A Demonstrator for Self-organizing Robot Teams

Gianluca Aguzzi(✉), Lorenzo Bacchini, Martina Baiardi, Roberto Casadei, Angela Cortecchia, Davide Domini, Nicolas Farabegoli, Danilo Pianini, and Mirko Viroli

Alma Mater Studiorum—University of Bologna, Cesena, Italy
{gianluca.aguzzi,m.baiardi,roby.casadei,angela.cortecchia,davide.domini,
nicolas.farabegoli,danilo.pianini,mirko.viroli}@unibo.it,
lorenzo.bacchini4@studio.unibo.it

Abstract. Aggregate computing is a paradigm with over a decade of investigation and multiple programming frameworks available, which proved to be particularly suitable for the simulation of applications in challenging domains such as smart cities and robot swarms. This paper introduces a toolchain for practical multi-robot demonstrations based on aggregate computing principles, and validates it with a live interactive demo in an open-public event in the context of the European Researchers' Night. More specifically, we show how we coordinated a team of mobile robots to form spatial patterns. We discuss the practical demonstration performed in an indoor environment, which exploits a camera system and ArUco markers for localization.

Keywords: Aggregate computing · Field-based coordination · Self-organization · Spatial formation · Multi-robot systems · Robot teams · Demonstrator

1 Introduction

Research Context. The rapid proliferation of Internet of Things (IoT) devices, pervasive [28] and collective [2] computing systems has created an unprecedented challenge in programming and coordinating highly distributed systems. Traditional programming approaches struggle to handle the complexity of these systems, where hundreds or thousands of devices need to work together seamlessly [20]. In particular, the coordination of robot teams is a challenging task, as issues related to localization, communication, and actuation add on top of the already complex problem of coordinating a group of devices. Several solutions have been proposed to address this challenge, including so-called *macro-programming* languages [12,22] that aim to address collective behaviour through

This work is supported by the Italian PRIN project "CommonWears" (2020 HCWWLP).

Published by springer Nature Switzerland AG 2025
C. Di Giusto and A. Ravara (Eds.): COORDINATION 2025, LNCS 15731, pp. 230–244, 2025.
https://doi.org/10.1007/978-3-031-95589-1_12

meso- or macro-scopic abstractions (e.g., spatiotemporal patterns, ensembles, collective interfaces and data structures). Examples include swarm-specific languages such as Buzz for practical swarm robot programming [26], high-level frameworks like Voltron for team coordination [21], and logic-based approaches like Meld for programming modular ensembles of robots [7]. In this paper, we focus on the aggregate computing paradigm [10] which emerges as a powerful paradigm to address this challenge, introducing *computational fields* [10,18] as a key abstraction to program the collective behavior of a group of (possibly situated) devices. This paradigm now has over a decade of investigation [30], with several programming frameworks like ScaFi [13], and FCPP [9] that provide the necessary abstractions to program aggregate systems and also led to several applications in smart cities and robotics [4].

Research Gap. Real-world deployment of aggregate systems faces several challenges. First, while theoretical foundations for general-purpose deployment methods exist (e.g., Pulverization [16]), practical toolchains tend to depend highly on the specific application context, making it difficult to devise generic platforms. This technical barrier, combined with the high costs associated with large-scale implementations, often complicates the translation of theoretical advances into widespread practical use. Consequently, real-world demonstrations of aggregate computing remain limited in scope. Hence, there is a need to improve the understanding of the paradigm's practical potential, possibly also positively impacting transition from theoretical validation (Technology Readiness Level [23], TRL 3) to system prototype demonstration in relevant environments (TRL 6).

Contribution. In this paper, we present a demonstrator for deploying aggregate computing solutions in real-world robot coordination scenarios. This is an entirely open-source tool [5], and can hence be used by anyone to run analogous or even more advanced demonstrators. This artifact is based on the MacroSwarm library [4] and the ScaFi programming framework [13]. Our contribution is threefold: (i) we propose a flexible and modular framework for deploying aggregate programs on robot teams, addressing key requirements such as localization, program distribution, and hardware abstraction; (ii) we implement a demonstrator that showcases the framework's capabilities using low-cost robots and vision-based localization; (iii) we validate our approach through a public demonstration at the European Researchers' Night, showing the effectiveness of aggregate computing in coordinating robot teams through spatial formations.

Paper Structure. The remainder of the paper is structured as follows. Section 2 provides an overview of aggregate computing and the ScaFi framework, as well as the MacroSwarm library. Section 3 describes the architecture of our framework and the requirements needed to deploy an effective aggregate system in a real team of robots. Section 4 presents the live demonstration held at the European Researchers' Night, detailing the setup, the execution, and the impact. Section 5 concludes the paper and outlines future work.

2 Background

2.1 Aggregate Computing in Practice

Aggregate computing is a programming paradigm that simplifies the development of distributed and decentralized systems by abstracting computations over networks of interacting devices as a single collective entity. At its core, aggregate computing introduces *computational fields* [10,18], mapping points in space and time with values that can be combined functionally. This paradigm enables the declarative specification of high-level behavior, providing robust and scalable coordination mechanisms.

