Using swarm algorithms to explore unknown areas in ROS2

Oleksandr Bezsonov[1], Sofiia Rutska[1], Oleg Rudenko[1], Stanislav Piskunov[2]

[1] *Kharkiv National University of Radio Electronics, Nauky Ave. 14, Kharkiv, 61166, Ukraine*
[2] *Ivan Kozhedub National Air Force University, Sumska str., 77-79, Kharkiv, 61023, Ukraine*

Abstract

This paper presents an investigation into using swarm algorithms to automate search, mapping, and localization tasks in multi-robot systems. The study focuses on developing effective coordination strategies for multiple robots operating within a shared network, aiming to explore unknown environments autonomously. The main objective of this research is to optimize route planning and minimize exploration time while enhancing system robustness through decentralized control and communication between agents. In addition, the work demonstrates the potential of swarm intelligence in improving the efficiency of collective decision-making processes. The proposed approach leverages the bee algorithm, a bio-inspired optimization method, to enable autonomous robots to explore, map, and localize within dynamic environments cooperatively. This study highlights the application of such systems in real-world scenarios, such as search and rescue missions, reconnaissance, and industrial automation, emphasizing their potential to address complex, large-scale tasks with improved scalability and adaptability.

Keywords
multi-agent bee algorithm, swarm intelligence, robot coordination, mapping, localization, decentralized control, autonomous exploration

1. Introduction

In today's world, robotics is one of the leading technological advancements, with robots increasingly being integrated into various aspects of everyday life, making the world more efficient and convenient. However, in most cases, robotic applications focus on individual units performing specific tasks, while the challenge of coordinating multiple robots as a unified system remains complex. When working with groups of autonomous agents, issues such as coordination, communication, and task distribution arise, requiring advanced strategies to ensure efficiency and effectiveness.

One promising approach to solving these challenges is swarm intelligence, a field inspired by the collective behavior of biological systems such as ants, bees, and flocks of birds. Swarm robotics leverages decentralized control, local interactions, and simple rules to enable multiple robots to work collaboratively, achieving tasks that would be difficult or impossible for a single unit. This methodology is beneficial for autonomous exploration, where a team of robots must navigate and map unknown terrain, find optimal routes, and adapt to dynamic environments.

This paper presents a system that utilizes swarm intelligence for autonomous exploration, localization, and mapping. The relevance of this research lies in optimizing robotic control strategies for drones and ground robots in unknown environments. Such a system could explore terrain efficiently and perform specialized search operations when equipped with cameras or sensors. A swarm of robots could be deployed for tasks such as locating missing persons, detecting gas leaks, or identifying hazardous substances, making them valuable in disaster response, environmental monitoring, and industrial applications. Moreover, advancements in artificial intelligence and edge computing have significantly enhanced the capabilities of swarm robotic systems, enabling real-time decision-making and adaptive behaviors in unpredictable environments. Swarm robotics is crucial in large-scale automation, from smart cities to space exploration.

[1]CMIS-2025: Eighth International Workshop on Computer Modeling and Intelligent Systems, May 5, 2025, Zaporizhzhia, Ukraine

oleksandr.bezsonov@nure.ua (O. Bezsonov); sofiia.rutska@nure.ua (S. Rutska); oleh.rudenko@nure.ua (O. Rudenko); piskunoff@ukr.net (S.Piskunov)
0000-0001-6104-4275 (O. Bezsonov); 0009-0009-0575-8172 (S. Rutska); 0000-0003-0859-2015 (O. Rudenko); 0000-0002-4685-527X (S.Piskunov);

2. Related works

Swarm robots have some critical characteristics that differentiate them from other types of platforms, including but not limited to simplicity, size, scalability, cooperative ability, and communication capabilities [1]. In particular, two fundamental issues are their size and cost, as these aspects significantly affect the scalability of real swarm systems.

A solution has been proposed for operating a multi-robot system using a proprietary ROS-based architecture, demonstrating efficient real-time performance with multiple robots [2]. This solution relies on the first generation of ROS, which is now considered outdated due to the numerous enhancements introduced in ROS2. Modern ROS2-based systems offer improved facilities for distributed communication, real-time, and modularity, greatly enhancing the possibilities of building swarm systems [3].

More recently, there has been extensive research on the customization and deployment of the ROS navigation stack [4], which emphasizes the continued development of the ROS2 ecosystem and its applicability to autonomous mobile navigation tasks. However, such works mainly focus on single robots rather than their coordination within a swarm.

At the same time, there is a growing interest in applying swarm algorithms in mobile robotic systems, especially in robot-to-robot communication and decentralized control. Some developments, such as HeRo 2.0 [5] and HeRoSwarm [6], focus on building specialized hardware platforms for swarm systems. Despite interesting hardware solutions, these projects do not provide compatibility with ROS2, which limits their flexibility and repeatability of results in a scientific environment.

An alternative approach is offered by the ROS2swarm library [7], which is one of the first attempts to implement swarm behaviors in the ROS2 environment. It provides basic agent interaction patterns and simulation support but stays within a limited set of scenarios and does not fully evaluate the scalability or performance of the chosen algorithms.

The advantages of the proposed approach over existing solutions are a balanced ratio between low cost and performance, simplicity of the implemented swarm algorithm, and high efficiency of coverage of the studied territory. The developed system provides flexibility in setting parameters. It allows for a comprehensive analysis of the collective behavior of agents in various conditions - both in a homogeneous and a heterogeneous environment.

