A Software Framework for Self-Organized Flocking System Motion Coordination Research

Fredy Martínez, Holman Montiel, Edwar Jacinto
Universidad Distrital
Francisco José de Caldas
Bogotá D.C., Colombia

***Abstract*—We describe and analyze the basic algorithms for the self-organization of a swarm of robots in coordinated motion as a flock of agents as a strategy for the solution of multi-agent tasks. This analysis allows us to postulate a simulation framework for such systems based on the behavioral rules that characterize the dynamics of these systems. The problem is approached from the perspective of autonomous navigation in an unknown but restricted and locally observable environment. The simulation framework allows defining individually the characteristics of the basic behaviors identified as fundamental to show a flocking behavior, as well as the specific characteristics of the navigation environment. It also allows the incorporation of different path planning approaches to enable the system to navigate the environment for different strategies, both geometric and reactive. The basic behaviors modeled include safe wandering, following, aggregation, dispersion, and homing, which interact to generate flocking behavior, i.e., the swarm aggregates, reach a stable formation and move in an organized fashion toward the target point. The framework concept follows the principle of constrained target tracking, which allows the problem to be solved similarly as a small robot with limited computation would solve it. It is shown that the algorithm and the framework that implements it are robust to the defined constraints and manage to generate the flocking behavior while accomplishing the navigation task. These results provide key guidelines for the implementation of these algorithms on real platforms.**

Keywords*—*Flocking; formation control; motion planning; multi-robot system; obstacle avoidance; swarm

I. Introduction

A field of great interest in robotics poses the solution of problems not with a high-performance robot but with a group of robots, simpler in structure, but which can interact to behave as a single system [1], [2]. These systems are known as multi-agent, and seek to exploit the ability of biological systems such as flocks of birds [3], [4], schools of fish [5], [6], ants [7], [8] or aggregations of bacteria to solve complex tasks together [9], [10]. However, the control of these types of systems raises problems of high complexity given the characteristics of these dynamics. These are systems with self-organization capacity, which results as an emergent consequence of the system from basic behaviors of each of the agents [11]. The design of these basic behavioral rules that should generate the emergent behavior of the system is difficult. In addition, the robots, as a single system, must perform some task.

The flocking behavior presents interesting characteristics that make it of high interest for the design of artificial systems, particularly in problems of localization, search and rescue. This type of behavior has been observed in birds, is similar to schooling fish and swarming insects, and is characterized by a joint movement of the group without central coordination [12], [13]. The first basic rules of this dynamic were established in 1987 as alignment, cohesion, and separation [14]. Subsequently, in 2003, a mathematical model of the 1987 Reynolds rules was proposed taking advantage of the geometrical strategies and the theory of Artificial Potential Fields (APF) [15], [16]. In these works, it was demonstrated how potentials in the environment are not only able to guide the navigation of the robots to the target point, but also to form a homogeneous flock along with the navigation task [17]. It has been observed in different applications that much of the success of this dynamics lies in the local communication scheme between agents, which determines the ability to self-organize within the system [18], [19].

The dynamics of a flock of birds corresponds to that of randomized search algorithms [20]. These algorithms rely on a large number of agents distributed in the search space, which at the same time identify local information, maintain communication with their neighbors to jointly identify the most optimal solution option. In the case of the flock of agents, this solution corresponds to the region of the environment with the characteristics most similar to those defined in the search problem. These characteristics of the dynamics are what make it suitable for solving complex navigation problems, particularly in unknown and dynamic environments.

The complexity in the design of these systems lies in the fact that they can produce a system that is unable to solve the task either because the number of robots is too high or too low, if the local signal being tracked is too strong for the population size, or if the navigation environment is too complex [21]. These types of problems can drive the system to local minima, or impede the performance of the task. The motion planning of each robot within the system is linked to the structure of the system and the local information it can detect from the environment [22], [23]. Part of the information from the environment is transmitted to each robot from the movement of the system, which makes it robust to continuous changes of dynamics in the environment, but also dependent on the characteristics of the environment and the system for the success of the task. In this sense, the proper design of both the system and the behavioral policies of the agents is fundamental, particularly if we are looking for a system made up of agents with modest computational resources [24].

This research focuses on the development of multi-agent navigation strategies in this type of environment, guided by the identification of regions of interest in the environment [25].