Over the years, several frameworks have been developed to support aggregate computing, including ScaFi [13], and FCPP [9]. Each of these frameworks provides the core constructs of the *higher-order* computational field calculus [10], which serves as a foundation for "building blocks" that enable information propagation, retrieval, and evolution in time. These foundational building blocks constitute the backbone of libraries containing *self-stabilizing* [15] higher-level functions, including general-purpose [13] and specialized collective behaviours for domains such as swarm robotics [4] or hierarchical coordination [25].

Recent advances in aggregate computing research have explored its deployment on resource-constrained devices [8,16], enabling real-world applications where small embedded systems execute aggregate computations efficiently. For instance, lightweight implementations [9] allow the execution of aggregate programs on microcontrollers and low-power embedded boards, making the paradigm suitable for IoT networks and robotic systems. Among the available frameworks, ScaFi [13] stands out as a Scala-based aggregate computing environment designed to be usable both in simulation and real-world deployment. ScaFi provides a high-level Application Programming Interface (API) to define aggregate computations concisely while supporting extensibility for integrating with different execution environments.

2.2 Robot Programming with Aggregate Computing

In this paper, as a reference scenario for demonstration purposes, we select the field of swarm robotics, as it is one of the target applications highlighted in variety of recent works, including environmental monitoring [3], search and rescue, and, in particular, wildlife monitoring (e.g., tracking the movement of zebras in the wild [17]).

To respond to the emergent need for aggregate computing applications in swarm robotics, MacroSwarm [4] has been proposed as a minimal yet functional set of building blocks for swarm behavior, ranging from pattern formation to collective decision-making (see Fig. 1 to grasp the idea of the behavior which can be programmed with MacroSwarm). In the following, we briefly describe the assumptions on which the MacroSwarm library is based, to better understand the requirements upon which the demonstrator is built.

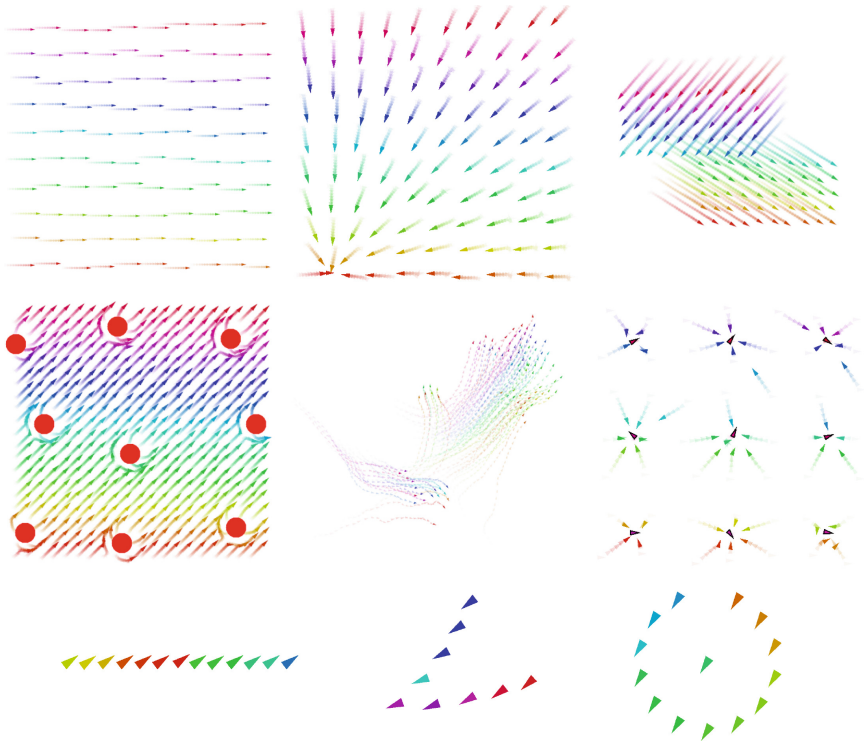

Fig. 1. Some of the patterns that can be programmed with MacroSwarm. From top left to bottom right: constant movement, sink to leader, branching (multiple direction), obstacle avoidance, flock (Reynold like), team formation, and various shapes

System Model. A MacroSwarm system consists of a collection of devices, each equipped with sensors and actuators. Each device communicates with its neighbors, defined by a *neighborhood relationship*, enabling inter-device interaction. Communication is *identity-based*, allowing devices to uniquely recognize one another. Moreover, every node must be capable of determining its own position in the environment as well as the positions of its neighbors. These minimal assumptions are essential for the effective execution of programs built upon the MacroSwarm framework.

Execution Model. MacroSwarm is built on ScaFi and leverages an asynchronous, atomic execution of computation *rounds*. Each round consists of three distinct phases:

1. **Sensing:** Every device gathers data from its local state and receives information from its neighbors, forming the *context* for the next computation.
2. **Computation:** The device evaluates the MacroSwarm program over the constructed context, generating *messages* for further communication and the actuation intent.
3. **Actuation:** Based on the computed intent, the device updates its local state and transmits the corresponding message to its neighbors.

