of 10 (c) Trees found for a swarm size of 30

the tree found. We tested the simulation with 5 different algorithms, each 5 times for 10, 20, 30, 40, and 50 drones, for a total of 125 different tests, and $50,000$ trees in a $2km^2$ area. Each simulation ran for 20 minutes in real time.

The running of and data collection for large amounts of tests were performed using the Test Queue, with the results in Fig. 5. The results indicate that the pheromone avoidance algorithm performs consistently the best and can achieve nigh complete coverage with 30 drones. The pheromone avoidance algorithm also has the smallest error bars for all swarm sizes. This most likely due to the pheromones encouraging drones to avoid visiting areas that has already been visited by another drone, thus less randomness with the individual drone's behavior. The ability to replicate these tests is shown in the README file.

The performance of SALSA and the Testbed was also analyzed by measuring the time taken to simulate between 10 and 1000 drones, and for each number of drones between 100 and $10,000$ targets, modeled as our trees. The simulation space was an empty $2km^2$ map, with the drones using our 'Flocking' behavior and spawning in the center, and ran for 100 seconds. The Testbed on average performed all simulations faster than real time, the results are shown in Table I. SwarmLab reported results of a RTF of 0.02s-11s for 2-1024 drones [28], indicating SALSA has greater scaling capability.

A 2021 MacBook Pro Laptop with an M1 Apple processor and 32GB memory was used for these experiments.

VI. Conclusion

The proposed SALSA Library and Testbed provides functionality for rapid development of swarm algorithms. Dynamic and responsive interactions allow users to visualize their algorithms and how parameters change the behavior in a unique way, and allows for a deeper understanding in the ways these parameters affect a swarm algorithm. By giving users freedom to create the algorithms however they want, with the only limitation being to implement a single function means that SALSA is easy to learn, and effective to use - we believe there will be little significant overhead spent learning how to use the system. Furthermore, custom testbeds could be created using the Library for purposes other than aerial drones, such as terrestrial robots. Future work includes extending the base simulator, perhaps implementing drone controllers, different drone types, involving real drone components, adding weather, or perhaps a damage or drop out system, in order to enable users to keep developing within SALSA for longer before moving towards a more realistic simulation. The integration of SALSA into development pipelines - primarily in the algorithm design stage - would streamline workflows, and therefore future work should also be spent trying to specialize SALSA to fit and work with realism-focused simulation software. A sophisticated comparison of SALSA to other swarm simulators in terms of efficiency and scalability would also be important in the future.

Acknowledgment

The work described in this paper received support from the UKRI Trustworthy Autonomous Systems pump-priming project 'ASPEN', the EPSRC project EP/V026747/1 'UKRI Trustworthy Autonomous Systems Node in Resilience', and the Centre for Assuring Autonomy.

References

[1] J. Yang, J. Qian, and H. Gao, "Forest wildfire monitoring and communication uav system based on particle swarm optimization," in *Journal of Physics: Conference Series*, vol. 1982, no. 1. IOP Publishing, 2021, p. 012068.

[2] S. Manoj and C. Valliyammai, "Drone network for early warning of forest fire and dynamic fire quenching plan generation," *EURASIP Journal on Wireless Communications and Networking*, vol. 2023, 11 2023.

[3] G. Tzoumas, L. Pitonakova, L. Salinas, C. Scales, T. Richardson, and S. Hauert, "Wildfire detection in large-scale environments using force-based control for swarms of uavs," *Swarm Intelligence*, vol. 17, pp. 1–27, 11 2022.

[4] G. De Masi and E. Ferrante, "Quality-dependent adaptation in a swarm of drones for environmental monitoring," in *Proceedings of the 2020 Advances in Science and Engineering Technology International Conferences (ASET)*, 02 2020, pp. 1–6.

[5] J. Zhang, J. Hu, J. Lian, Z. Fan, X. Ouyang, and W. Ye, "Seeing the forest from drones: Testing the potential of lightweight drones as a tool for long-term forest monitoring," *Biological Conservation*, vol. 198, 03 2016.

[6] G. Kantor, S. Singh, R. Peterson, D. Rus, A. Das, V. Kumar, G. Pereira, and J. Spletzer, "Distributed search and rescue with robot and sensor teams," in *Springer Tracts in Advanced Robotics*, vol. 24, 07 2003, pp. 529–538.

[7] D. Stormont, "Autonomous rescue robot swarms for first responders," in *CIHSPS 2005. Proceedings of the 2005 IEEE International Conference on Computational Intelligence for Homeland Security and Personal Safety, 2005.*, 2005, pp. 151–157.

