Multi-robot systems
Final Project

Jorge Aranda NIP: 800839
Alberto Zafra NIP: 876628

MRGCV - 2023/2024

1 Introduction

In the field of robotics, the ability to control multiple robots in a coordinated way has become increasingly significant. This multi-robot coordination is crucial for a variety of applications such as search and rescue missions, environmental monitoring, entertainment, and industrial automation. One of the key challenges in this domain is the design of control algorithms that ensure the robots maintain a specified formation while moving towards a designated target or enclosing it.

This report presents the development and implementation of a completely distributed enclosing and formation control algorithm that allows several groups of robots to maintain a specific formation while collectively moving towards a target position, reducing computational costs and enhancing scalability.

As a demonstration of the system, a two-tiered enclosing algorithm is proposed. Each agent will perform a two-level control system simultaneously, specifically a local and a global control algorithm. This ensures that each individual belongs to a group and that the centroid of its designated robot group follows a desired trajectory, while adjusting its position to preserve a predefined formation structure. This implementation leverages the Kabsch algorithm for accurate formation maintenance and incorporates control gains to fine-tune the movement of both the centroid and the individual robots within the formation.

On the other hand, to achieve a fully distributed system, each agent is executed on a separate machine from those available in laboratory L1.02 at the university [1]. Furthermore, a combination of both Remote Procedure Call (RPC) [2] and Secure Shell (SSH) [3] communication protocols has been used to exchange information among the robots in real-time. More concretely, each individual shares its local control predictions within a predefined neighborhood, enabling collaborative decision-making and global consensus over the refined enclosing formation. Finally, to perform a graphical representation of the agents' positions and their evolution over time, an external entity receives and records their positions at a predefined frequency.

In the subsequent sections, we delve into the detailed design and implementation of the formation control algorithm, the simulation environment, and the results of our tests. The findings from this report demonstrate the efficacy of the proposed control scheme in maintaining a stable formation while navigating towards a target, highlighting its potential for real-world robotic applications.

2 Approach

The approach involves the development of a formation control system tailored for a multi-agent robotic swarm using Python. This system integrates diverse libraries and tools to enable seamless communication, efficient computation, and effective coordination among the robotic agents. The following subsections delve into key aspects of the system, including the communication infrastructure, system architecture, and control algorithms.

2.1 Communication infrastructure

As it has been aforementioned, each agent in the system is represented by a different device. For this project, the computers in `lab102` were selected, with IP addresses ranging from `155.210.154.191` to `155.210.154.210`. These computers are ideal due to their six cores, which enhance the performance of the system by efficiently handling the concurrent and distributed tasks required by the project.

Communication between devices is facilitated through SSH and RPC protocols, without needing extensive configuration beyond copying the necessary code snippets to each destination device. SSH is utilized to start remote Python scripts that initialize each agent with the desired configuration. Meanwhile, RPC is employed for transmitting information between agents via a secondary thread, ensuring efficient and reliable communication within the swarm.

2.2 Architecture

The system's architecture involves a remote user who connects to the University of Zaragoza's remote server, `central.cps.unizar.es`, via SSH or another preferred communication protocol to initialize the system by launching the `main.py` script. This script is responsible for starting and launching all the agents or workers, which are specified in configuration files passed as arguments to the `main.py` script, via SSH.

Once launched, the workers located in `lab102` initialize their designated agents with the received configuration and establish direct communication with neighboring agents or bots as indicated in the communication graph stored in the `communication.txt` file. The peer-to-peer communications between bots use RPC calls within the LAN network to exchange necessary information. During the agents' operation, the remote server that initialized them logs the state of each agent at each iteration of the simulation using the RPC protocol. This logging facilitates later graphical representation of the agents' evolution.

The state evolution log is necessary because the remote server lacks the plotting libraries. Therefore, the graphical representation of the system's evolution is generated locally by the user with the data obtained from the remote server.

In summary, as illustrated in Figure 1, the user connects to the university's remote server to initialize the system, which runs the simulation in a fully distributed and concurrent manner using the devices available in `lab102`. After the simulation concludes, the user visualizes a graphical representation of the simulation locally on their own device.