This continuous, asynchronous evaluation of MacroSwarm scripts is key to the emergence of complex behavior (e.g., flocking and formation patterns, see Fig. 1).

In MacroSwarm, actuation is represented as a velocity vector sent to each robot. The robot then converts this vector into the appropriate actuation command for execution.

Deployment Model. As described in the system model section, MacroSwarm adopts a peer-to-peer architecture. Nonetheless, several deployment approaches have been explored over the years, enabling the system to operate in a centralized, distributed, or hybrid manner. Early studies focused on general architectures, which eventually evolved into a more complex deployment model known as *pulverization* [16]. In this model, each node is conceptualized as comprising five main components: sensors, actuators, state, behavior (where the program is actually executed), and communication (which defines the neighborhood policies). In conclusion, this flexibility allows for deployment in both fully decentralized setups and centralized server-based configurations, thus broadening the range of possible applications and solutions.

2.3 Related Works

The challenge of programming and coordinating teams of robots has spurred the development of diverse methodologies and tools. Traditional approaches often require low-level control with C/C++ for hardware interfacing, while simulation environments typically leverage high-level languages like Java and Python [6,21, 26].

Numerous frameworks and architectures aim to simplify this development process. The Robot Operating System (ROS) serves as a widely adopted foundation, providing a rich set of tools and libraries that can be extended for multi-robot coordination [1]. Domain-specific solutions include ALLIANCE for heterogeneous swarm control [24], actor-based frameworks for cooperative task programming [31], and specialized languages like Buzz designed specifically for practical swarm robotics programming [26]. Frameworks such as Voltron focus on team coordination [21], while logic-based approaches like Meld enable programming for modular robot ensembles [7]. Simulation environments play a crucial role in robotics development, with tools enabling large-scale testing before physical deployment [19,27,29]. However, bridging the persistent simulation-to-reality gap remains a critical challenge, necessitating practical validation on physical hardware [14].

Our work contributes to this landscape by providing a practical demonstrator toolchain for robot team coordination that bridges the gap between simulation and physical deployment. The toolchain enables accessible execution of aggregate programs on real robots, making it possible to create effective demonstrations of aggregate computing concepts for both technical and non-technical audiences. Moreover our demonstrator prioritizes hardware abstraction and extensibility through a modular architecture that accommodates diverse localization methods and robot platforms.

3 Demonstrator Description

In this section, we first describe the requirements needed to deploy an effective aggregate system in a real team of robots for a demonstration based on the system model of MacroSwarm. Then, we present the architecture of our framework, which is designed to be flexible and adaptable to different deployment scenarios, ranging from centralized to fully distributed, thanks to the flexibility of the deployment model of MacroSwarm. The framework, based on ScaFi, is free, open-source, and publicly available on GitHub [5] under a permissive license.

3.1 Requirements

An aggregate program for a team of robots must satisfy several requirements to be effective in coordinating robots. In the development of our demonstrator, we identified the following requirements:

***R1**: Robot Positioning System.* Several aggregate computing algorithms rely on the knowledge of the relative position of the robots in the team, like the gradient computation (used to shared information) or several spatial formation pattern (like the circle or the line) discussed in MacroSwarm. To satisfy this requirement, it should exist a way to localize the robots in the environment, e.g., using GPS in the case of outdoor environments or using a camera system and marker in the case of indoor environments. These positions should also consider the orientation of the robots, as it is essential for the correct execution of the formation patterns, and the identifier, since the robots should be able to recognize each other.

***R2**: Homogenous Aggregate Program Loading.* Typically, in aggregate computing, all robots in the team compute the same program. This is due to the inherent nature of macro-programming, which encodes the collective behavior, letting the interpreter/compiler break it down into the local behavior of each robot. To satisfy this requirement, the framework must provide a mechanism to deploy the same program across all robots in the team. This deployment should support runtime updates without requiring system shutdown, enabling seamless program evolution during operation. While transient states may occur during updates where robots temporarily execute different program versions, the aggregate abstraction handles this by treating robots running older versions as external entities until synchronization completes.

***R3**: Neighborhood Policy.* Another fundamental concept in aggregate computing is the definition of a neighborhood, namely, the set of robots a robot can communicate directly with. To support aggregate computations, the framework must define the neighborhood relationship between robots. It may be based on spatial distance, connectivity, or other criteria.