3. Methods and Materials

Robots and various approaches to territory exploration, including swarm algorithms, were used in the study. Simulation environments were employed to improve modeling and visualization accuracy, allowing for detailed analysis of robot behavior in virtual space. Additionally, tools facilitated navigation, mapping, and autonomous movement, enabling comprehensive testing and refinement of algorithms before real-world implementation.

The project used the following tools: RViz is a 3D robot visualization tool in ROS 2 that allows displaying sensor data, maps, robot trajectories, and other parameters in real-time. It supports extensions through plugins, and the librviz library allows embedding visualization capabilities into applications. Gazebo is a physical simulation environment designed to test robots in virtual conditions without the need for their actual operation. Nav2 is the ROS 2 navigation stack responsible for mapping, localization, route planning, and motion control in autonomous robotic applications. ROS 2 is a robot operating system that provides the necessary tools and libraries for developing, simulating, and controlling robots in various applications [8].

The robot was a four-wheeled structure with two axles connecting pairs of wheels. It was equipped with a sensor (LiDAR) to scan its surroundings, a camera for visual perception, and an odometry and mapping system for precise positioning and localization. Odometry was used to estimate the robot's movement, using sensors to measure the distance traveled and movement speed.

The SLAM (Simultaneous Localization and Mapping) algorithm allows robots to simultaneously build a map of the environment and determine their location without prior known information. The SLAM algorithm used odometry and scanner data for the project to locate and construct the map.

4. Swarm algorithms

The project is based on the use of swarm algorithms, and there are several conditions under which the best option was chosen:

- The algorithm should be oriented towards exploring the space, not just finding the optimal solution.
- Work in real-time, i.e., it should not require prior knowledge of the map.
- Distributes robots efficiently to avoid crowding them in one place.
- Takes into account obstacles and the changing environment.

The challenge may be how this map can then be merged, so a single starting area has been adopted for all robots, which will then explore the terrain from this point.

4.1. The particle swarm optimization algorithm

The particle swarm optimization (PSO) algorithm models the social-psychological behavior of a crowd. PSO is an optimization algorithm capable of solving nonlinear and multidimensional problems, typically achieving reasonable solutions quickly while requiring minimal parameterization.

The algorithm and its concept, Particle Swarm Optimization (PSO), were introduced by James Kennedy and Russell Eberhart in 1995 [9]. The central concept of the algorithm is the creation of a swarm of particles that move through their surrounding space (problem space) in search of their target or the location that best meets their needs, as defined by the fitness function [10]. In this algorithm, the swarm is fully connected; all particles exchange information, and each particle knows the best position ever visited in the swarm.

This approach focuses on presenting the best result for a particular robot and the swarm. Such an algorithm converges quickly to an optimal solution and works well if the goal is known. However, this approach is inefficient in a completely unknown environment, as there is no explicit 'best solution' for navigation. Robots may concentrate in one place instead of exploring uniformly. For these reasons, PSO is more suitable for optimization problems but is poorly suited for exploring new territory.

4.2. Ant algorithm

Ant Colony Optimization (ACO) is a widely recognized algorithm inspired by how ants forage for food and is frequently applied to tackle combinatorial optimization problems. Initially introduced by Dorigo, Maniezzo, and Colorni [11], this nature-inspired metaheuristic simulates the ability of ants to discover optimal paths between their nest and food sources, successfully addressing shortest-path problems without prior knowledge of the problem's structure [12]:

$$p_{i,j} = \frac{\left(\tau_{i,j}^{\alpha}\right)\left(\eta_{i,j}^{\beta}\right)}{\sum_{k \in N} \left(\tau_{i,k}^{\alpha}\right)\left(\eta_{i,k}^{\beta}\right)}, \tag{1}$$

where $\tau_{i,j}$ — the amount of pheromone on the edge (i, j); α — a parameter controlling the influence of the pheromone; $\eta_{i,j}$ — the attractiveness of the edge (i, j), usually equal to $\frac{1}{d_{i,j}}$, where $d_{i,j}$ is the distance between nodes; β — a parameter controlling the influence of attractiveness.

Continuous Ant Colony Optimization (CACO) is an improved version of classical ACO adapted for problems in continuous space rather than discrete graphs [13]. Unlike ACO, where ants move between predefined nodes, in CACO, the robots move freely along coordinates and update a 'virtual pheromone' in a region of space [14]. This algorithm allows more efficient exploration of unknown terrain, primarily when the map structure is unknown in advance. This makes CACO suitable for the task of collective map construction by robots: it provides flexibility of movement, allows dynamic adaptation to changing environments, and does not require a predefined grid of routes.

4.3. Bee algorithm

The Bee Algorithm is a metaheuristic optimization algorithm inspired by the foraging behavior of real bees. It simulates the work of scouts searching for new sources of nectar and worker bees intensifying their search in the most promising areas.

In the bee algorithm, agents (bees) must find optimal solutions, similar to how honeybees find food sources. Bees share information about the location of food sources with other colony members using a waggle dance. This dance, which consists of alternating turns and waggles of the abdomen, conveys two important parameters: the distance to the food source and its direction [15].

The duration of the waggle indicates the distance, and the angle between the sun and the movement of the abdomen on the comb indicates the direction. In the algorithm, other agents can apply these instructions and follow the best solution, similar to how bees navigate to the best food sources. The decision-making mechanism depends on the quality of the solution, which allows agents to choose more promising options.

