Research Grant Report – Warehouse Project

Carlos Soria Elizalde

November 2024 – April 2025

1. Introduction

This document relates to the University Research Grant which I held from November 2024 to April 2025 in the Statistics, Computer Science and Mathematics Department. It was supervised by Dr. José Enrique Armendáriz-Íñigo, professor who belongs to the department. The aim of the scholarship was to do research about Swarm Robotics, so that its findings could contribute to the subject "Concurrent and Distributed Programming" which he teaches in this institution. In this period, I have carried out research 9 hours per week both at the department's workplace and from home when the circumstances prevented me from travelling to the university.

2. What is Swarm Robotics?

Swarm Robotics is an emerging field of science that studies systems composed of autonomous units whose individual interactions and performances lead to a collective outcome that would be unattainable by a single individual alone [1][2]. It draws inspiration from natural systems such as insect colonies, where simple agents following local rules achieve complex, robust, and adaptive group behaviors [3]. The main characteristics of Swarm Robotics are:

- Independence: The units operate autonomously without requiring centralized control, although they may interact with each other to accomplish shared goals [2][4].
- Robustness: The system can continue functioning even if some robots fail, as collective performance is distributed across many agents [1][3].
- Scalability: The number of units can increase without compromising the system's stability or efficiency, making swarm systems suitable for large-scale tasks [1][2].
- Equality: No single robot acts as a "master." All robots are either identically programmed to perform the same task or grouped into subpopulations with distinct roles but equal status in the system [1][4].
- Locality: Each robot operates based on locally available information and local interactions, rather than global awareness of the system [2][4].
- Adaptability: The swarm adjusts to environmental or internal changes without needing a central mechanism to coordinate the adaptation [1][3].
- Bio-inspired or heuristic algorithms: Many swarm robotic systems rely on bio-inspired, heuristic, or metaheuristic algorithms—such as Ant Colony Optimization (ACO) or Particle Swarm Optimization (PSO)—that emulate natural collective intelligence to achieve coordination and problem solving [3][5].

Swarm robotic systems are highly suitable for dynamic environments and tasks that demand flexibility, redundancy, and scalability—such as search and rescue, environmental monitoring, and distributed sensing—because of their decentralized control and emergent robustness [6][10]. Beyond theory, there are already practical demonstrations moving toward real-world deployment. For instance, NASA has pursued multi-robot concepts for planetary exploration, including foldable scouting micro-rovers designed to traverse and survey difficult terrain on the Moon (and potentially Mars) [8], as well as the SWIM concept—a swarm of cellphone-size micro-swimmers intended to explore subsurface oceans on icy moons like Europa and Enceladus [7]. Likewise, researchers at ETH Zurich have shown that autonomous flying robots can assemble load-bearing structures, exemplified by a 7.4-meter rope bridge built without human intervention [9]. The application space is broad, and recent surveys argue that swarm robotics is poised to contribute significantly across sectors—e.g., disaster response, environmental management, agriculture, and space systems—thanks to its versatility and scalability [6].

3. Aim of my Research Scholarship

My work has focused on developing two applications related to Swarm Robotics:

The first project consists of a swarm of robots that must carry efficiently food from its source (which is in the beginning unknown to every robot) to their basement (known by every robot since the beginning). Therefore, robots must first find the source of food and then start to carry food from the latter to its basement; wise robots (those who are aware of the source of food at a specific moment of the simulation) can tell naïve robots a direction to follow as a clue to find the source of food (this simulates circumstances that prevent wise robots to provide complete information). The simulation stops when every robot is aware of the source of food's location. This project is not intended to be put into practice in real life, but to serve as an introduction into the Swarm Robotics' world, with its difficulties and challenges. Nevertheless, if adapted, it could help to solve certain land reconnaissance and transportation of goods problems.