***R4**: Actuation Agnosticism.* The aggregate computing evaluation should not be tied to a specific actuation system or robot. Instead, the same aggregate program should be able to run on different robots with different actuation systems. In MacroSwarm, the actuation system is abstracted away: the idea is to output

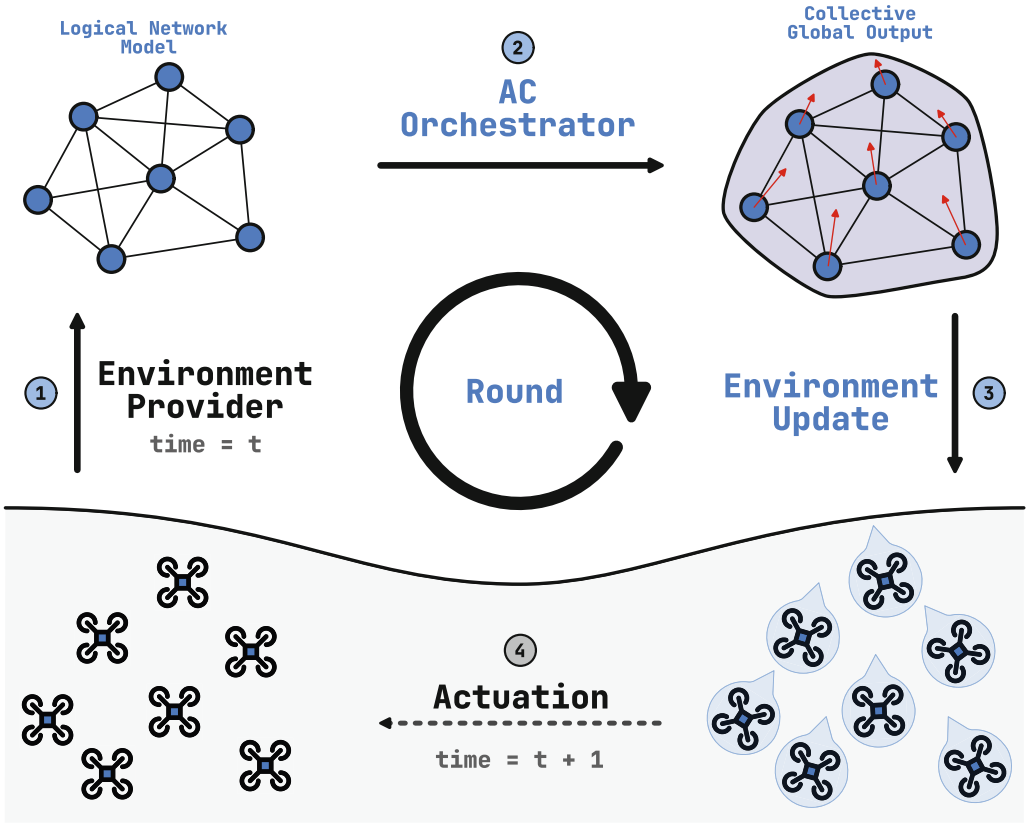

Fig. 2. Framework architecture showing the system workflow. Steps 1–3 represent the information processing pipeline: from environment detection through program evaluation to command execution, while step 4 illustrates the physical actuation that modifies the environment state.

an "actuation intent"–say, just a velocity vector–that is then sent to the robot, which is responsible for executing it by converting it into the actual actuation.

R5*: Platform Agnosticism.* Any robot should be able to join the team and execute the aggregate program, regardless of its harware platform and software configuration. This requirement is an extension and generalization of **R4**. The framework should support both robot-local and remote execution of the aggregate program, and robots must be allowed to join and leave the team at any time.

3.2 Architecture

To satisfy the requirements introduced in Sect. 3.1, the proposed architecture provides the following main abstractions: `Environment`, `Robot`, `Orchestrator`, `EnvironmentProvider`, and `EnvironmentUpdate`. In the following, we describe each of these abstractions and their role in the framework.

Environment. This abstraction provides a digital representation of the deployment context, encompassing all information required for aggregate program evaluation and context creation. Specifically, it maintains the state of the robot team (**R2**), their neighborhood relationships (**R3**), and additional metadata essential for program execution.

Robot. The `Robot` abstraction encapsulates the digital identity of each robot in the `Environment`, along with communication-related metadata required for system operation. For centralized deployments, this includes network parameters such as IP addresses. This representation maintains hardware independence, adhering to requirement **R5**.

EnvironmentProvider. This component addresses requirement **R1** by generating and maintaining the `Environment` representation of the robot collective. The implementation may leverage various robot localization technologies, from camera-based systems to GPS. Additionally, it handles neighborhood policy definition (**R3**), which may be based on physical distance or alternative criteria.

Orchestrator. Following requirement **R2**, this component manages program distribution and execution across the robot collective. In centralized deployments, it handles program loading and evaluation on the server for the entire team. Starting from the `Environment` representation, the `Orchestrator` creates the context for each robot, and then it computes velocity vectors for robot actuation. The architecture also supports message passing to enable distributed scenarios.

EnvironmentUpdate. This component implements requirement **R4** by managing actuation. It translates the computed velocity vector into robot-specific commands, ensuring compatibility across heterogeneous robot platforms.

Centralized Versus Distributed Deployment. While our demonstrator is based on a centralized *implementation*, the architecture supports distributed execution, where each robot can host its own `Orchestrator`, `EnvironmentProvider` and `EnvironmentUpdate` instances. Indeed, it is crucial to distinguish between (i) the deployment and execution implementation of the aggregate system and (ii) the logically decentralized computation implied by the aggregate program, which is conceptually the same despite a centralized `Orchestrator` is used in place of robot-to-robot communication and local program evaluation. The coordination patterns, spatial formations and self-healing behaviors, arise from field-based computations where each robot's actions derive from neighbor interactions encoded in the program. The aggregate computing abstraction ensures these coordination mechanisms remain consistent whether executed centrally or in a distributed manner [12,16].