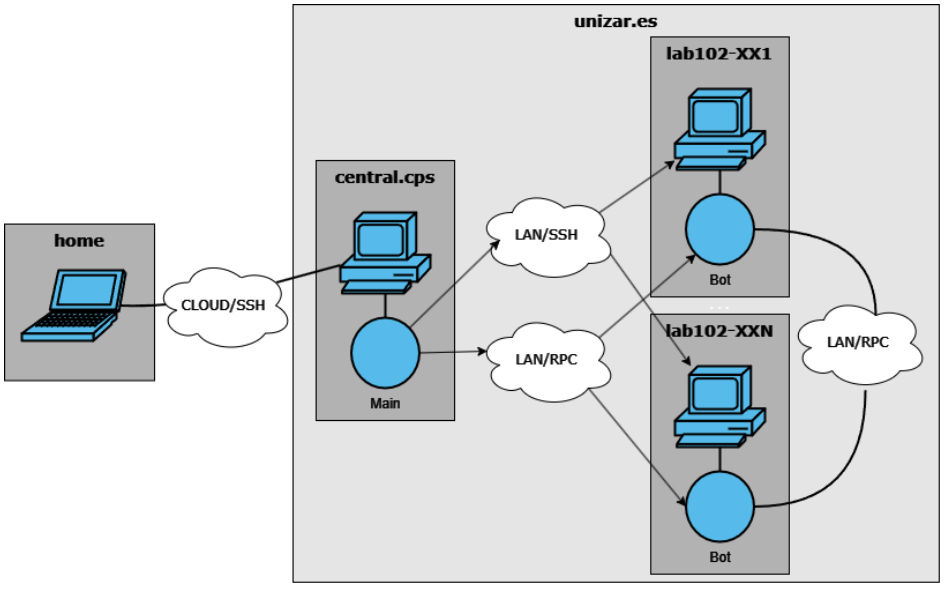


Figure 1: System architecture

2.3 Agents

Each agent is the responsible of performing both, the control algorithms and the communications with other agents, simultaneously in concurrent threads. Hence, during start-up each agent would initialize the control system described in Section 2.4 and the communication algorithm described below.

In order to establish a direct communication between them, the agents use RPC functions to transmit their knowledge. This knowledge includes the believed group and external agents positions as well as their known current position. Nevertheless, since the number of connections between agents is limited according to the previously mentioned communication graph, Lamport logical clocks have been used [5]. This system spreads the position information across agents without the need of direct connections. For this purpose, the information shared is combined with a time stamp in a matrix form of size ($number\_of\_groups \times agents\_per\_group \times 3$), as presented in equation 1.

$$\begin{pmatrix} x_1 & y_1 & time\_stamp_1 \\ ... & ... & ... \\ x_n & y_n & time\_stamp_n \end{pmatrix} \tag{1}$$

Hence, when a bot receives a communication from another agent, the listener will only update its knowledge with the positions that contain a higher timestamp, indicating that those are more likely to be the most accurate, as the current position of the agent will always have the highest timestamp. Conversely, the group positions are not transmitted between agents, as they are computed by consensus by averaging the transmitted agent positions.

It is worth noting, that as agents communicate with each other periodically and iteratively to have the most updated information in each time-step, whether directly from the current connection or indirectly from its knowledge, deadlocks may occur. This is because when two agents communicate, the position variables are locked during an infinitesimal amount of time. However, after a thoughtful analysis, the probability of reaching a deadlock has been reduced by using redundant variables that contain a copy of the position values, and that are updated frequently. An example of the aforementioned solution is presented in figure 2. The left image shows the uncorrected system reaching a deadlock, the middle image illustrates the possible cause of the deadlock, where two or more agents attempt to establish direct communication and lock their respective shared memory. Finally, the right side of the image demonstrates how the corrected system, which uses a mutex to copy the position variable to a redundant variable, ensures both the integrity and availability of the data. This approach might introduce a slight delay of a few milliseconds due to the copying of the agent's knowledge, which is acceptable as it does not compromise the system's integrity.