The algorithm uses Path Integration (PI), a method in which each agent updates its position, considering all the distances traveled and changes in direction. This allows agents to efficiently explore the space and find optimal solutions, similar to the behavior of insects using this navigation method.

Multi-agent systems inspired by bee behavior uniquely adapt to environmental changes and efficiently solve resource allocation problems. Such systems are based on the principles of self-regulation, where each agent acts based on local information and interacts with other agents to achieve a common goal. Importantly, these systems can be used to develop efficient algorithms and better understand the mechanisms of collective decision-making in nature [16].

However, this approach may have some drawbacks, such as a tendency to get stuck in local minima, which may make it difficult to effectively explore large or complex areas, as robots may be limited to exploring only nearby areas rather than covering all possible areas. Another obstacle is the need for multiple robots to provide sufficient coverage, which may increase computational costs, especially if the number of swarm members or the size of the area to be explored is large. In addition, the bee algorithm may be sensitive to the initial placement of agents, which may lead to uneven coverage or insufficient exploration of the area in the context of dynamic search.

To address the challenges of local exploration and resource allocation, we utilize the multi-agent bee algorithm, a probabilistic approach to determine which areas should be prioritized by the agents. This is achieved by calculating the probability \($P(x)$ \) of selecting a specific point \(x \) based on its attractiveness, which is defined by the following formula:

$$P(x) = \frac{f(x)}{\sum_{y \in R} f(y)}, \tag{2}$$

where x — is a potential research point; R — is the set of all possible research points; $f(x)$ — is the attractiveness function of the point x, which can be defined as:

$$f(x) = \alpha \cdot d(x, B) + \beta \cdot g(x) + \gamma \cdot h(x), \tag{3}$$

where $d(x, B)$ — distance from point x to the nearest known boundary point; $g(x)$ — function estimating the potential of point x to become a boundary point; $h(x)$ — function taking into account the history of the study of point x; α, β, γ — weighting coefficients.

The following formula describes how a robot explores a point based on several factors:

$$t_i(t) = argma\,x_{x \in R¿(t)} \left[\omega^1 \cdot d(x, B) - \omega^2 \cdot d(x, s_i) + \omega^3 \cdot q(x) \right], \tag{4}$$

where $E(t)$ — set of points explored by other robots at time t; $d(x, B)$ — distance to the nearest known boundary; $d(x, s_i)$ — distance from the robot to the point; $q(x)$ — quality function of the point (potential to be a boundary); ω_1, ω_2, ω_3 —weight coefficients.

At time t, each robot must choose a point $t_i(t)$ from the set of possible points R, excluding those already explored points represented by $E(t)$. The robot's decision is based on a weighted combination of factors, allowing it to select points that offer the best coverage of the area, given the current conditions and constraints.

Finally, let *I(t)* represent the global information about the boundary points at time *t*. Each robot n_i updates its local information $I_i(t)$ as follows:

$$I_i(t+1)=I_i(t)\cup_j I\,(t)\vee_j n\in C(n_i,r), \tag{5}$$

where $C(n_i, r)$ is the set of robots within communication radius r from robot n_i.

The chosen approach reflects the core decision-making process of the bee algorithm, prioritizing exploration based on a combination of multiple factors. Robots can effectively balance between exploring new areas and reinforcing previously identified boundaries by utilizing a probabilistic approach to select points based on their attractiveness. The weighting coefficients in the formula allow for flexibility in adjusting the impact of different factors, such as the proximity to known boundaries, the distance from the robot, and the potential of a point to become a boundary. This adaptability makes the bee algorithm particularly effective for dynamic environments where exploration and resource allocation must be adjusted in real-time. Furthermore, the formula accounts for the collaborative nature of the swarm, where each robot's local information is updated through communication with nearby agents, ensuring that the exploration process remains efficient and comprehensive. By selecting the most promising points based on these criteria, the algorithm maximizes the overall coverage and effectiveness of the swarm, ensuring a balance between local and global exploration objectives.

5. Robot architecture

5.1. The base structure of the robot

Below is a structure that shows how the different parts of the robot (sensors, steering mechanisms, wheels) are linked through the transformation system in ROS 2. The presence of global (map, odom) and local (base_link, wheel_link) frames allows the robot to determine its position in space and control movement correctly.

In ROS 2, nodes are the fundamental building blocks of the system. A node is an executable that performs a specific task, such as reading sensor data, processing that data, or controlling the robot’s motors. Nodes communicate with each other using topics, services, and actions. They can be run on the same machine or distributed across multiple machines in a network. The flexibility of ROS 2 nodes allows for modular design and efficient control of robot systems [17].

Each robot in the system operates within its namespace, ensuring that multiple robots can function simultaneously without interference. This structure is crucial in swarm robotics, where each unit must manage its own transformations while also exchanging information with others.

Additionally, ROS 2 supports real-time processing, enhancing robotic operations' accuracy and responsiveness. By leveraging the ROS 2 middleware, robots can efficiently synchronize sensor data, execute navigation commands, and dynamically update their internal state based on environmental changes. This approach improves the scalability and robustness of autonomous robotic systems, making them suitable for large-scale deployments in real-world applications.

Figure 1 shows a graphical visualization of the ROS 2 frame tree (TF) created with tf2 and the "view_frames" tool and converted into a block diagram. This diagram shows the hierarchy and relationships between the robot's different coordinate systems (frames) in the simulation. Two conventional groups, "sensors_group" and "steering_group", are responsible for the operation and stability of the sensors and allow the distribution of responsibilities between components.