These strategies must be following the task to be solved, so parameters such as system size, the distance between agents, the scope of the communication system, and characteristics of the basic behaviors must be different in each case [26]. Some assumptions are also made to simplify the model, but without moving it functionally away from the real prototypes. Among the initial assumptions, it is proposed to work with a single robotic platform (uniform system) with perfect displacement capabilities, constant values of the parameters throughout the development of the task, and discrete behavior of the agents in the sense that each action is triggered by a certain stimulus [27]. The objective is to achieve preliminary results that allow evaluating the success of the strategy and its possible implementation in real robots. Possible tasks in real applications include tracking pollution sources (static, dynamic, and multi-focal), intruder detection, or wildlife tracking.

One of the first developments in software tools to replicate the behavior of robot swarms was the one developed by Craig Reynolds to demonstrate the performance of his basic rules for flocking behavior in this type of multi-agent systems [14]. A later evolution by the same author was the OpenSteer project, a framework for implementing autonomous agent motions and behaviors [28]. These tools were originally intended for video game development, yet robotics has benefited from their use in demonstrating behavioral algorithms on them. The advantage (and disadvantage) in the use of this type of tool is the need for prior knowledge about the dynamics of multi-agent systems, a simulator for video games does not require such information, while a simulator dedicated to robotics requires it while allowing great versatility in the implementation of algorithms.

Perhaps the best-known robotic simulation tool is Player Project [29]. It is an open-source software specifically developed for robotics that allows a large number of controls and sensors to be incorporated into a client/server network interface. The results of these simulations can be visualized in the two-dimensional module Stage or the three-dimensional viewer Gazebo. Unfortunately, the last update of the project was made at the end of 2010. Still, Gazebo evolved independently, and today it is a dedicated high-performance simulation tool. Another project developed for robotics is Robot Virtual Worlds or RVW [30], which, although oriented more for robotics education purposes, under specific conditions can function as a simulation platform with its programming language. Other platforms worth mentioning include the CoppeliaSim project [31], and Webots from Cyberbotics Ltd [32], active projects, programmable in different languages, with a commercial focus.

We present the problem formulation in Section II. Here we give special attention to the basic behaviors that generate a flocking structure, and to the simplified representation assumed for the navigation problem. Section III presents the methodology followed for the construction of the framework, detailing not only the algorithm used but also the characteristics of its implementation. Section IV shows the results achieved in tests for controlled laboratory conditions, under which it is possible to perform experimental validation, and finally, the conclusions are presented in Section V.

II. Problem Statement

The flocking behaviors emerge in a multi-agent system as a consequence of a combination of basic or primitive behaviors executed independently by each of the agents. Among these primitive behaviors, five of them can be selected as the basis for the flocking structure [33]:

- Safe wandering: In the design of path planning solutions it is fundamental to guarantee the safe wandering of the robot along with the environment, this includes reducing collisions with other agents as well as with the obstacles and limits of the environment [34].
- Following: Robots must be able to establish their motion strategy from the motion of nearby robots. In the case of flock-based navigation strategies, each agent must identify neighboring agents, calculate its distance to them, and define its forward direction and speed to reduce interference in the system's motion [4].
- Aggregation: While agents must follow the movement of their neighbors, flocking behavior also requires that agents can dynamically assemble during navigation while maintaining a safe distance between them [35].
- Dispersion: Like aggregation, dispersion (self-localization of each robot in the system) turns out to be an important quality in autonomous coordination schemes, and is fundamental to the structure of the system throughout task development [36].
- Homing: This behavior allows each agent to move to the given target as part of the system task using sensed information in the environment [37].

That is, in a flock, each agent wants to stay close to the other agents (which it can detect), to do its best not to collide with them, and to move simultaneously towards the desired location. These behaviors make it possible to define a robust and well-organized flock. The objective of this research is to demonstrate that these basic behaviors allow generating a flocking behavior for an artificial system from fixed rules. This demonstration is done by simulation in a framework developed in Python for a system composed of TurtleBot 3 Burger robots from ROBOTIS. The goal is to scale the primitive behaviors to a large population of these agents.

Consequently, the problem is defined from the activation of n holonomic robots with known physical dimensions (not points) in an environment W unknown to the robots but partially observable from their sensors, and which is defined in a connected and compact two-dimensional plane $\left(W \subset \mathbb{R}^2\right)$. From this definition, it follows that all the constraints to which the TurtleBot 3 robotic platform can be subjected are integrable in positional constraints of the form:

$$f\left(q_1, q_2, q_3, \cdots, q_n; t\right) = 0 \tag{1}$$

where the variables q_i corresponds to the coordinates of the system.