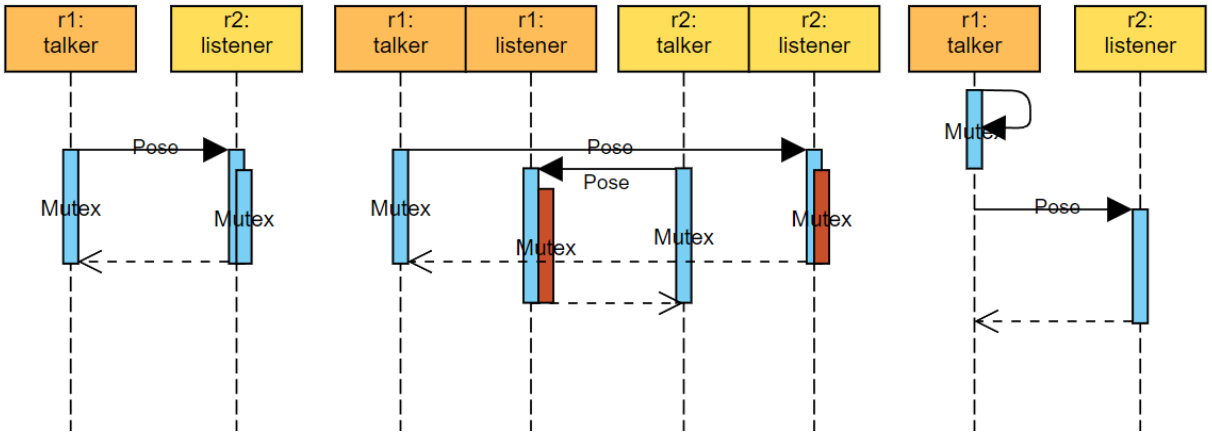


Figure 2: Left: normal information transmission. Middle: case of deadlock. Right: fixed system

Finally, in some cases it could happen that the system terminates without correctly closing the connections

between the remote devices. Therefore, a remote shutdown script, named as *killer.py*, has been implemented. This script kills any of the processes that are started from the system, ensuring that no "zombies" are present within the system's architecture.

2.4 System Control

As it has been aforementioned, the each agent performs two different control algorithms (a global and a local control) simultaneously, which govern the behaviour and movement of individual robotic agents within the swarm. This means, that whereas the agent aims to find the optimal trajectory to perform a group formation, it also moves accordingly to displace the group centroid, computed by using a consensus algorithm, to reach a global formation between groups. This combination is necessary in order to achieve the behaviour illustrated in Figure 3.

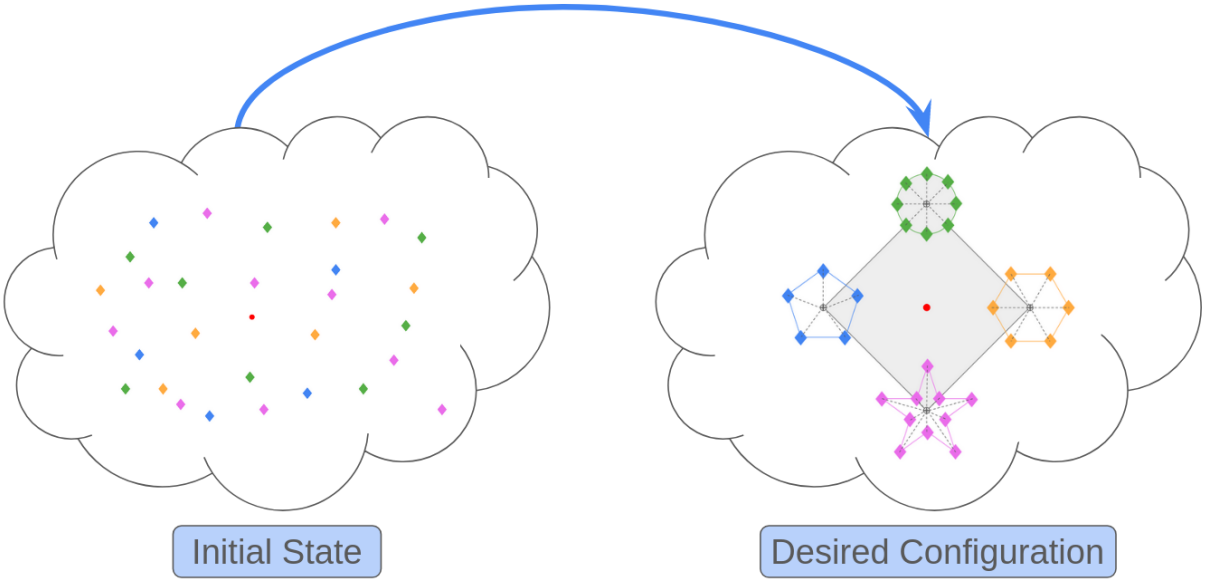


Figure 3: Desired Behaviour

2.4.1 Global Control Algorithm

As it has been previously stated, the global control algorithm employed in the system orchestrates the coordinated movement of individual robotic agents to achieve a desired formation pattern between groups, as presented in figure 4.