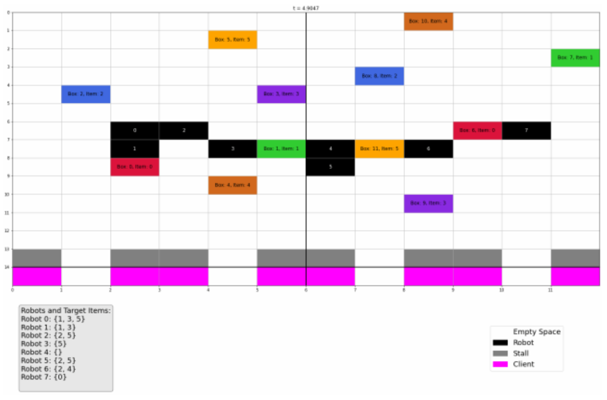
The second project (which is explained in this report) is related to a real problem: a store in which robots must attend the customers' commands. Customers may ask for a certain combination of items, which are in specific boxes in the back part of the store, and to do so, they must calculate the optimal route for getting all the elements of the client's command, trying to optimize the distance traveled. Using previous information, robots calculate the fastest route with less operations by considering only a part of all possible routes. There is an extra difficulty with the implementation, since in the beginning the store is divided in two unconnected sides, each of them with all the possible items, so that in a certain time, the two sides merge and become the original store, and robots that were originally in one side may go to boxes that belong to the other side. Furthermore, from time to time a metaheuristic algorithm (genetic algorithm - GA) is executed to improve the distribution of the boxes, with the aim of minimizing the total distance travelled by robots.

4. Simpy

SimPy is the main library that I have used to implement successfully both programs. It consists of several functions related to the simulation of events. Thanks to it, I could first create an environment in which robots belonging to the swarm can act. Second, it can also provide traffic lights which I have used to coordinate the update of the positions of the robots when they move (so that the update happens linearly, and the sequence of movements is consistent). Furthermore, this library also has a function that makes the robots wait for a certain period of time (so that no robot gets always the traffic light resource and does not give the other robots the chance to update their respective positions).

5. Second Application: Store managed by robots

In this application, robots must manage a store by themselves. Therefore, they are endeavored with tasks that management implies: taking clients' commands, finding the required items in the most suitable order and consuming the least number of resources possible, and keeping record of their services and commands' number of steps to be able to extract valuable information. As robots may come across each other in this problem, they can get advantage and share information so that it is eventually spread among the robots. There are two boxes per item (for example, ice cream flavors and buckets; there are two buckets per ice cream flavor), one at the left half of the store, and the other one at the right half. Initially, there is a screen which separates both parts of the shop. Consequently, robots at a certain side can only get the required items from the boxes that belong to that half. At a certain moment, it disappears, so robots can get the items from whichever part of the shop (in the most convenient way). In order to improve the functioning of the warehouse by minimizing the distance travelled by robots, a genetic algorithm is executed from time to time, using the information that robots know. Moreover, there are different parameters which users can modify to change the complexity of the problem: the number of "items", the number of robots that attend clients, the moment of the simulation in which the screen/wall disappears, the parameter of the arrival rate of clients that follows a Poisson Process (λ), the maximum number of items per command and the time in which the simulation stops. I also use MatPlotLib to create graphics in which the situation at every moment of the simulation is shown and described (at the end of the simulation, all images are put together in a GIF file).



Robots and Target Items:
Robot 0: {1, 3, 5}
Robot 1: {1, 3}
Robot 2: {2, 5}
Robot 3: {5}
Robot 4: {}
Robot 5: {2, 5}
Robot 6: {2, 4}
Robot 7: {0}

Figure 1: Graph showing the warehouse situation at a certain moment during the simulation

6. Implementation

6.1 Store Class

First, we define the Store class. It contains the most important elements and the space in which robots move, including those shared between the robots (boxes, items…). To determine the space needed to host a certain number of robots and boxes, we follow this reasoning:

- The positions of the boxes cannot touch each other horizontal, vertical or diagonally: the distance must be greater than $\sqrt{2}$ squares. In the worst case, the distance between two boxes will be 2 squares (in the same row or column, with one spare space between them).
- A box takes up a (3×3) space if they cannot touch each other even diagonally. However, it is true that 2 boxes can occupy a (3×5) space instead of (3×6) if they are set in the distribution mentioned before. In order to think about the size of the warehouse that will be able to hold all boxes, I must think as in the first case.
- So, if we have n boxes, we must have at least $n \times (3 \times 3)$ squares of storage space = $9n$ squares.
- Also, if we have R robots and stalls, if we want each robot (which occupies (1×1) squares of space) to have its own stall, then the length (number of columns) of the warehouse must be $max(R, \lceil\sqrt{9n}\rceil)$. If the result were odd, we would add one because we want the store to be symmetric.
- For the height (number of rows), we will leave 3 spaces for the customer service area (no boxes on the way), one row for the stalls, and one extra line for the customers. So, the height of the store is $3 + 1 + 1 + \lceil\sqrt{9n}\rceil = 5 + \lceil\sqrt{9n}\rceil$

Then, I put randomly each box on each side of the store, not allowing boxes to be placed in the neighboring positions. After, I create two dictionaries that contain information about the label of the box and the item it contains. The following step is to place the robots' stalls (where they attend clients' commands) in the most even way. I create a function that does this. Then, I create two dictionaries in which I keep the positions of the stall and related client for each robot. I create then different elements of the simulation such as the global waiting line, a list that will contain the graphics created by the robots (to create the visual simulation at the end), a palette for the colors… Other important functions that I create are one for initializing every robot or another one that helps me assign efficiently colors to the boxes (to be able to differentiate them in the graphic)

A key component of the project is how we set the frequency of each possible command (itemset). The user specifies the desired frequency distribution by itemset size and (optionally) the per-item probabilities; if the latter are not provided, they are sampled and normalized. For any itemset S, its base probability is computed as the product of the individual item probabilities, $P(S) = \prod_{i \in S} P(i)$. Finally, we renormalize across itemsets within each size to match the user-provided frequency for that size, ensuring that single-item commands do not dominate higher-order commands.

6.2 Robot Class

This class is the most important one as it defines how the robot works.

After calling the "initialize_robot" function from the Store class, we define the position of its stall and client, its dictionaries to keep track of the movements and orders that all the robots undertake (for each possible command, there is a list initialized with as many zeros as existing robots), a dictionary that contains the optimal order of boxes to visit for each command (and a list that does something similar but it is only used to achieve an efficient merging), and certain parameters (for example, a list that indicates from which robot our robot has its updated information; this helps to avoid unnecessary sharing of information between robots). I also define a graphic function that creates each graphic that will be part of the final simulation GIF.

Later, I create a function that calculates the shortest path between two places using Dijkstra (excluding blocked cells due to robots, stalls and boxes). This function is used inside another function called "calculate_total_distance" which calculates the distance to get all the items of a particular command. Again, this last function is referenced in a "find_optimal_order" function, which receives all the possible order of items to pick and chooses the one with the shortest distance associated.

To accelerate the process of determining the order of boxes to visit (avoid all possible combinations), the robot will use information it will have been collecting. As it has the best order for each possible command, if it has a new command that has part of the elements of an already done one, it will take the repeated elements in the same order (it will consider the route with the highest number of repeated elements). Starting from that partial optimal route, there's another function that creates all the possible "wise" combinations with the new elements (those which do not appear in the old route). The only condition of these new combinations is that the elements of the old order appear in that very order, but there can be new boxes between old elements.

Moreover, robots can share information if they come across each other. For the number of times each robot has attended a specific command, they update themselves with the highest value available.

- If Robot C has for command {1}: [0,1,2], and Robot B has {1}: [3,2,1], they would have after meeting each other {1}: [3,2,2]

For the number of movements (they keep the sum of movements of the different commands), they also use the maximum

- If Robot C has for command {1}: [0,7,17], and Robot B has {1}: [38,16,10], they would have after meeting each other {1}: [38,16,17]

For the optimal routes, they share to each other the optimal route they have for each command if one of them lacks that information. They also update the information they have already up to date about themselves. Finally, the general dynamics of how robots can meet is defined.

After, I define the functions related to the GA that from time to time they execute to minimize the travelled distance. The last special function created is one that helps me to consider all possible combinations of boxes to visit for a special command (quite relevant when robots can visit whichever of the 2 boxes that contain a required item).