Any typical environment W to be modeled by the framework contains in its interior a set of obstacles called O

that consists of regions inaccessible to the robot within W, where each of these regions is characterized by a closed and connected boundary. Therefore, the set O is also considered connected, finite, and piecewise analytic. An additional characteristic of each of the obstacles in O is that they are disjoint pairs of each other, so they do not share common points. The boundaries of W, denoted by ∂W, constrain the movement of the robots within the environment. In addition, the boundaries of the obstacles are also part of ∂W. The free space through which the robots can navigate is denoted by E and is defined as $W - O$.

Each of these robots (Fig. 1), according to its mechanical design, can be represented in the two-dimensional environment by a dish with a radius of 0.105 m (with center at the LiDAR sensor position) and an obstacle detection range (field of view of the LiDAR sensor) of 360 degrees with a range of 3.5 m. Other parameters derived from its design include a maximum forward speed of 0.22 m/s, and an acceptable range to define that it has reached a certain point in the environment of ± 0.5 m.

These robots have no explicit communication among themselves, only the ability to locate themselves and define their relative position concerning their neighbors (a basic type of local communication). The simulation framework assumes that the robots' sensing capability is perfect, that they are capable of perfect omnidirectional motion, and that they all follow the same navigation rules to define the path in W (Fig. 2).

III. Methods

The framework was developed in Python 3.7.12, with support for Numpy 1.19.5, Scipy 1.4.1, and Matplotlib 3.2.2. The tool allows simulating the movement of robots in the environment at a scaled relative speed, and the result is compressed into a video file. This is achieved with the Matplotlib animation library, included in the Matplotlib 1.1 version, which enables to obtain a visual demonstration of the behavior from the features programmed in the navigation algorithm. The base class of the animation tool is `matplotlib.animation.Animation`, on which the animation functionality is built. The interfaces of this tool are `TimedAnimation` and `FuncAnimation`, the latter is the one used in our framework. More details are provided below. From Numpy we use the linear algebra library `numpy.linalg.norm` to calculate the norms of the n-dimensional vectors. From Scipy we use the libraries `scipy.spatial.distance.pdist` to calculate the distances between points in the n-dimensional space, and `scipy.spatial.distance.squareform` that takes the previous results to form a square matrix of distances.

The first part of the code defines the global variables of the framework, which correspond mostly to user-configurable parameters according to the conditions of the problem to be simulated (Fig. 3 and Fig. 4). These variables include the population size of the system (how many robots will conform to the multi-agent system), the size of each of the robots (two-dimensional circular shape is assumed), the sensing range of the 360-degree distance sensors, the distance programmed in each robot to initiate the avoidance policy, the maximum speed, distance from the target to consider that the robot reached its destination, interval between simulation steps in

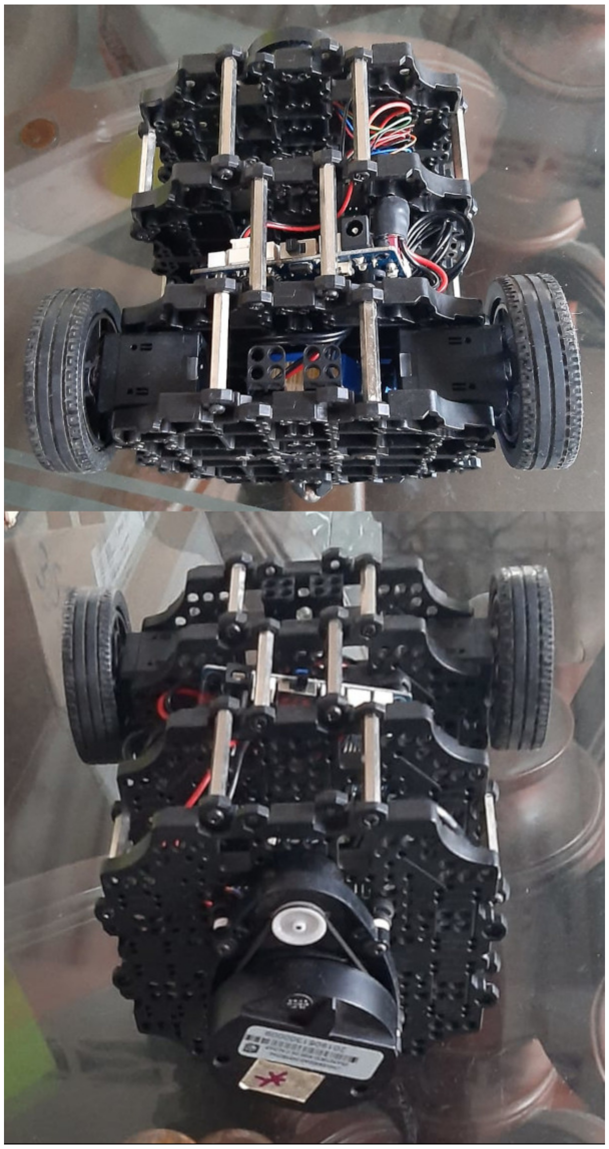
Fig. 1. TurtleBot 3 Burger Robot from ROBOTIS

simulation time, the initial position of the robot swarm, and dimensions of the simulation environment (the shape is always assumed to be two-dimensional rectangular, complex shapes in the environment can be achieved later with the definition of the obstacles O).