As its core, the Kabsch algorithm is utilized to compute the optimal rotation matrix aligning the current positions of the group centroids with their desired relative positions within the formation [4, 6]. The desired relative positions (C) and the current inter-robot distances (Q) are represented by the matrix A, which is decomposed using singular value decomposition (SVD) as shown in equation 2. Here, U, S, and V represent the left singular vectors, the diagonal matrix of singular values, and the right singular vectors, respectively. Subsequently, the rotation matrix (R) is computed using equation 3, where D is a diagonal matrix with the sign of the determinant of VU^T.

$$A = C^T Q = USV^T \tag{2}$$

$$R = VDU^T = V \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & d \end{pmatrix} U^T \tag{3}$$

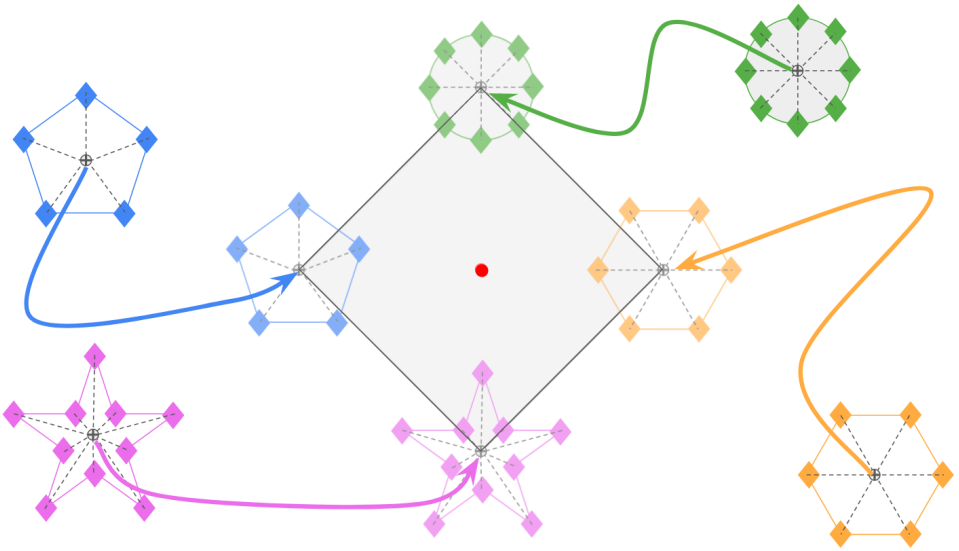


Figure 4: Global Control

Hence, the general control algorithm, that guides each robot towards its target position within the formation, can be generated by applying the previously computed rotation matrix (R) to the difference between the current (q_{Ni}) and desired (c_{Ni}) relative positions, computed by the shape formation algorithm presented in Section 2.4.3; and by adjusting it with a customized control gain (Kc), as presented in equation 4.

$$\dot{q}_i = K_c(q_{Ni} - Rc_{Ni}) \tag{4}$$

Additionally, the algorithm incorporates orbiting control functionality, which allows robots to dynamically adjust their positions based on the movements of neighboring robots, thereby enhancing the system's adaptability and resilience in dynamic environments. This effect is achieved by adding a velocity component proportional to the relative position difference between the controlled robot (q_i) and its neighbor (q_{i+1}), and adjusting it with an additional control gain (K_g), as shown in equation 5. This functionality can be activated or deactivated at the agent's startup. However, to avoid possible malfunctions, all agents should have the same setting, as it determines whether the groups will rotate around the target or not.

$$\dot{q}_i = K_c(q_{Ni} - Rc_{Ni}) - K_g(q_{i+1} - q_i) \tag{5}$$

Finally, it is worth mentioning that the global control algorithm computes the behavior of the consensual average center of the group during the simulation, using a global target as the center of the formation. The results of this global control algorithm are then fed to the local control algorithm, which adjusts each agent's trajectory based on the computed group centroid. This ensures that the center position of the group remains within the designated global trajectory.

2.4.2 Local Control Algorithm

Once the global control algorithm computes the desired group centroid position, the local control algorithm adjusts the trajectories of individual agents within the swarm to ensure formation coherence and alignment with the global target, as shown in Figure 5.