3.3 Behavior and Interaction

In this demonstrator, we leverage a centralized deployment, where a server performs the core computation and robots are responsible for performing actua-

tion. In particular, the computation is divided into three main steps (depicted in Fig. 2):

1. The `EnvironmentProvider` creates the `Environment` representation of the robots in the team, using a camera system to localize them in the environment.
2. The `Orchestrator` loads the aggregate program on the central server and evaluates that program for all the robots in the team.
3. The `EnvironmentUpdate` translates the velocity vectors resulting from the aggregate program into robot-specific commands.

Although these macroscopic steps are conceptually divided into sequential phases, in general their practical execution happens concurrently—sometimes necessarily. For instance, in our case, the `EnvironmentProvider` must execute in parallel with the `Orchestrator`, as the cameras provide a continuous stream of data flowing in faster than the `Orchestrator` and `EnvironmentUpdate` can process (mostly because of network-related latencies). Executing the components sequentially would force us to skip frames, losing valuable information about the system state (for instance, hindering noise detection).

4 Live Demo at the European Researchers' Night

We showcase the effectiveness of our demonstrator during the European Researchers' Night[1]. In the following, we describe the demo objectives (Sect. 4.1), setup (Sect. 4.2), implementation details (Sect. 4.3) and the results (Sect. 4.4).

4.1 Objectives

In addition to satisfying the technical requirements outlined in Sect. 3.1, this demonstration aims to make the topics of aggregate computing accessible and easy to understand. By showcasing an engaging solution for coordinating robot teams, we illustrate the practical potential of the paradigm, demonstrating how intuitive spatial formations and self-healing behavior emerges from simple, collective computations. In particular, we aim to highlight the following properties of aggregate computing:

G1–*Spatial formation*: among the patterns available in MacroSwarm, we chose to showcase spatial formation because it is a non-trivial, visually appealing behavior requiring coordination among robots easily understood by a general audience.

G2–*Self-healing*; aggregate programming is inherently robust to disturbances and node failures—see recent work on quasi-static approximation [4]. To check this property in a real-world scenario, we introduced controlled perturbations–such as unplanned robot movements or temporary node removals– and observed whether the system could autonomously reconfigure

[1] https://www.nottedeiricercatori-society.eu/eventi/ecosistemi-coordinati-di-robot-programmabili, archived for future reference at https://archive.is/yixcd.

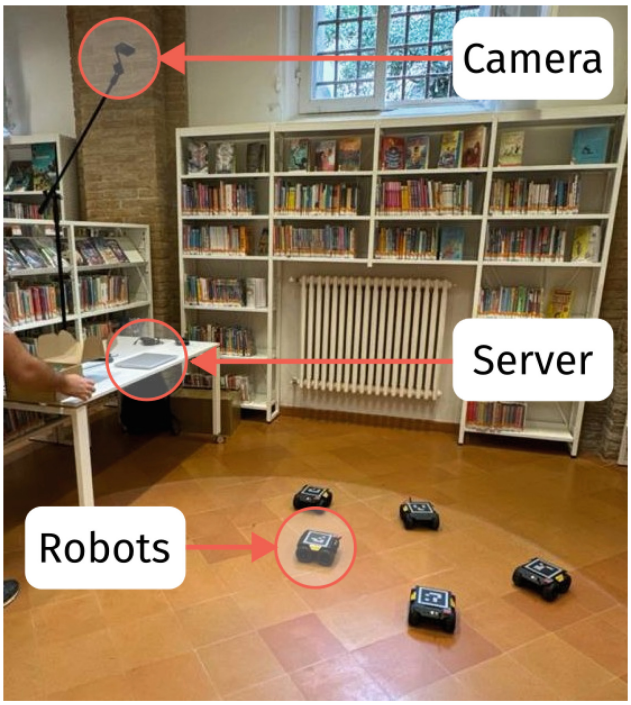

Fig. 3. Setup for the researchers' night. A camera system was used to localize the robots in the environment with the ArUco marker system. The central server was responsible for the computation, while the robots executed the commands sent by the `EnvironmentUpdate` component.

to restore the intended formation. To make this part interactive and engaging, we invited the audience (including children) to cause perturbations to the system.

G3–*Scalability and Topology independence*: Aggregate programs should perform effectively regardless of team size or configuration. The demonstrator highlights this adaptability by dynamically varying the number of robots, thereby confirming that the coordination logic remains robust and consistent even as the network topology changes.

4.2 Setup

For this demonstration, we used a centralized setup (see Fig. 3) in which the primary computations were executed on a central server. The robots, with limited computation capabilities, were instead responsible for the execution of the commands sent by the `EnvironmentUpdate` component. Specifically, the demo was performed with a team of five Rover Wave robots[2], based on the ESP32 microcontroller—a low-cost device with Wi-Fi capabilities.