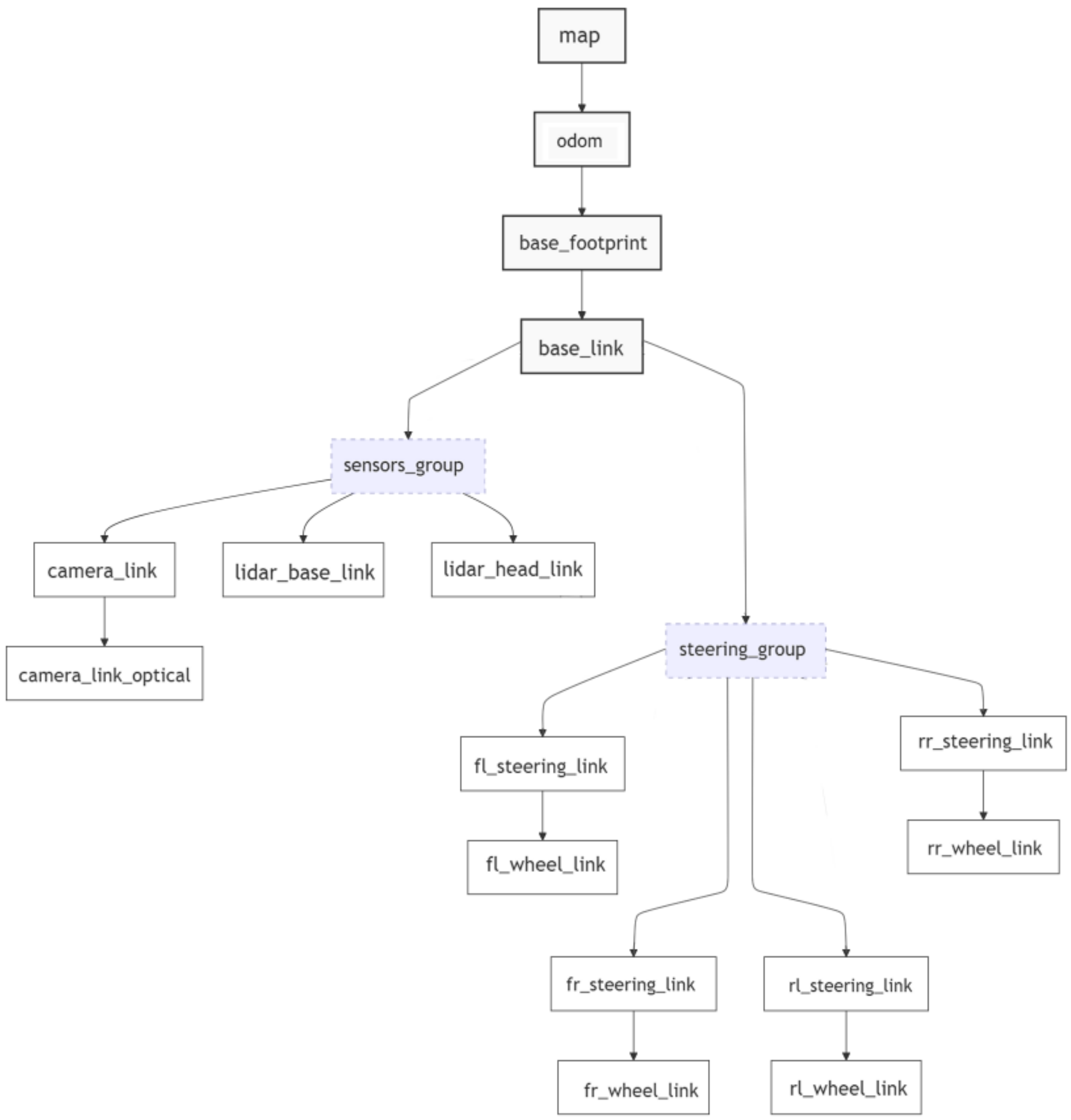

Figure 1: Visualization of robot architecture

The top node of the tree is “map”, the global coordinate system. From it there is a link to “odom” (the odometric system), which is updated over time and contains information about the robot's position relative to the starting point. The structure is then connected to “base_footprint”, the robot's base point representing its position on the surface.

From “base_footprint” follows a connection to “base_link”, which is the robot's main frame to which all other components are linked.

Figure 2 represents the robot that corresponds to the graph above. This image was taken in a gazebo simulation and shows the visual characteristics.

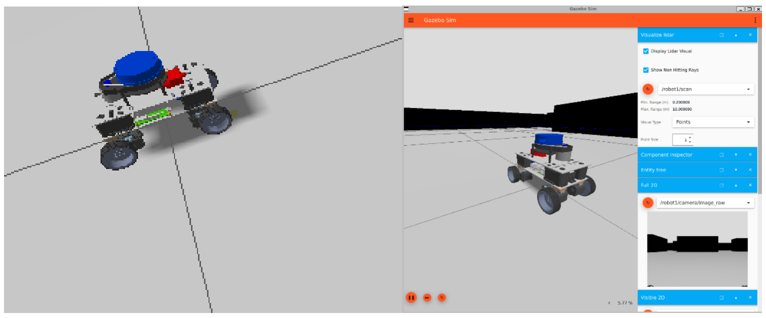

Figure 2: Visualization of robot in Gazebo sim

5.2. Using SLAM in robot architecture

5.2.1. SLAM

The robot building process also considers the future use and availability of the model and system for mapping and localization in a dynamically changing environment. The robot-building process also considers the future use and availability of the model and system for mapping and localization in a dynamically changing environment. With such input parameters, ensuring that these systems are available for scaling and integration into different scenarios is very important. For example, in an urban or manufacturing environment, a robot faces moving objects, changing infrastructure, or unpredictable disturbances. To function in such environments, the system must quickly process data from sensors (lidars, cameras, gyroscopes) and instantly adjust its route, avoiding collisions and minimizing delays. This is especially critical in systems with swarm intelligence, where many robots interact to fulfill a common task – for example, when searching for survivors in an emergency zone or synchronized delivery of goods to a warehouse.