The local control algorithm operates by receiving the desired group's centroid position from the global control algorithm and combining it with the desired shape formation. Each agent calculates its position relative to this centroid and adjusts its velocity and direction to maintain its designated position within the formation, using the control equations 4 and 5, presented in Section 2.4.1. Akin to the global control, the local control algorithm incorporates an orbiting control functionality, which enables group rotation. However, unlike the global control algorithm, only the robots within a specific group need to have the same setting,

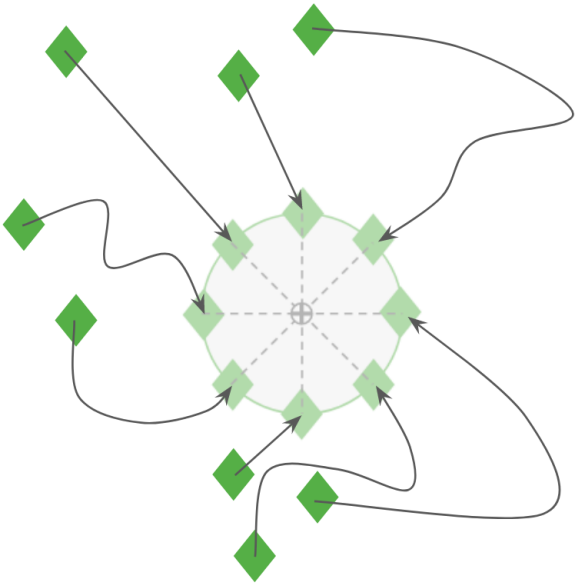


Figure 5: Local Control

allowing the local rotation of a single group without affecting the behavior of other groups.

Additionally, the local control algorithm takes into account the positions of neighboring robots to avoid collisions and maintain a cohesive formation. By continuously updating each agent's trajectory based on both the global centroid and local interactions, the local control algorithm ensures that the entire swarm moves harmoniously towards the global target while adapting to dynamic changes in the environment.

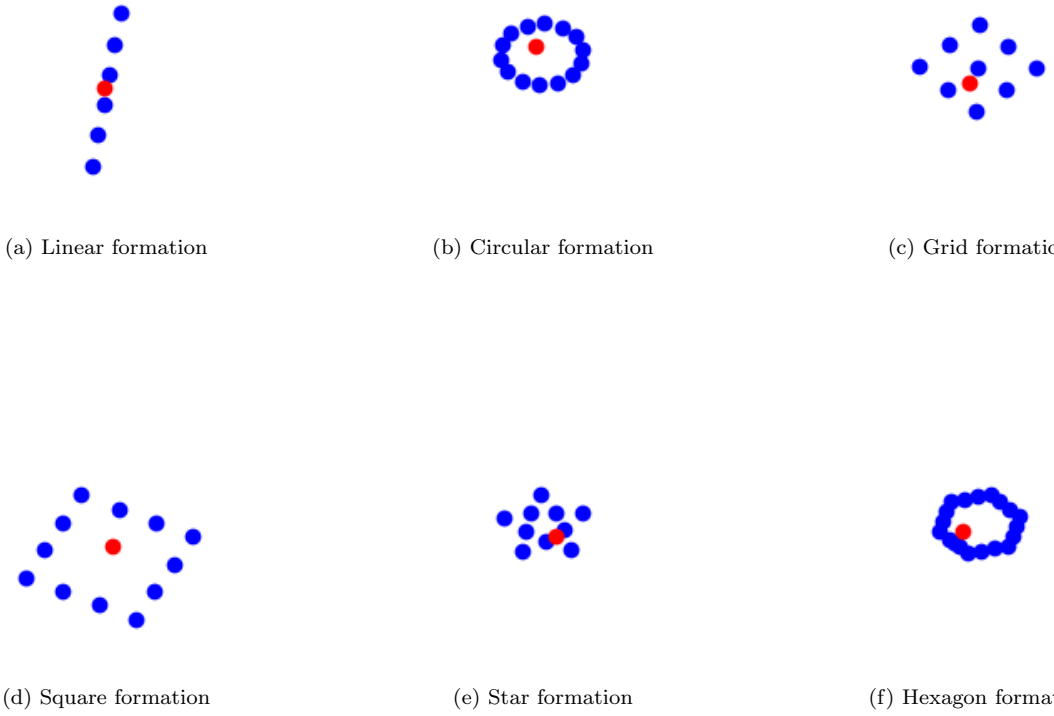
2.4.3 Shape Formation

The shape formation mechanism is a critical component of the swarm control system, enabling the group of robots to organize themselves into predefined geometrical patterns. This functionality enhances the swarm's ability to perform coordinated tasks and navigate complex environments. Moreover, the shape formation algorithm is adaptive, allowing the swarm to respond to changes in the environment or task requirements. As the global target or environmental conditions change, the local control algorithm continuously updates the robots' trajectories to maintain the formation.