6.2.1 Run Function

The "run" function determines how the robot works. It waits in its stall until a client arrives and wakes it up. It tries to get the traffic light and when achieved, it calculates the optimal route of the boxes to pass by thinking in an efficient way (avoiding use Dijkstra if possible) by considering previous commands or information given by other individuals (with the functions described before). Then, for each box it must visit, calculates the shortest path from where it is to the box (although it uses Dijkstra, the calculations are much shorter). It tries to take a step. If it can, it takes it and sees whether there are any neighbor robots to share information. If not, it calculates again a path considering the blocked cell. When it gets all the items, it calculates the shortest path to its stall, following the same mechanics as before. When it arrives, updates all its information, makes the client leave, and may ask for executing the GA. If so, it remains in its stall, waiting for every robot to arrive to its respective stall. At that very moment, it executes the algorithm.

6.2.2 GA

In this section, I will explain how the GA has been implemented.

First, each individual represents a possible spatial configuration of the warehouse, encoded as a permutation of the available storage positions (or "boxes"). This permutation determines the relative location of each item with respect to the service center or stall, and therefore implicitly defines the travel distances required to pick individual items or combinations of them. In subproblems where the warehouse is divided into two areas (east and west), individuals correspond to permutations of the boxes within the corresponding section.

The algorithm follows the classical structure of a generational genetic algorithm. A population of candidate layouts evolves through successive iterations (generations), guided by a fitness function that quantifies the operational efficiency of each configuration. At each generation:

- Selection identifies the most promising layouts using a tournament strategy.
- Crossover (Partially Matched Crossover, PMX) combines two parent layouts to create new offspring while maintaining valid permutations.
- Mutation introduces small random changes by shuffling a few elements, preserving diversity and avoiding premature convergence.
- The best individual found so far is preserved by elitism (Hall of Fame mechanism).

The process continues until a fixed number of generations is reached, yielding as final output the layout that minimizes the objective function described below.

The goal of the optimization is to minimize the expected travel distance required to fulfill customer commands, given the frequency with which each product or combination of products is requested. Let:

- N: total number of observed commands,
- p_i: probability that item i appears in a command,
- p_{ij}: joint probability that both items i and j appear together in the same command,
- $d_i(\pi)$: minimum round-trip distance from the service center to the box assigned to item i under layout π,
- $d_{ij}(\pi)$: minimum closed route distance to collect both items i and j starting and ending at the service center under layout π.

These probabilities are derived empirically from the historical frequency of commands, while distances depend on the geometric arrangement defined by each individual (layout permutation).

For a given layout π:

- Each item may be associated with one or two storage positions (depending on the time that the GA is executed). The distance $d_i(\pi)$ is the minimum round-trip distance from the service center to either of its positions.
- For pairs of items (i, j), all feasible routes connecting the center, both items, and back to the center are evaluated, so that:

$$d_{ij}(\pi) = \min_{\alpha,\beta \in \{a,b\}} [\mathrm{dist}(C, \mathrm{box}_\alpha(i)) + \mathrm{dist}(\mathrm{box}_\alpha(i), \mathrm{box}_\beta(j)) + \mathrm{dist}(\mathrm{box}_\beta(j), C)].$$

The shortest of these paths is used as the effective distance for the pair. The objective function to be minimized is therefore:

$$F(\pi) = \sum_i p_i \, d_i(\pi) \;\; + \;\; \beta \sum_{i<j} p_{ij} \, d_{ij}(\pi),$$

where the first term captures the expected cost of retrieving individual items and the second term accounts for the additional travel generated by items that are frequently ordered together. This formulation reflects the practical objective of minimizing total walking (or picking) distance given empirical command frequencies.

The reason of the scaling factor β is the following: there are $k - 1$ transactions between items in a command containing k items, although there are $\binom{k}{2}$ possible pairs of elements (for example, if you have four items “a”, “b”, “c”, and “d” in a command, the robot may do “a”-“b”, “b”-“c”, “c”-“d” to get them: 3 transactions). In the second term of the objective function described above, we are adding pair costs (considering every possible pair in the command), therefore, the cost of moving between items is being overrepresented.