The second section of the code defines the navigation strategy to be followed by the robot swarm, i.e. how it will move in W to find the target area. In this part, we have facilitated the incorporation of several common algorithms, from the explicit definition of the route using navigation coordinates to the incorporation of geometric strategies such as Potential Field algorithm, Dijkstra, and A*, and even reactive strategies based on local sensing. Also in this section, the location of the target region is defined.

The third part of the initial configuration corresponds to the definition of the obstacles O. These are drawn on the environment using the `matplotlib.patches` library,

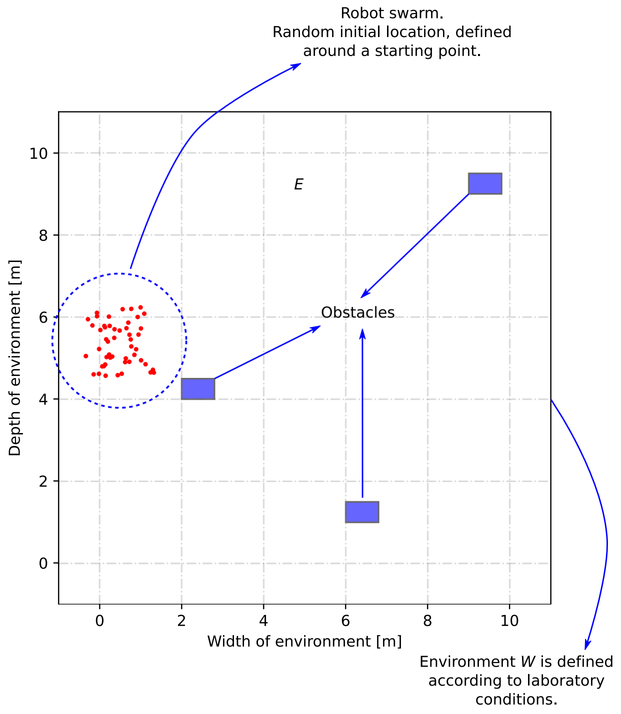
Robot swarm.
Random initial location, defined around a starting point.
E
Obstacles
Depth of environment [m]
Width of environment [m]
Environment W is defined according to laboratory conditions.
0
2
4
6
8
10

Fig. 2. Navigation Environment Defined in the Framework. The Definition of W, the Free Space E, the Obstacles, and the Swarm of Robots are Detailed

```
Initialize system
    - Number of agents
    - Agent size
    - Sensing range
    - Avoidance distance
    - Maximum speed
    - Final distance to target
    - Simulation window
    - Starting position
    - Environment dimensions
Initialize path planning
    - Navigation strategy
    - Location of target region
Obstacle configuration

BEGIN
    Build position and speed matrices
    For each agent i in population P do
        Background = init() → Obstacles
        Movement of agents = update()
            wandering()
            following()
            aggregation()
            dispersion()
            homing()
        Animate
    End For
End
```

Fig. 3. Pseudocode Detailing the Framework Structure

which corresponds to 2D objects defined by coordinates inside W. The coordinates are captured graphically with the help of a mouse pointer and converted to closed polygons that finally define E. It is also possible to define the coordinates of a rectangle, an option that is more versatile when looking for an analysis of sensitivity to the position of the obstacles. By default all obstacles in O are drawn in blue, to differentiate them from other elements of the simulation (agents are set in red and W boundaries in black).

With this information, we proceed to generate the initial position and velocity matrices of the system agents. In both cases, the values are generated randomly within the defined ranges. The initial state of the system is defined by scaling the environment and the robots for its visualization (the robots are represented as red-colored circles of proportional size to the environment). The figure is created with `matplotlib.pyplot.figure`, and all robots, obstacles, information labels, and a grid are added to facilitate position analysis. This configuration corresponds to the initial state of the system.