The `EnvironmentProvider` used a camera system to localize the robots in the environment (**R1**), leveraging the ArUco marker system for detecting mark-

[2] https://archive.is/D3j0T.

ers in images[3], in conjunction with the computer vision library OpenCV [11]. Additionally, the `EnvironmentProvider` defined the neighborhood policy based on the detected robot positions. For our demo, we approximated the camera field of view (1920×1080 pixels) as an equirectangular projection of the arena floor, then computed distances based on the pixel coordinates of the markers, considering as neighbors those within 200 pixels of each robot. Although imprecise as a general-purpose indoor localization system, this approach was precise enough for the demonstration's purposes (**R3**).

Subsequently, the `Orchestrator` loaded the aggregate program on the central server (**R2**) and evaluated the program for all robots following the execution model described in Sect. 2. Finally, the `EnvironmentUpdate` component transmitted the respecting command to each robot. In this context, the velocity vector is translated from Cartesian coordinates into polar coordinates (Euclidean norm and angle relative to the camera horizontal axis), then sent to robots, satisfying requirements **R4** and **R5**.

4.3 Implementation Details

For this demonstration, we utilize the circle and line formations. The circle pattern arranges robots equidistantly around a central leader node, while the line formation positions robots sequentially along a straight path. The code in Fig. 4 will produce a circle or line formation based on the effective implementation of `computeFormation`. The `computeFormation` function calculates the target positions for each robot based on the selected formation (circle or line) and the robots' relative positions. The leader robot acts as the center of the formation, and the other robots move to their assigned positions around it—see Fig. 5 for the circle formation.

4.4 Results

The experimental results are depicted in Fig. 5, 6 and 7. As shown, the robots effectively converged to the desired circle or line formation, demonstrating the framework's ability to coordinate the team (**G1**). To verify the topology independence property, we conducted experiments with varying team sizes (from 3 to 5 robots), observing consistent formation behavior across different configurations (**G3**). Finally, we tested the self-healing capabilities (**G2**) by introducing controlled failures (moving robots as in Fig. 6 and changing the robot team size as in Fig. 7) during formation, confirming the system's ability to automatically reconfigure and maintain the desired geometric pattern.

As an additional challenge for the system, we let the audience perturb the robots by moving them around, finding that, if given enough time, the swarm was able to recover the formation even after significant disturbances. Having children interact with the system was particularly engaging: they tended to play

[3] https://archive.is/lpTB2.

```
class Formation extends BaseMovement {
    override def main(): = {
        // Select a leader device that will be the center of the circle
        val leader = mid() == leaderSelected
        // Compute distance from leader using gradient
        val potential = gradient(leader)
        // Share positions relative to the leader across the network
        val positions = collectPositions(potential)
        // Calculate target positions on the circle for each device
        val targetPositions = branch(leader){
            computeFormation (positions)
        } { (Map.empty) }
        // Share target positions from leader to all devices
        val myTarget = getMyTargetPosition(leader, targetPositions)
        // Move towards target or stop if close enough
        moveToTarget(myTarget)
    }
}
```

Fig. 4. Programming a robot formation in MacroSwarm.

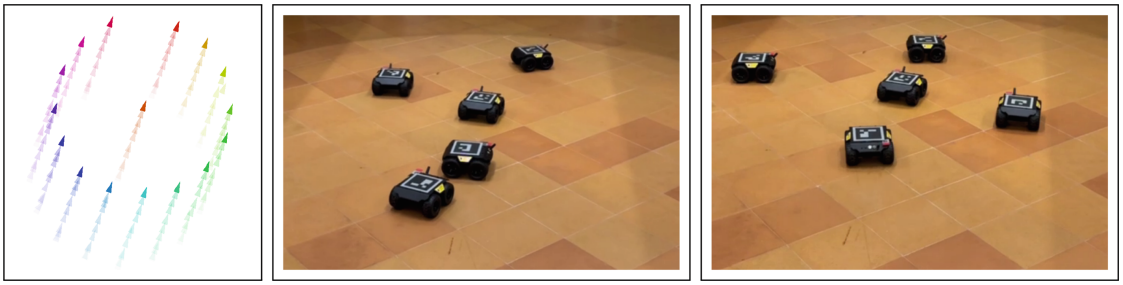

Fig. 5. Example of the circle formation performed by the robots. On the left there is the computation performed in simulation, in the middle the robots start in a random position, and then they converge to the circle formation (right).

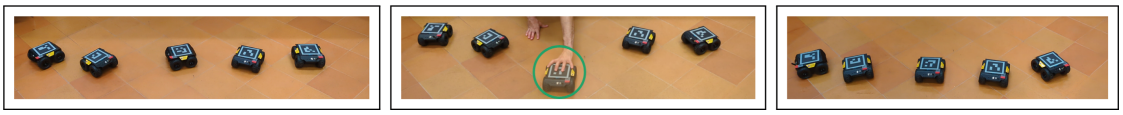

Fig. 6. Example of self-healing in a line formation: one robot in the middle (green circle) is perturbed (i.e., moved towards) and then eventually come back to a line formation (right). (Color figure online)

with the robots, disturbing them continually and preventing the pattern formation completion, a condition that was not foreseen in the initial design of the demonstration. This kind of interaction, however, was beneficial for the demonstration, as we could also observe how the system reacted to continuous noise: the swarm approximated the formation without diverging, showing that the system converges to stability even with continued perturbations.