This is where SLAM (Simultaneous Localization and Mapping) comes to the fore. This technology allows each robot to simultaneously build a map of an unknown environment and determine its position in it. Recently, the demand for intelligent robotics has increased, and there are more advanced approaches to detecting the environment for robots [18]. The object can be a domestic robot, autonomous vehicle, planet rover[19], uncrewed aerial vehicle (UAV) [20] [21] or other automated systems. SLAM becomes indispensable in environments with no pre-prepared terrain map or the device's position is unknown, making it a versatile tool for various tasks. Due to the rapid development of robotics, SLAM is attracting increased interest from academia and industrial developers.

SLAM systems can collect environmental data from different types of sensors: laser, acoustic, or visual. For example, robots with cameras analyze images to determine their position and orientation in space. This approach, known as VSLAM (Visual SLAM) [22], has several advantages: reduced hardware costs, simplified object recognition and tracking, and access to detailed visual and semantic data. The resulting images are used for navigation and computer vision tasks such as semantic segmentation or object detection due to the large amount of information they contain.

In the SLAM problem, filters continuously refine estimates of an object's position and velocity using imprecise location measurements. They also improve the accuracy of spatial landmark positions.

The Kalman Filter is an algorithm used to estimate the state of a system under uncertainty [23][24]. It is applied in SLAM, navigation, signal processing, and other fields where state estimation based on noisy measurements is required. In the original implementation of the SLAM algorithm, the primary source of information about the robot's movement was odometry obtained from wheel rotation. Despite its simplicity and widespread use, this method has several significant limitations. Odometry errors accumulate over time due to wheel slippage, interaction with uneven or slippery surfaces, as well as during sudden braking or collisions with obstacles. In addition, wheel odometry does not provide absolute measurements of position and orientation in space, which is especially critical during long-term autonomous operation. In the absence of external correction, for example, due to visual or lidar sensors, the system is prone to drift. For this reason, modern SLAM systems implement a multimodal approach to state assessment, combining odometry data with information from LiDAR, cameras, or inertial measurement units (IMUs). Such sensor integration can significantly improve localization accuracy and the algorithm's resistance to various types of noise and external influences. This study uses a camera and LiDAR as the primary data sources, eliminating the reliance on odometry while improving the system's robustness in simulated conditions.

The RF2O (Range Flow 2D Odometry) method estimates robot odometry by comparing successive laser scans. The basic idea is to find the optimal transformation between two scans that minimizes the point-matching error:

$$\left(R^{¿}, t^{¿}\right) = arg\min_{R,t} \sum_{i=1}^{N} |R \cdot p_i + t - q_i|^2, \tag{6}$$

where p_i — is the point from the previous lidar scan, q_i — is the corresponding point from the current scan, R — is the rotation matrix, t — is the translation vector, N — is the number of matched points.

This approach (6) helps to improve the accuracy and stability of odometry estimation, especially in dynamic and complex environments where traditional methods may have limitations.

5.2.2. Merge maps

In an environment with many robots, the system handles each robot's output separately from the others using the delegation method through namespaces. When each robot publishes a separate map to a separate topic in a namespace, it becomes necessary to merge these maps together. To merge the maps effectively, it is crucial to consider the raw data from each robot and factors like sensor noise, alignment errors, and differences in the local coordinate frames.

One common challenge is the misalignment of the maps, which can occur when the robots use different frames of reference or have slightly different odometry information. Several approaches can address this, including using Iterative Closest Point (ICP) algorithms and techniques like graph-based optimization. Additionally, careful handling of the transform (tf) between robots is required to ensure accurate map alignment. Even with these techniques, some artifacts may persist, particularly in areas where the robots' sensors had conflicting readings. These issues can be minimized by fine-tuning the merging process and employing robust alignment methods, leading to a more accurate global map. In Figure 3, pictures (a) and (b) show the local maps of two separate robots operating in different namespaces, where each robot builds its version of the environment. Figure (c) is a merged map, the result of merging data from both robots, where it can be seen that the map boundaries are aligned. However, there are artifacts and distortions, probably due to errors in coordinate alignment or tf-frames.