To perform the simulation the first thing to do is to initialize the robot speed arrangements. The speed of each robot can be kept constant at a percentage of the maximum value, or set to random in the same range. This is handled internally with a multiplier on the maximum speed between 0 and 1. This matrix is updated according to the motion policies applied to each robot. Animations in Matplotlib consist of three elements, an `init()` function that returns the background of each animation frame, an `update()` function that returns the figures that should appear in each background frame, and the code in charge of acquiring the animation object. In our case, the function `init()` loads the obstacles defined for the environment, and the function `update()` updates the position matrix from the previous matrix, the velocity matrix, and the result of the movement policies. The `animation.FuncAnimation` function takes as arguments the `init()` function, as well as the number of frames per iteration, the total animation interval, and some smoothing and updates instructions.

Each of the basic behaviors was implemented separately in functions that compute the velocity vector for each of the robots (each basis function for the entire system, from zero to n). The function for aggregation checks the readings from the distance sensors of each robot and adjusts the velocity vectors so that the robot moves towards its nearest neighbors. Distances to nearby robots are used to define the relative location of each robot in the environment, as well as its movement strategy [4]. A function is also implemented to establish attractors in the environment from the local readings, and the navigation strategy used. These attractors allow defining the velocity vector of each robot as if it were following a specific route. To avoid collisions, a function is defined that verifies the fulfillment of minimum distances between robots, obstacles, and environment limits, forcing the movement in a random direction when it detects a possible collision condition. These basic behaviors are combined to create more complex flocking behaviors. For example, the aggregation function and the attractor function combine to produce the following behavior and combined with the collision function they form a safe wandering behavior. Thus by combining homing, aggregation, and avoidance, the desired flocking behavior is achieved. In

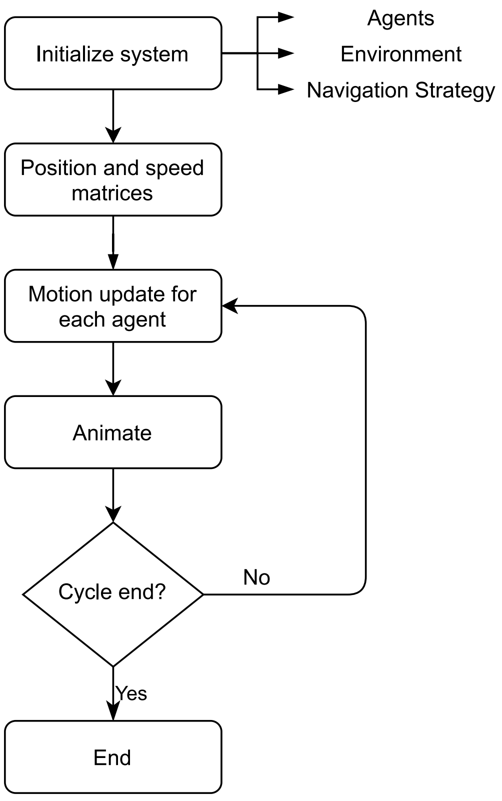

Fig. 4. Algorithm Flowchart

these combinations of basic behaviors, the effect of each is weighted to allow for a greater impact of one on the other, in most cases small homing versus high avoidance values and moderate aggregation produces the best flocking behavior.

IV. Results and Discussion

We have performed several tests with the framework for different conditions of the environment, the system, and the navigation strategy. The results shown below were performed for the TurtleBot 3 Burger robot, the platform on which we analyzed the flocking behavior. The characteristics of the environment (rectangular 10 m $\times$ 10 m, with three fixed obstacles) and the navigation strategy (reactive from intensity landmarks in the environment) were also kept constant. The varied parameters were population size and initial position of the system, with the intention not only to observe the self-organization in flocks but also to determine the performance of the system for a given navigation task concerning the population size. The system initialization parameters for these tests were as follows:

- Number of agents: between 5 and 100
- Agent size: circular with 0.105 m radius.
- Sensing range: 3.5 m
- Avoidance distance: 0.5 m
- Final distance to target: 0.5 m
- Simulation timestep: 0.01 s
- Starting position: Random in E
- Environment size: 10 m $\times$ 10 m

The navigation strategy forces the flock of robots to navigate the environment in a clockwise direction towards the lower right region of the environment. We evaluated the time it takes for the system to navigate to this region for different population sizes, from a flock of five agents to a system with 100 agents. Fig. 5 shows the capture of four such simulations, each with a different population size (20, 30, 40, and 50 agents). The starting point is randomly generated in E using a computer time-dependent variable seed. In the simulations, it is observed how the system self-organizes according to the basic behaviors, and after reaching equilibrium, it starts to move along the route without altering the formation, only in the event of encountering an obstacle, in which case the agent avoids it, and returns to the formation. A small animation of 20 s with these four cases can be seen in the following link:

https://youtu.be/R09mFqAb_-c

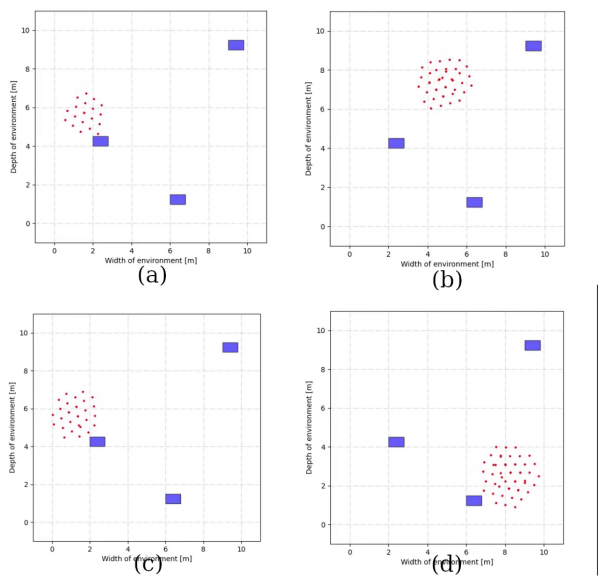

Fig. 5. Screenshots Corresponding to Four Simulations with Different Population Sizes Performing the Same Task. (a) 20 Agents, (b) 30 Agents, (c) 40 Agents, and (d) 50 Agents

The strength of the tool is observed when evaluating the performance of this type of system in the development of tasks, which is much more complex in implementations on real prototypes. To evaluate this, the above configuration was followed to assess the impact of population size on the total time required for task development. Since the navigation strategy relies on local readings, which may vary depending on the system agent detecting the landmarks, and according to the initial position of the system, statistical analysis with multiple simulations for multiple population sizes is needed to analyze the impact problem. Following the conditions of the previous simulations, the exercise was repeated for different population sizes: 5, 10, 20, 30, 40, 50, 60, 70, 80, 90, and 100 agents. For each population size, 100 simulations were performed (1100 simulations in total) and the times in seconds that each case required were recorded. In the simulations, a 100% success rate was obtained (in all cases the system reached the target region), but in some cases, the time required was an outlier (excessively long), which was also experimented with in real tests. The results are shown in Fig. 6, which shows a basic statistical analysis with median values by population size, quartiles, and excluded outliers.

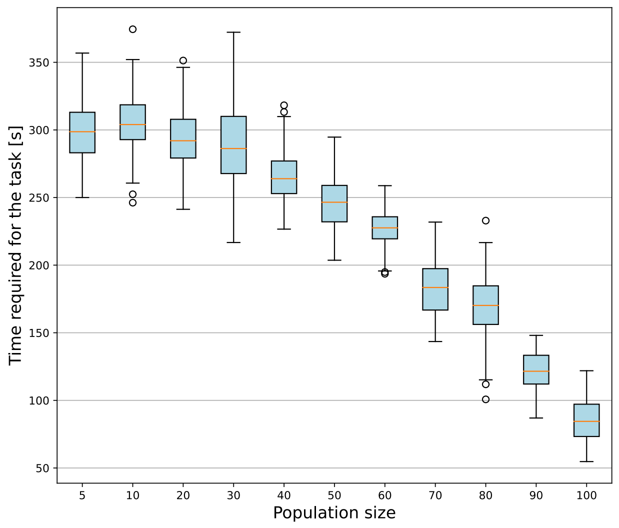

Fig. 6. Box and Whisker Plot of the Times for the Development of the Task for Different Population Sizes. Median, Quartiles, and Outliers are Detailed

The results show that beyond the fact that the system can perform the task, there is a relationship between the time required and the population size, for the particular working conditions (type of robot, size of the environment, and configuration of O) [25]. This relationship can be represented by a mathematical function, which could be useful for system sizing. This same exercise can be repeated with another set of obstacles, a different size environment, different speeds of the robots, or even some dynamic conditions making the obstacles change their position every so often. The framework allows to analysis and generalizes the behaviors of this kind of system, in a fast, economic and reliable way. These results can then be contrasted and scaled according to laboratory tests with some real robots.