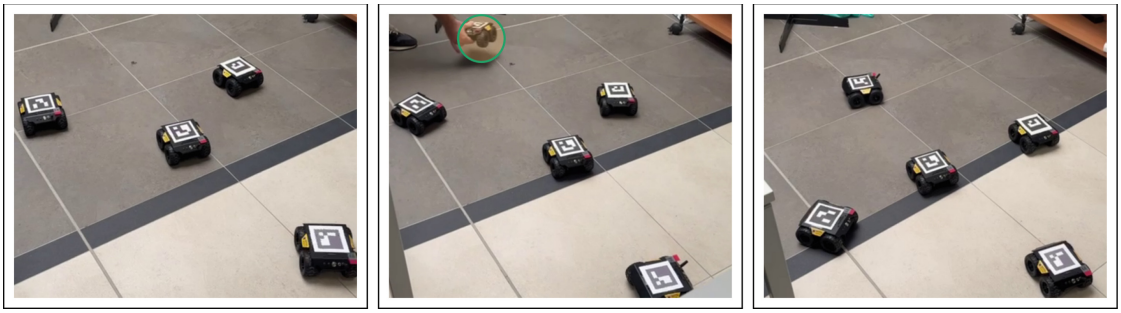

Fig. 7. Example of self-healing in a circle-formation: a new robot is added to the team (middle, green circle), and the other robots reconfigure to maintain the circle formation. (Color figure online)

5 Conclusion and Future Work

In this paper, we presented a demonstrator for self-organizing robot teams based on aggregate computing principles. Our artefact, built upon ScaFi and MacroSwarm, bridges the gap between theoretical foundations and practical implementations, providing a flexible and modular architecture that satisfies key requirements for real-world deployments. Through our demonstration at the European Researchers' Night, we validated the effectiveness of our approach, successfully showcasing fundamental aggregate computing properties including self-healing capabilities, topology independence, and spatial formation patterns. Additionally, the demonstration engaged a broad audience, including families and children.

Future work will focus on expanding the framework's capabilities in several directions: (i) supporting heterogeneous robot teams including ground robots and UAVs, (ii) implementing more complex spatial formations and coordination patterns, (iii) supporting mixed reality to show high-density scenarios in a more effective way, (iv) developing user-friendly interfaces for programming and monitoring aggregate systems, (v) investigating robustness against adversarial moving obstacles that may disrupt coordination and pattern formation.

References

1. micros_swarm_framework - ROS Wiki — wiki.ros.org. https://wiki.ros.org/micros_swarm_framework. Accessed 15 Apr 2025
2. Abowd, G.D.: Beyond Weiser: from ubiquitous to collective computing. Computer **49**(1), 17–23 (2016). https://doi.org/10.1109/MC.2016.22
3. Aguzzi, G., Casadei, R., Pianini, D., Viroli, M.: Dynamic decentralization domains for the internet of things. IEEE Internet Comput. **26**(6), 16–23 (2022). https://doi.org/10.1109/MIC.2022.3216753
4. Aguzzi, G., Casadei, R., Viroli, M.: Macroswarm: a field-based compositional framework for swarm programming, vol. abs/2401.10969 (2024). https://doi.org/10.48550/ARXIV.2401.10969