Given a certain size of commands k, as there are $k-1$ real transactions between items and $\binom{k}{2}$ possible transactions, the scaling factor would be

$$\beta_k = \frac{k-1}{\binom{k}{2}} = \frac{2}{k}$$

The following step is to work out the expression for β considering the average number of items per command. This value is denoted by

$$\bar{k} = \sum_i p_i.$$

Consequently, the scaling factor would be

$$\beta = \frac{E[k-1]}{E\left[\binom{k}{2}\right]} = \frac{\bar{k}-1}{\sum_{i<j} p_{ij}(\pi)}$$

To reduce the computational effort, if we assume that $E\left[\binom{\mathrm{k}}{2}\right] \approx \frac{\bar{k}(\bar{k}-1)}{2}$, the final scaling factor would be

$$\beta \approx \frac{2}{\bar{k}}$$

To summarize, this optimization process encourages frequently requested products to be placed closer to the service center, reducing average access time. At the same time, products that are often ordered together tend to be positioned near each other, minimizing the marginal cost of picking both in a single route. The weighting factor βensures that both single-item and multi-item efficiency are balanced according to the typical order size. As a result, the final layout approximates a statistically optimal configuration that minimizes the expected total picking effort for future operations.

7. Client Generator Class

The last class created is one for adding clients with specific command to the global or split queues (depending on whether the screen that divides the store has vanished or not).

8. Next steps

The project will conclude with the implementation of two additional metaheuristic algorithms—Particle Swarm Optimization and the Gravitational Search Algorithm—to compare their efficiency and performance, along with the necessary tests to evaluate the impact of these methods. Moreover, since box changes are currently applied automatically, the project would gain significant value if the robot that executed the GA also computed the required adjustments to achieve the new box distribution and explicitly instructed its peers on the specific changes they need to perform.

9. Conclusion

Swarm Robotics is a promising area whose development in the following years will certainly bring diverse positive daily life applications and benefits. Thanks to this collaboration scholarship, I have gained valuable knowledge and helped to develop algorithms that may be implemented in academic or certain real-life environments. Therefore, I would like to thank again the Public University of Navarre and its Statistics, Computer Science and Mathematics Department for offering me this opportunity, and Dr. José Enrique Armendáriz-Íñigo for guiding me along these last months.

Carlos Soria Elizalde, April 30th 2025.

References

[1] E. Şahin, Sertan Girgin, Levent Bayindir and Ali Emre Turgut “Swarm Robotics: From Sources of Inspiration to Domains of Application,” in Swarm Robotics, Lecture Notes in Computer Science, vol. 3342, Springer, 2005, pp. 10–20.
[2] M. Brambilla, E. Ferrante, M. Birattari, and M. Dorigo, “Swarm robotics: A review from the swarm engineering perspective,” Swarm Intelligence, vol. 7, no. 1, pp. 1–41, 2013.
[3] E. Bonabeau, M. Dorigo, and G. Theraulaz, Swarm Intelligence: From Natural to Artificial Systems, Oxford University Press, 1999.
[4] M. Dorigo, M. Birattari, and M. Brambilla, “Swarm robotics,” Scholarpedia, vol. 9, no. 1, pp. 1463, 2014.
[5] H. Hamann, Swarm Robotics: A Formal Approach, Springer, 2018.

[6] L. V. Nguyen, “Swarm Intelligence-Based Multi-Robotics: Applications and Challenges,” Digital, vol. 4, no. 4, 2024.

[7] NASA/JPL, “Swarm of Tiny Swimming Robots Could Look for Life on Distant Worlds (SWIM),” News release, 28 Jun. 2022.

[8] NASA/JPL, “A-PUFFER (Autonomous Pop-Up Flat Folding Explorer Robot),” project page.

[9] ETH Zürich, Flying Machine Arena, “Building a rope bridge with flying machines,” 18 Sept. 2015.

[10] M. Bakhshipour and A. Hashemi, “Swarm robotics search & rescue: A novel artificial intelligence-based approach,” Applied Soft Computing, vol. 57, pp. 708–726, 2017.