V. Conclusion

In this paper, we propose a new simulation framework to replicate the flocking behavior of a swarm of robots, as a strategy for the efficient and safe evaluation of the performance of this type of system. The control scheme is designed based on the basic behaviors identified in the literature as fundamental for a flocking system: safe wandering, following, aggregation, dispersion, and homing. Each of these basic behaviors is replicated from each agent's sensor data, and weighted in conjunction with the navigation strategy to form the speed vector. We identify the weighted value of each basic behavior while allowing it to be adjusted according to the simulation needs. The framework also allows modeling different types of environments and robots, but the tests and calibrations were performed with the TurtleBot 3 Burger platform from Robotis. With this platform, the performance was tuned for a pair of agents, which was scaled to allow evaluating tens and hundreds of agents. In this sense, our framework allows us to adjust specific parameters such as robot size, sensing capacity, and maximum speed. The simulation performs an animation of the system using Python's Matplotlib library, which is exported to video. In this animation, the obstacles are placed in the background, while the agents are dynamically updated in the foreground. The code allows to implementation of a wide range of navigation strategies, both geometric from the global characteristics of the environment, as well as reactive navigation strategies based on local readings. The tool was verified for a simple navigation task, evaluating both the self-organizing capability of the system and the impact of system features on task performance. Future research on this tool includes the incorporation of other robotic platforms of the research group.

Acknowledgment

This work was supported by the Universidad Distrital Francisco José de Caldas, specifically by the Technological Faculty. The views expressed in this paper are not necessarily endorsed by Universidad Distrital. The authors thank all the students and researchers of the research group ARMOS for their support in the development of this work.

References

[1] M. Talamali, A. Saha, J. Marshall, and A. Reina, "When less is more: Robot swarms adapt better to changes with constrained communication," *Science Robotics*, vol. 6, no. 56, pp. ST1–ST16, 2021.

[2] M. Otte, "An emergent group mind across a swarm of robots: Collective cognition and distributed sensing via a shared wireless neural network," *The International Journal of Robotics Research*, vol. 37, no. 9, pp. 1017–1061, 2018.

[3] H. Ling, G. Mclvor, K. Vaart, R. Vaughan, A. Thornton, and N. Ouellette, "Costs and benefits of social relationships in the collective motion of bird flocks," *Nature ecology & evolution*, vol. 3, no. 6, pp. 943–948, 2019.

[4] F. Martínez, "Minimalistic control scheme for the development of search tasks with flocks of robots," *Journal of Physics: Conference Series*, vol. 1993, no. 1, p. 012025, 2021.

[5] L. Demidova and A. Gorchakov, "Research and study of the hybrid algorithms based on the collective behavior of fish schools and classical optimization methods," *Algorithms*, vol. 13, no. 4, p. 85, 2020.

[6] L. Li, M. Nagy, J. Graving, J. Bak-Coleman, G. Xie, and I. Couzin, "Vortex phase matching as a strategy for schooling in robots and in fish," *Nature Communications*, vol. 11, no. 1, pp. 1–9, 2020.

[7] S. Jones, T. J. Czaczkes, A. J. Gallager, F. B. Oberhauser, E. Gourlay, and J. P. Bacon, "Copy when uncertain: lower light levels increase trail pheromone depositing and reliance on pheromone trails in ants," *Animal Behaviour*, vol. 156, no. 1, pp. 87–95, 2019.

[8] Q. Luo, H. Wang, Y. Zheng, and J. He, "Research on path planning of mobile robot based on improved ant colony algorithm," *Neural Computing and Applications*, vol. 32, no. 6, pp. 1555–1566, 2020.

[9] T. Trunk, , H. S. Khalil, and J. C. Leo, "Bacterial autoaggregation," *AIMS Microbiology*, vol. 4, no. 1, pp. 140–164, 2018.

[10] F. Martínez, "Robust eletronic hardware system based on quorum sensing," phdthesis, Universidad Nacional de Colombia, 2017.

[11] S. Zhang, M. Liu, X. Lei, Y. Huang, and F. Zhang, "Multi-target trapping with swarm robots based on pattern formation," *Robotics and Autonomous Systems*, vol. 106, no. 1, pp. 1–13, 2018.

[12] P. Zhu, W. Dai, W. Yao, J. Ma, Z. Zeng, and H. Lu, "Multi-robot flocking control based on deep reinforcement learning," *IEEE Access*, vol. 8, pp. 150 397–150 406, 2020.

[13] J. Cheng and B. Wang, "Flocking control of mobile robots with obstacle avoidance based on simulated annealing algorithm," *Mathematical Problems in Engineering*, vol. 2020, no. 1, pp. 1–9, 2020.

[14] C. Reynolds, “Flocks, herds and schools: A distributed behavioral model,” *Computer Graphics*, vol. 21, no. 4, pp. 25–34, 1987.

[15] H. Tanner, A. Jadbabaie, and G. Pappas, “Stable flocking of mobile agents. i. fixed topology,” in *42nd IEEE International Conference on Decision and Control (IEEE Cat. No.03CH37475)*, 2003.