5. Aguzzi, G., Farabegoli, N., Domini, D.: Cric96/researcher-night-demo: artefact – COORDINATION 2025 (kick the tiers) (2025). https://doi.org/10.5281/ZENODO.14938868
6. de Andrade, E.M., Fernandes, A.C., Sales, J.S.: PySwarming: a research toolkit for swarm robotics. J. Open Source Softw. **8**(90), 5647 (2023). https://doi.org/10.21105/JOSS.05647
7. Ashley-Rollman, M.P., Goldstein, S.C., Lee, P., Mowry, T.C., Pillai, P.: Meld: a declarative approach to programming ensembles. In: 2007 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 2794–2800. IEEE (2007). https://doi.org/10.1109/IROS.2007.4399480
8. Audrito, G., Terraneo, F., Fornaciari, W.: FCPP+Miosix: scaling aggregate programming to embedded systems. IEEE Trans. Parallel Distrib. Syst. **34**(3), 869–880 (2023). https://doi.org/10.1109/TPDS.2022.3232633
9. Audrito, G., Torta, G.: FCPP to aggregate them all. Sci. Comput. Program. **231**, 103026 (2024). https://doi.org/10.1016/J.SCICO.2023.103026
10. Audrito, G., Viroli, M., Damiani, F., Pianini, D., Beal, J.: A higher-order calculus of computational fields. ACM Trans. Comput. Log. **20**(1), 5:1–5:55 (2019). https://doi.org/10.1145/3285956
11. Bradski, G., Kaehler, A.: Learning OpenCV: Computer vision with the OpenCV library. O' Reilly Media, Inc. (2008)
12. Casadei, R., Viroli, M.: Declarative macro-programming of collective systems with aggregate computing: an experience report. In: PPDP 2024. ACM, New York (2024). https://doi.org/10.1145/3678232.3678235
13. Casadei, R., Viroli, M., Aguzzi, G., Pianini, D.: Scafi: a Scala DSL and toolkit for aggregate programming. SoftwareX **20**, 101248 (2022). https://doi.org/10.1016/J.SOFTX.2022.101248
14. Dias, P., Silva, M.C., Filho, G., Vargas, P.A., Cota, L.P., Pessin, G.: Swarm robotics: a perspective on the latest reviewed concepts and applications. Sensors **21**(6), 2062 (2021). https://doi.org/10.3390/S21062062
15. Dijkstra, E.W.: Self-stabilizing systems in spite of distributed control. Commun. ACM **17**(11), 643–644 (1974). https://doi.org/10.1145/361179.361202
16. Farabegoli, N., Pianini, D., Casadei, R., Viroli, M.: Scalability through Pulverisation: declarative deployment reconfiguration at runtime. Future Gener. Comput. Syst. **161**, 545–558 (2024). https://doi.org/10.1016/J.FUTURE.2024.07.042
17. Grushchak, D., et al.: Decentralized multi-drone coordination for wildlife video acquisition. In: IEEE International Conference on Autonomic Computing and Self-Organizing Systems, ACSOS 2024, Aarhus, Denmark, 16–20 September 2024, pp. 31–40. IEEE (2024). https://doi.org/10.1109/ACSOS61780.2024.00021
18. Mamei, M., Zambonelli, F., Leonardi, L.: Co-fields: a physically inspired approach to motion coordination. IEEE Pervasive Comput. **3**(2), 52–61 (2004). https://doi.org/10.1109/MPRV.2004.1316820
19. Michel, O.: Cyberbotics ltd. webots™: professional mobile robot simulation. Int. J. Adv. Robot. Syst. **1**(1) (2004). https://doi.org/10.5772/5618
20. Mone, G.: Rise of the swarm. Commun. ACM **56**(3), 16–17 (2013). https://doi.org/10.1145/2428556.2428562
21. Mottola, L., Moretta, M., Whitehouse, K., Ghezzi, C.: Team-level programming of drone sensor networks. In: Proceedings of the 12th ACM Conference on Embedded Network Sensor Systems (SenSys 2014), pp. 177–190. ACM (2014). https://doi.org/10.1145/2668332.2668353

22. Newton, R., Morrisett, G., Welsh, M.: The regiment macroprogramming system. In: 6th International Symposium on Information Processing in Sensor Networks, IPSN 2007, pp. 489–498. IEEE (2007). https://doi.org/10.1145/1236360.1236422
23. Olechowski, A., Eppinger, S.D., Joglekar, N.: Technology readiness levels at 40: a study of state-of-the-art use, challenges, and opportunities. In: 2015 Portland International Conference on Management of Engineering and Technology (PICMET), pp. 2084–2094. IEEE (2015). https://doi.org/10.1109/PICMET.2015.7273196
24. Parker, L.E.: ALLIANCE: an architecture for fault tolerant multirobot cooperation. IEEE Trans. Robot. Autom. **14**(2), 220–240 (1998). https://doi.org/10.1109/70.681242
25. Pianini, D., Casadei, R., Viroli, M., Natali, A.: Partitioned integration and coordination via the self-organising coordination regions pattern. Future Gener. Comput. Syst. **114**, 44–68 (2021). https://doi.org/10.1016/J.FUTURE.2020.07.032
26. Pinciroli, C., Beltrame, G.: Buzz: a programming language for robot swarms. IEEE Softw. **33**(4), 97–100 (2016). https://doi.org/10.1109/MS.2016.95
27. Pinciroli, C., et al.: ARGoS: a modular, parallel, multi-engine simulator for multi-robot systems. Swarm Intell. **6**, 271–295 (2012). https://doi.org/10.1007/s11721-012-0072-5
28. Saha, D., Mukherjee, A., Bandyopadhyay, S.: Pervasive computing, pp. 1–37. Springer, Boston (2003). https://doi.org/10.1007/978-1-4615-1143-4_1
29. Valentini, G., et al.: Kilogrid: a novel experimental environment for the Kilobot robot. Swarm Intell. **12**(3), 245–266 (2018). https://doi.org/10.1007/s11721-018-0155-z
30. Viroli, M., Beal, J., Damiani, F., Audrito, G., Casadei, R., Pianini, D.: From distributed coordination to field calculus and aggregate computing. J. Log. Algebraic Methods Program. **109** (2019). https://doi.org/10.1016/J.JLAMP.2019.100486
31. Yi, W., et al.: An actor-based programming framework for swarm robotic systems. In: IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2020, Las Vegas, NV, USA, 24 October 2020–24 January 2021, pp. 8012–8019. IEEE (2020). https://doi.org/10.1109/IROS45743.2020.9341198