[8] M. Bernard, K. Kondak, I. Maza, and A. Ollero, "Autonomous transportation and deployment with aerial robots for search and rescue missions," *Journal of Field Robotics*, vol. 28, no. 6, pp. 914–931, 2011.

[9] A. Slowik and H. Kwasnicka, "Nature inspired methods and their industry applications—swarm intelligence algorithms," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 3, pp. 1004–1015, 2018.

[10] R. Poli, "Analysis of the publications on the applications of particle swarm optimisation," *Journal of Artificial Evolution and Applications*, vol. 2008, p. 10, 01 2008.

[11] M. Abdelkader, S. Güler, H. Jaleel, and J. S. Shamma, "Aerial swarms: Recent applications and challenges," *Current Robotics Reports*, vol. 2, no. 3, pp. 309–320, 2021. [Online]. Available: https://doi.org/10.1007/s43154-021-00063-4

[12] C. W. Reynolds, "Flocks, herds and schools: A distributed behavioral model," *SIGGRAPH Comput. Graph.*, vol. 21, no. 4, p. 25–34, aug 1987. [Online]. Available: https://doi.org/10.1145/37402.37406

[13] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, vol. 3, 2004, pp. 2149–2154 vol.3.

[14] J. Meyer, A. Sendobry, S. Kohlbrecher, U. Klingauf, and O. von Stryk, "Comprehensive simulation of quadrotor uavs using ros and gazebo," in *Simulation, Modeling, and Programming for Autonomous Robots*, I. Noda, N. Ando, D. Brugali, and J. J. Kuffner, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 400–411.

[15] L. Pitonakova, M. Giuliani, A. Pipe, and A. Winfield, *Feature and Performance Comparison of the V-REP , Gazebo and ARGoS Robot Simulators*. Springer International Publishing, 2018.

[16] 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*, 2013, pp. 1321–1326.

[17] O. Michel, "Cyberbotics ltd. webots™: Professional mobile robot simulation," *International Journal of Advanced Robotic Systems*, vol. 1, no. 1, p. 5, 2004.

[18] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "Airsim: High-fidelity visual and physical simulation for autonomous vehicles," 2017.

[19] N. I. Ospina, E. Mojica-Nava, L. G. Jaimes, and J. M. Calderón, "Argrohbots: An affordable and replicable ground homogeneous robot swarm testbed," *IFAC-PapersOnLine*, vol. 54, no. 13, pp. 256–261, 2021.

[20] S. Carpin, M. Lewis, J. Wang, S. Balakirsky, and C. Scrapper, "Usarsim: a robot simulator for research and education," in *Proceedings 2007 IEEE International Conference on Robotics and Automation*, 2007, pp. 1400–1405.

[21] C. Pinciroli, V. Trianni, R. O'Grady, G. Pini, A. Brutschy, M. Brambilla, N. Mathews, E. Ferrante, G. Di Caro, F. Ducatelle, M. Birattari, L. M. Gambardella, and M. Dorigo, "Argos: a modular, parallel, multi-engine simulator for multi-robot systems," *Swarm Intelligence*, vol. 6, pp. 271–295, 12 2012.

[22] J. Klein and L. Spector, *3D Multi-Agent Simulations in the breve Simulation Environment*. London: Springer London, 2009, pp. 1–79. [Online]. Available: https://doi.org/10.1007/978-1-84882-285-6_4

[23] R. Vaughan, "Massively multi-robot simulation in stage," *Swarm Intelligence*, vol. 2, no. 2, pp. 189–208, 2008.

[24] G. Adorni, "Simulation of robot swarms for learning communication-aware coordination," 2023.

[25] G. Vásárhelyi, C. Virágh, G. Somorjai, T. Nepusz, A. E. Eiben, and T. Vicsek, "Optimized flocking of autonomous drones in confined environments," *Science Robotics*, vol. 3, no. 20, 2018.

[26] C. Virágh, G. Vásárhelyi, N. Tarcai, T. Szörényi, G. Somorjai, T. Nepusz, and T. Vicsek, "Flocking algorithm for autonomous flying robots," *Bioinspiration & Biomimetics*, vol. 9, 10 2013.

[27] F. D'Urso, C. Santoro, and F. Santoro, "An integrated framework for the realistic simulation of multi-uav applications," *Computers & Electrical Engineering*, vol. 74, pp. 196–209, 03 2019.

[28] E. Soria, F. Schiano, and D. Floreano, "Swarmlab: a matlab drone swarm simulator," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020, pp. 8005–8011.

[29] P. Cai, C. Indhumathi, Y. Cai, J. Zheng, Y. Gong, T. Lim, and P. Wong, *Collision Detection Using Axis Aligned Bounding Boxes*. Springer Berlin Heidelberg, 11 2014, pp. 1–14.