[16] ——, “Stable flocking of mobile agents. II. dynamic topology,” in *42nd IEEE International Conference on Decision and Control (IEEE Cat. No.03CH37475)*, 2003.

[17] F. Martínez, *Robótica Autónoma: Arquitecturas Multiagente que Imitan Bacterias*. Universidad Distrital Francisco José de Caldas, 2021, vol. 1.

[18] C. Veitch, D. Render, and A. Aravind, “Ergodic flocking,” in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019.

[19] E. Soria, F. Schiano, and D. Floreano, “The influence of limited visual sensing on the reynolds flocking algorithm,” in *2019 Third IEEE International Conference on Robotic Computing (IRC)*, 2019.

[20] L. Li and A. Talwalkar, “Random search and reproducibility for neural architecture search,” in *Proceedings of The 35th Uncertainty in Artificial Intelligence Conference*, ser. 1, vol. 115, 2020, pp. 367–377.

[21] H. Wei and X. Chen, “Flocking for multiple subgroups of multi-agents with different social distancing,” *IEEE Access*, vol. 8, pp. 164 705–164 716, 2020.

[22] O. Misir and L. Gökrem, “Flocking-based self-organized aggregation behavior method for swarm robotics,” *Iranian Journal of Science and Technology, Transactions of Electrical Engineering*, vol. 45, no. 4, pp. 1427–1444, 2021.

[23] L. Beaver and A. Malikopoulos, “An overview on optimal flocking,” *Annual Reviews in Control*, vol. 51, no. 1, pp. 88–99, 2021.

[24] J. Benitez, L. Parra, and H. Montiel, “Diseño de plataformas robóticas diferenciales conectadas en topología mesh para tecnología zigbee en entornos cooperativos,” *Tekhnê*, vol. 13, no. 2, pp. 13–18, 2016.

[25] A. Rendón, “Evaluation of autonomous navigation strategy based on reactive behavior for mobile robotic platforms,” *Tekhnê*, vol. 12, no. 2, pp. 75–82, 2015.

[26] Y. Kambayashi, H. Yajima, T. Shyoji, R. Oikawa, and M. Takimoto, “Formation control of swarm robots using mobile agents,” *Vietnam Journal of Computer Science*, vol. 06, no. 02, pp. 193–222, 2019.

[27] J. Castañeda and Y. Salguero, “Adjustment of visual identification algorithm for use in stand-alone robot navigation applications,” *Tekhnê*, vol. 14, no. 1, pp. 73–86, 2017.

[28] U. Erra, B. Frola, and V. Scarano, “BehaveRT: A GPU-based library for autonomous characters,” in *Motion in Games*, 2010, pp. 194–205.

[29] F. Martínez and D. Acero, “Autonomous navigation strategy for robot swarms using local communication,” *Tecnura*, vol. 18, no. 39, p. 12, 2013.

[30] K. Takaya, T. Asai, V. Kroumov, and F. Smarandache, “Simulation environment for mobile robots testing using ROS and gazebo,” in *2016 20th International Conference on System Theory, Control and Computing (ICSTCC)*, 2016, pp. 1–6.

[31] S. Rooban, S. Suraj, S. Vali, and N. Dhanush, “CoppeliaSim: Adaptable modular robot and its different locomotions simulation framework,” *Materials Today: Proceedings*, vol. 2021, no. 1, pp. 1–6, 2021.

[32] L. Turkler, T. Akkan, and L. Akkan, “Control of swarm robotics in webots with PSO,” in *2021 3rd International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA)*, 2021, pp. 1–6.

[33] M. Mataric, “Interaction and intelligent behavior,” phdthesis, Massachusetts Institute of Technology, 1994. [Online]. Available: https://dspace.mit.edu/handle/1721.1/7343

[34] B. Tong, Z. Xiaoqing, L. Xiaoli, and R. Xiaogang, “A mobile robot path planning method based on safe pathfinding guidance,” in *33rd Chinese Control and Decision Conference (CCDC 2021)*, 2021, pp. 1–6.

[35] O. Mısır, L. Gökrem, and S. M., “Fuzzy-based self organizing aggregation method for swarm robots,” *Biosystems*, vol. 196, no. 1, p. 104187, 2020.

[36] A. Kshemkalyani, A. Rahaman, and G. Sharma, “Dispersion of mobile robots on grids,” in *WALCOM: Algorithms and Computation*. Springer International Publishing, 2020, pp. 183–197.

[37] J. Xun, Z. Qidan, M. Junda, L. Peng, and Y. Tianhao, “Three landmark optimization strategies for mobile robot visual homing,” *Sensors*, vol. 18, no. 10, p. 3180, 2018.