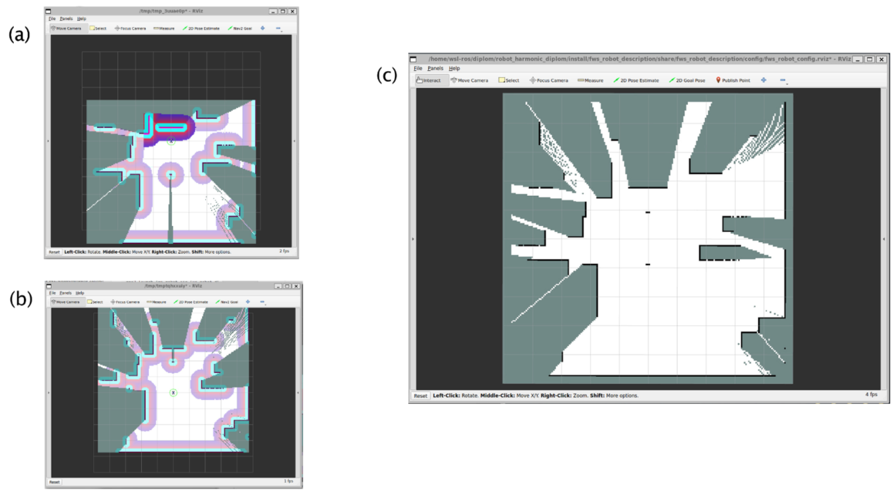

Figure 3: Visualization of merging maps from different robots

5.3. Using Navigation2 for path planning and obstacle avoidance

Navigation2 (Nav2) is a powerful navigation stack for ROS 2, designed for the autonomous movement of mobile robots. It is the successor to the classic ROS Navigation Stack from ROS 1, but offers an improved architecture, advanced customization options, and support for modern navigation algorithms. Nav2 is used for motion planning, path control, obstacle avoidance, and real-time map updates, making it an important tool in robotics for delivery services, autonomous warehouses, and service robots.

Nav2 follows a modular design, each component handling a specific part of the navigation process. Key components include the Planner Server, which builds a global route; the Controller Server for local movement control; the Behavior Tree Engine, which organizes navigation processes; and Costmap 2D, responsible for creating walkability maps. Due to the architecture's flexibility, Nav2 allows easy customization of the planning and traffic control algorithms for specific tasks.

Navigation2 supports integration with various sensors, including lidars, cameras, and IMUs, and works both in simulation (Gazebo, Isaac Sim) and on real robots. The ROS community actively develops the system, and researchers and companies use it in various projects. Support for hybrid AI algorithms, reinforcement learning (RL), and SLAM makes Nav2 a promising solution for dynamic environments.

ROS2 and Navigation2 include the SLAM Toolbox [25] as the core package for SLAM solutions. It provides real-time mapping capabilities for large areas through advanced graph implementations, making it well-suited for dynamic environments. With a focus on customizability, it also offers a variety of options to meet the specific requirements of specific use cases. At the same time, Navigation2 developers have shown great interest in supporting emerging Visual SLAM (VSLAM) approaches that can replace traditional methods that require expensive lidar sensors [26].

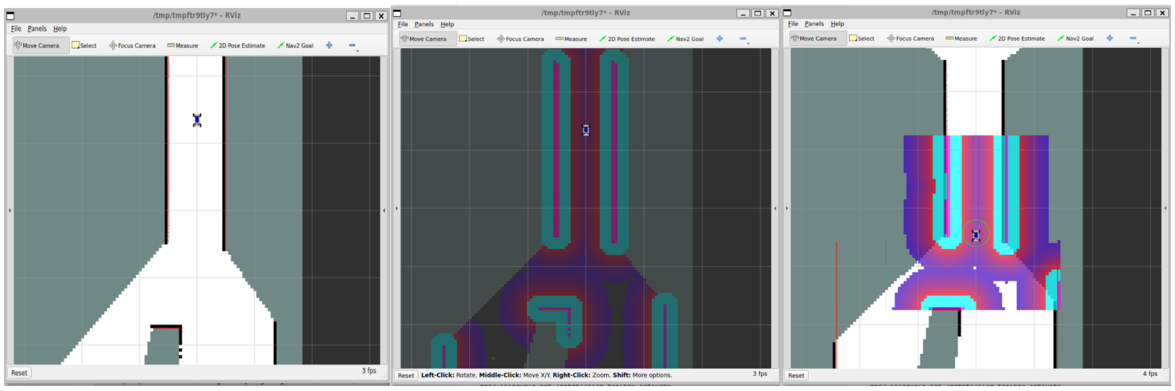
Figure 4: Visualized differences between SLAM map, global costmap and local costmap

6. Experiments

Several experiments were conducted with the robot system on different maps, scaling the number of robots used to build a map of an unknown area. In Experiment 1, a basic pattern was run where a swarm of robots searched for optimal routes to unknown areas using a static environment and tracking other robots with a sensor. In Experiment 2, a combined basic pattern was run and additional sensors and cameras were used to detect other robots in the swarm in a dynamic environment. These two experiments should help us analyze the difference in swarm behavior in static and dynamic environments, and their necessity and difficulty in scaling the swarm.

Three types of maps were used for Experiments 1 and 2: a rectangular box with obstacles inside, a corridor-type area, and an area around a large object. In all three types of maps, the robots had the same amount of time to cover as much territory as possible and return to the starting point.