The desired shapes are defined as a set of relative positions that each robot should occupy within the formation. These positions are specified relative to the group centroid, which is determined by the global control algorithm in the case of the local control algorithm, or relative to the target position in the case of the global control algorithm. The desired shapes can vary depending on the task requirements and environmental constraints. In this case the following shapes have been implemented:

- **Line**: the robots will perform a linear formation keeping the robot target in the median point of the line, as presented in Figure 6a.
- **Circle**: the robots will perform a circular formation keeping the robot target in the centre point of the circle, as presented in Figure 6b.
- **Grid**: the robots will perform a grid formation keeping the robot target in the mean point of the grid, as presented in Figure 6c.
- **Square**: the robots will perform a square formation keeping the robot target in the mean point of the square, as presented in Figure 6d.

- **Star**: the robots will perform a star formation keeping the robot target in the median point of the star, as presented in Figure 6e.
- **Hexagon**: the robots will perform a hexagonal formation keeping the robot target in the median point of the hexagon, as presented in Figure 6f.



(a) Linear formation (b) Circular formation (c) Grid formation

(d) Square formation (e) Star formation (f) Hexagon formation

Figure 6: Different swarm formations

3 Results

Several tests have been performed to validate the system's performance and optimise hyper-parameters. These tests were executed both locally and remotely using the *lab102* facilities. This section presents the results obtained from these tests, demonstrating the effectiveness of the proposed control algorithms in achieving and maintaining desired formations while moving towards target positions.

To evaluate the system's ability to maintain a specified formation, a static formation test was conducted. In this scenario, the robotic agents were required to arrange themselves into predefined global and local formations. Specifically, the groups of robots were organised into a line formation relative to a central target, with each group forming a circular sub-formation, which was simplified to a triangle due to the limited number of available agents. Each group was composed of three different robots initialised at random positions, with both global and local rotations deactivated. Moreover, the robots followed a predefined communication graph, shown in Figure 7, where they could only communicate with the other members of their group and one or two outsiders from other groups. Finally, the global target was located at the center of the grid (0,0) and remained stationary throughout the test.

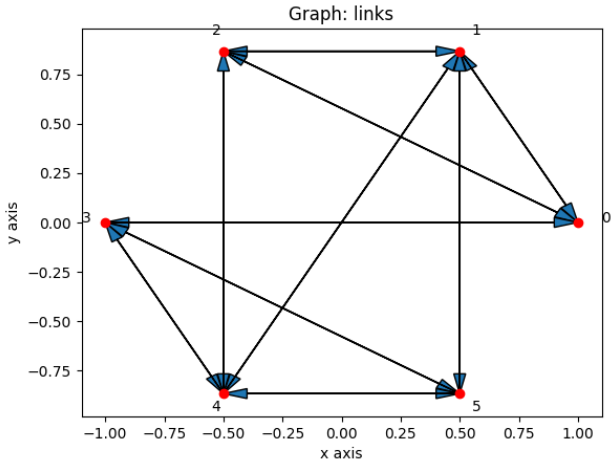


Figure 7: Communication graph

Figure 8 presents the static formation achieved by the robotic agents. The agents successfully maintained the triangular formations within their groups while the groups themselves aligned linearly with respect to the central target, demonstrating the capability of the proposed control algorithms to enforce and sustain precise formations. This test confirms the robustness of the control system in maintaining both local and global formation structures, even with the simplified triangle sub-formations due to the limited number of agents. The results also highlight the effectiveness of the communication strategy, ensuring coordinated behavior across the swarm. Throughout the test, the positional deviations of individual agents from their target points were minimal, underscoring the precision of the control algorithms.