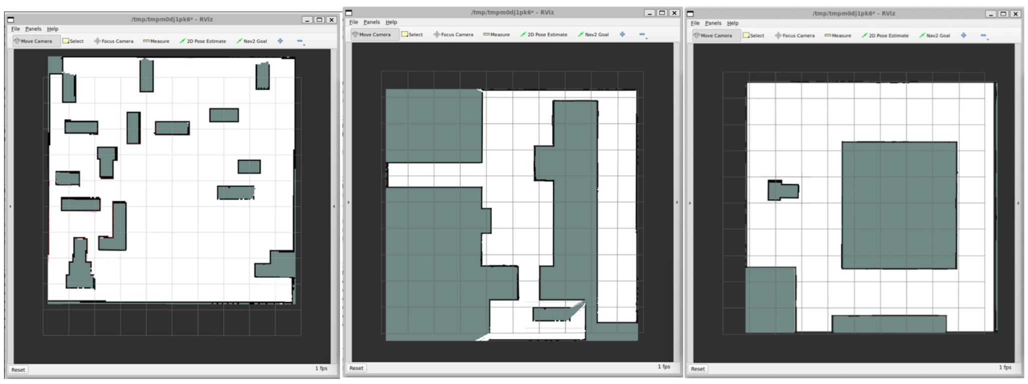
Figure 5: Available maps: a rectangular box with obstacles inside, a corridor-type area, an outer zone

Experiment 1 involves the use of the Multi-Agent Bee Algorithm, which allows scout robots to be used to cover a larger area for exploration.

Since LiDAR does not distinguish between robots and obstacles, robots may mistakenly see walls or other objects as other members of the swarm, which can sometimes alter their assessment of the area and cause one robot that has separated from the group to be unable to return to the swarm without outside intervention.

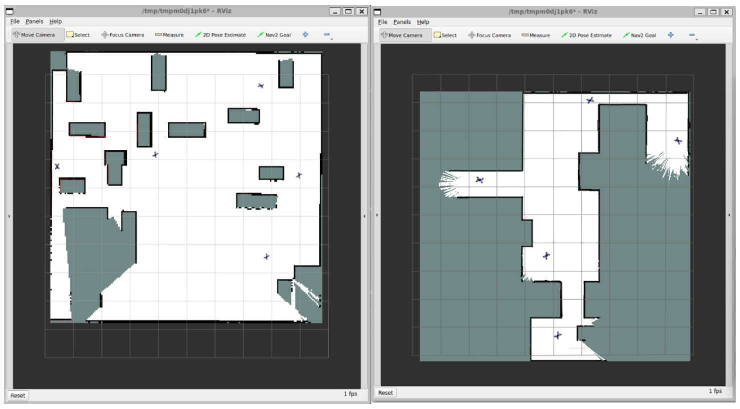

Figure 6: Example of system start-up

Experiment 2 used an algorithm similar to the first experiment but also created a dynamic environment in which the robots had to cover the territory, taking into account the possible emergence of new obstacles. Also, in a dynamic environment, a different number of robots in a swarm was tested, for a clear example of the scalability of the system. This simulation used the same types of maps as the previous experiment. The current setup of the robots differed from the previous one in that new sensors and cameras were used, which allowed them to accurately determine the positions of swarm members in a dynamically changing environment and avoid already marked and explored territories.

Table 1
Experiment results

Experiment	Number of robots	Coverage area, %	Type of map	Average Speed	Robots back in position
1	5	89	rectangular box area	43	5
1	5	72	a corridor-type area	37	4
1	5	85	outer zone	60	5
2	5	91	rectangular box area	46	4
2	5	83	a corridor-type area	36	3
2	5	85	outer zone	59	5
2	10	97	rectangular box area	40	10
2	10	89	a corridor-type area	44	9
2	10	92	outer zone	53	10

The essence of the experiment was that, despite new obstacles, the robots could correctly mark the map and find their way back to the starting position. The difficulty of this method was that the robots require a more complex setup and support, leading to an increase in the complexity and cost of such a swarm. Table 1 shows the results of two experiments, allowing us to analyze the system's stability in different situations and their advantages and disadvantages when scaling in a dynamically changing environment.

Future work will focus on improving the synchronization of robot movements, enhancing sensor fusion techniques, and optimizing the algorithms to handle increasingly dynamic environments with minimal human intervention.

7. Conclusion

The experiments demonstrated that the swarm robot system with basic and combined patterns can effectively explore unknown terrain in both static and dynamic environments. The first experiment showed that using the Multi-Agent Bee Algorithm allows scout robots to expand the exploration area quickly. However, LiDAR sensors' limitations sometimes lead to obstacle recognition errors. This can cause issues with individual robots returning to the formation, especially if they separate from the group and mistake walls for other swarm members.

The second experiment confirmed that adding new sensors and cameras improves the accuracy of robot and obstacle detection in a dynamic environment, which is crucial when the environment changes and new obstacles appear. Despite the increased setup complexity and higher system costs, these improvements enhance the swarm's stability and efficiency, which was clearly demonstrated across three different types of maps.

Comparing the results of the two experiments showed that the dynamic environment reduces the number of robots successfully returning to their starting position, particularly when the swarm size is small. However, increasing the number of robots to ten made the system more resilient, as confirmed by high coverage rates and a greater number of robots returning to the base. This demonstrates the swarm's scalability potential, especially when enhanced sensors and cameras are used.

The experiments confirmed that the swarm system can adapt to various terrains and conditions. The combination of basic and complex patterns, supported by additional sensors, proved effective when scaling the swarm. Nevertheless, the identified LiDAR limitations and the complexity of setting up dynamic systems require further improvements to enhance the swarm's autonomy and resilience in unpredictable environments.

Acknowledgments

The authors would like to thank the scientific and teaching staff of Kharkiv National University of Radio Electronics for their support. All the authors have read and agreed to the published version of this manuscript.

Declaration on Generative AI

During the preparation of this work, the authors used Grammarly in order to: Grammar and spelling check. After using this tool, the authors reviewed and edited the content as needed and take full responsibility for the publication's content.

References

[1] I. Olaronke, I. Rhoda, I. Gambo, O. Oluwaseun, O. Janet, A systematic review of swarm robots, Current Journal of Applied Science and Technology 39 (2020) 79–97. doi: 10.9734/cjast/2020/v39i1530719.
[2] C. Hu, C. Hu, D. He, Q. Gu, A new ROS-based hybrid architecture for heterogeneous multi-robot systems, the 27th Chinese Control and Decision Conference (CCDC) IEEE,(2015) pp. 4721–4726. doi:10.1109/CCDC.2015.7162759.
[3] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A.Y. Ng, et al., ROS: an open-source robot operating system, ICRA Workshop on Open Source Software., 2009, vol. 3, p. 5, Kobe, Japan.
[4] K. Zheng, ROS navigation tuning guide, CoRR (2017). doi:10.48550/arXiv.1706.09068.
[5] P. Rezeck, H. Azpúrua, M. F. S. Corrêa, L. Chaimowicz, HeRo 2.0: A low-cost robot for swarm robotics research (2022). doi:10.48550/arXiv.2202.12391.
[6] M. Starks, A. Gupta, S. O. V. Sarma, R. Parasuraman, eRoSwarm: Fully-capable miniature swarm robot hardware design with open-source ROS support, 2022. doi:10.48550/arXiv.2211.03014.
[7] T. K. Kaiser, M. J. Begemann, T. Plattenteich, L. Schilling, G. Schildbach, H. Hamann, ROS2swarm - A ROS 2 package for swarm robot behaviors, submitted on May 3rd, 2024. doi:10.1109/icra46639.2022.9812417.
[8] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, W. Woodall, Robot Operating System 2: Design, architecture, and uses in the wild, submitted on November 14th, 2022. doi:10.48550/arXiv.2211.07752.
[9] J. Kennedy, R.C. Eberhart, et al., Particle swarm optimization, in: Proceedings of IEEE International Conference on Neural Networks, volume 4, Perth, Australia, 1995, pp. 1942–1948.
[10] G. Pereira, Particle swarm optimization, April 15, 2011.
[11] M. Dorigo, V. Maniezzo, A. Colorni, The ant system: an autocatalytic optimization process, Res. Rept. (1991) 91–016.

[12] S. Goss, S. Aron, J. L. Deneubourg, J. M. Pasteels, Self-organized shortcuts in the Argentine ant, Naturwissenschaften 76(12) (1989) 579–581. doi:10.1007/BF00462870.

[13] G. Bilchev, I.C. Parmee, The ant colony metaphor for searching continuous design spaces, in: T.C. Fogarty (Ed.), Proceedings of the AISB Workshop on Evolutionary Computation, Vol. 993 of LNCS, Springer, Berlin Heidelberg New York, 1995, pp. 25–39.

[14] M. A. El-Dosuky, CACO: Competitive ant colony optimization, a nature-inspired metaheuristic for large-scale global optimization, 2013. doi:10.48550/arXiv.1312.4044.

[15] D. Karaboga, An idea based on honey bee swarm for numerical optimization, Technical Report TR06, Erciyes University, Engineering Faculty, Computer Engineering Department, 2005.

[16] N. Lemmens, S. Jong, K. Tuyls, A. Nowé, Bee behaviour in multi-agent systems, in: Proceedings of the 9th International Conference on Adaptive Agents and Multi-Agent Systems (AAMAS), Springer, 2007, pp. 145–156. doi:10.1007/978-3-540-77949-0_11.

[17] S. Soragna, A. Carroll, M. Ge, Impact of ROS 2 node composition in robotic systems, submitted on May 17th, 2023. doi: 10.48550/arXiv.2305.09933.

[18] A. Tourani, et al., Visual SLAM: What are the current trends and what to expect?, Sensors 22(23) (2022) 9297. doi:10.3390/s22239297.

[19] D. Geromichalos, M. Azkarate, E. Tsardoulias, L. Gerdes, L. Petrou, C. Perez Del Pulgar, SLAM for autonomous planetary rovers with global localization, Journal of Field Robotics 37(5) (2020) 830–847. doi:10.1002/rob.21943.

[20] T. Yang, P. Li, H. Zhang, J. Li, Z. Li, Monocular vision SLAM-based UAV autonomous landing in emergencies and unknown environments, Electronics 7(5) (2018) 73. doi:10.3390/electronics7050073.

[21] J. Li, Y. Bi, M. Lan, H. Qin, M. Shan, F. Lin, B. M. Chen, Real-time simultaneous localization and mapping for UAV: A survey, in: Proceedings of the International Micro Air Vehicle Competition and Conference, 2016, Vol. 2016, p. 237.

[22] X. Gao, T. Zhang, Introduction to Visual SLAM: From Theory to Practice, Springer Nature, 1st ed. 2021 edition.

[23] G. Welch, G. Bishop, An introduction to the Kalman filter, University of North Carolina at Chapel Hill, 2001. URL: http://www.cs.unc.edu/~welch/media/pdf/kalman_intro.pdf.

[24] M. I. Ribeiro, Kalman and extended Kalman filters: Concept, derivation and properties, 2004. URL: http://users.isr.ist.utl.pt/~mir/pub/kalman.pdf.

[25] S. McEnsky, I. Jambrichich, SLAM Toolbox: SLAM for the dynamic world, Journal of Open Source Software, 2021. doi: 10.21105/joss.02783.

[26] A. Merzlyakov, S. McEnsky, A comparison of modern general-purpose visual SLAM approaches, IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2021. doi:10.48550/arXiv.2107.07589.