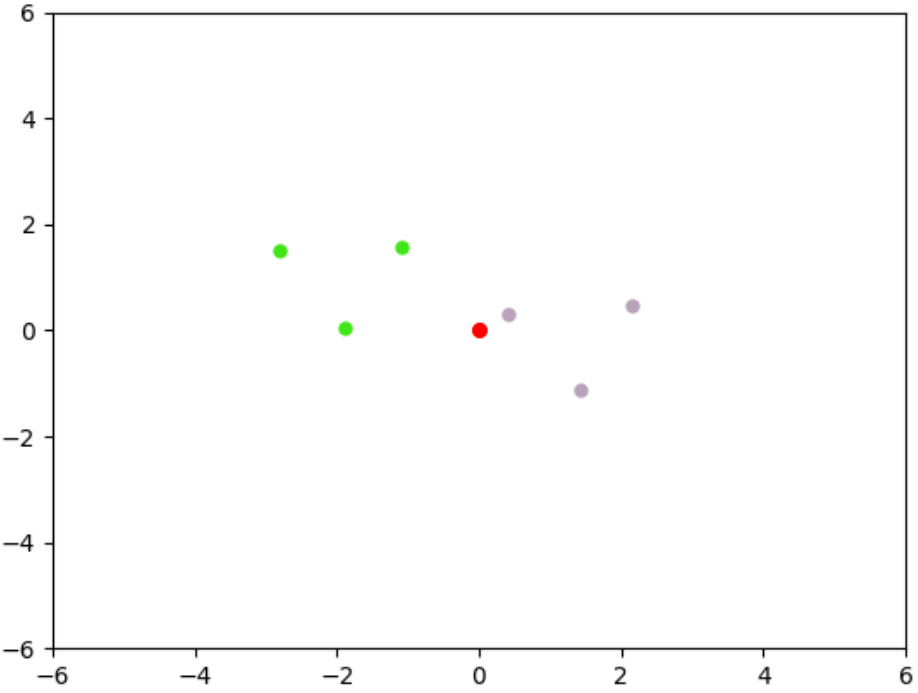


Figure 8: Static formation

4 Current Limitations

Due to user activity constraints in `lab102`, it was impossible to invoke a large number of robots simultaneously. However, the system's behaviour was verified by introducing a smaller number of robots during simulation. The system's scalability allows modifications to the simulation by making minor adjustments to the .txt parameter files. Therefore, it can be stated that the system has been proven to be scalable in all aspects (control and communications), despite not being able to evaluate the system with larger batches of robots in the `lab102` distributed environment due to user limitations and resource sharing.

5 Conclussions

The experiments and tests conducted in this study have demonstrated the effectiveness of the proposed distributed enclosing and formation control algorithms. The performance in static formation tests showed that the agents could organise themselves into predefined shapes, such as triangular sub-formations within groups that aligned linearly with a central target.

Additionally, the system exhibited excellent scalability and robustness (even though `lab102` limits). The control algorithms and communication protocols (RPC and SSH) proved efficient in managing larger swarms without significant performance degradation. Tests involving varying numbers of robotic agents and challenging environmental conditions, such as communication delays and agent failures, confirmed the system's resilience. Despite these disruptions, the system maintained overall formation and coordination with minimal impact on performance.

These results underline the potential of the presented approach for real-world applications in various domains, including search and rescue, environmental monitoring, and industrial automation.

References

[1] Jesús Alastruey Benedé, Natalia Ayuso Escuer, and José Antonio Gutiérrez Elipe. *Trabajo remoto en los equipos de los laboratorios L1.02 y L0.04*. `http://webdiis.unizar.es/~chus/docencia/trabajo_remoto/trabajo_remoto.html`. Accessed: 2024-05-29. Oct. 2020.

[2] Wikipedia contributors. *Remote procedure call — Wikipedia, The Free Encyclopedia*. `https://en.wikipedia.org/wiki/Remote_procedure_call`. Accessed: 2024-05-29. 2024.

[3] Wikipedia contributors. *Secure Shell — Wikipedia, The Free Encyclopedia*. `https://en.wikipedia.org/wiki/Secure_Shell`. Accessed: 2024-05-29. 2024.

[4] *Course Slides*. UNIZAR. URL: `https://moodle.unizar.es/add/pluginfile.php/10950706/course/section/998931/CIM%20-%203.%20Coded%20Photography.pdff`. (accessed: 21.03.2024).

[5] GeeksforGeeks. *Lamport's Logical Clock*. `https://www.geeksforgeeks.org/lamports-logical-clock/`. Accessed: 2024-05-29. 2023.

[6] Hunter Heidenreich. *Kabsch Algorithm: NumPy, PyTorch, TensorFlow, and JAX*. 2023. URL: `https://hunterheidenreich.com/posts/kabsch_algorithm/` (visited on 05/29/2024).