Programming and Compiler Toolchain for Swarm Robotic Systems

Isara Tillekeratne
Department of Computer Engineering
University of Peradeniya
Peradeniya, Sri Lanka
isara.tillek@gmail.com

Kavinaya Yogendren
Department of Computer Engineering
University of Peradeniya
Peradeniya, Sri Lanka
kavinaya1212@gmail.com

Hashini Wijerathne
Department of Computer Engineering
University of Peradeniya
Peradeniya, Sri Lanka
hashini.sharintha@gmail.com

Isuru Nawinne
Department of Computer Engineering
University of Peradeniya
Peradeniya, Sri Lanka
isurunawinne@eng.pdn.ac.lk

Mahanama Wickramasinghe
Department of Computer Engineering
University of Peradeniya
Peradeniya, Sri Lanka
mahanamaw@eng.pdn.ac.lk

Roshan Ragel
Department of Computer Engineering
University of Peradeniya
Peradeniya, Sri Lanka
roshanr@eng.pdn.ac.lk

***Abstract* - Swarm robotics is a field of research inspired by the collective behaviours seen in natural swarms. The primary characteristics of swarm robotics are agent homogeneity, autonomy, locality and decentralised control. We introduce a user-friendly Integrated Development Environment (IDE) equipped with a programming and compiler toolchain designed to be compatible with virtual and physical swarm robot platforms. The IDE includes a graphical, block-based programming interface to compose high-level algorithms following the bottom-up design approach. Specialised support for behaviours such as random movement with obstacle avoidance, dynamic task allocation, and object finding are provided. The IDE automatically converts graphical-level algorithms into C++ and Java and supports the compilation process. Generated executables can be executed on virtual and physical swarm robot platforms, supporting over-the-air (OTA) code uploading. Through this research, we accomplished our aim of developing a user-friendly platform allowing researchers and developers to program, compile, and execute diverse swarm behaviours accurately and efficiently. The efficacy of our work is validated using the results obtained by the swarm behavioural experiments done using the developed IDE.**

***Keywords* - Swarm Robotics, Integrated Development Environment (IDE), Programming Framework, Block-Based Programming, Bottom-Up Design, Decentralized Control, Dynamic Task Allocation, Object Finding Algorithm**

I. Introduction

Swarm robotics is a field of research and development inspired by the collective behaviour observed in natural swarms such as flocks of birds, schools of fish, and colonies of ants. The systems of multiple autonomous robots that work together to achieve common objectives lay the foundation of Swarm Robotics. It is based on the idea that individual agents, following simple rules, can collectively exhibit complex behaviours and accomplish tasks that would be challenging for a single robot to achieve. By coordinating their actions and interactions, the swarm of robots can achieve robustness, scalability, and adaptability. Research in swarm robotics focuses on developing swarm behavioural algorithms, communication protocols, programming frameworks, simulation platforms and hardware platforms.

When considering the characteristics of swarm robotics, the homogeneity of the agents, meaning that all the robots in the swarm have similar capabilities, is crucial. This homogeneity allows for simplicity in the design, as they can perform similar tasks and communicate with one another using the same set of rules. Being autonomous is another crucial characteristic of swarm robotics. The principle of locality and decentralised behaviour which is another fundamental in swarm robotics, states that the behaviour of each agent depends primarily on its local perception of the environment and the information it acquires from its immediate neighbours. It implies that instead of relying on a central controller, the swarm's behaviour arises from the individual agents' decisions. This decentralised nature enhances robustness, as the failure of a single agent does not significantly affect the overall outcome of the swarm.

Some of the swarm behaviours that have been experimented with are aggregation, dispersion, pattern formation, collective movement, task allocation, source search and collective transportation of objects. Real-world application domains such as search and rescue missions, environmental monitoring, warehousing and logistics, construction, manufacturing processes, surveillance and security are widely benefited from these swarm behaviours.

Achieving a complex collective swarm behaviour is not an easy task, as it requires handling the complexities of programming each robot and the interactions between them. Most of the available frameworks for swarm programming focus only on software-level simulations, which do not discuss extending them to real hardware robot platforms, and they are limited only to a few pre-programmed sets of behaviours and do not give developers the ability to re-program new behaviours.

From this research, we developed an Integrated Development Environment (IDE) that comprises a programming and compiler toolchain for swarm robots to overcome the mentioned complexities of swarm behavioural programming. The IDE is compatible with an existing virtual and physical swarm robot platform. The ability to compose high-level algorithms is the main characteristic of the developed IDE. It allows the users to program swarm behaviours in a graphical block-based interface in a code-less approach. The swarm behaviours are designed based on a bottom-up design approach and categorised into four levels, providing re-programmability. This abstraction manages the complexity of programming swarm behavioural algorithms. The IDE has special support for behaviours such as random movement with obstacle avoidance, dynamic task allocation, and object finding. Then, it automatically converts the graphical-level algorithm to C++ and Java programming languages and facilitates the compilation process. The generated executables can be executed on virtual and physical swarm robot platforms. In the context of the physical robot platform, the IDE supports over-the-air (OTA) code uploading which enables the re-programming of hardware devices remotely without physical access. These features enable the users to experiment with various swarm behaviours, from programming to executing them on real robot systems and observing the final outcome.

Multiple experiments with swarm behaviours were carried out using the developed IDE. The experiments of the dynamic task allocation behaviour showcased that the swarm robots converged to the desired task distribution while some robots specialised in specific task types. The object-finding behavioural experiments done using virtual and physical robots demonstrated the ability to find and converge to an object placed in the arena. These results validated the accuracy of the programming, compiling, and execution processes of the IDE. Finally, a qualitative performance analysis of the IDE was carried out.

II. Related Work

We conducted the literature review based on two main categories: swarm programming tools-based studies and swarm behavioural algorithms-based studies.

A. Swarm Programming Tools and Frameworks

Creating swarm intelligence programming frameworks with features such as visualisation tools, simulation environments, and optimization algorithms has completely changed the field of swarm intelligence by making it possible to create and test sophisticated algorithms more quickly and effectively. The following programming frameworks were well-identified in industrial usage and related robotics research.

Swarm-Bench [1] is a benchmarking framework designed to evaluate and compare swarm intelligence algorithms. Since it primarily focuses on benchmarking and evaluation rather than algorithm implementation, it may require additional programming work to integrate custom algorithms.SwarmOps [2] and PySwarm [2] are Python libraries specializing in swarm intelligence optimization, with a focus on algorithms like particle swarm optimization (PSO) and differential evolution (DE). PySwarm offers a simpler interface with customization options for algorithm parameters. In contrast, SwarmLib [3], a Java-based library, covers a broader range of swarm-based optimization techniques, including PSO, ant colony optimization (ACO), and artificial bee colony (ABC) algorithms. However, these libraries, including SwarmOps and PySwarm, are limited to optimization algorithms and lack support for real robot systems.

Physicomimetics [4] employs a virtual physics framework to simulate swarm behaviors by replicating the collective dynamics observed in living organisms. Although effective for simulating behaviors like pattern formation and obstacle avoidance, Physicomimetics is constrained by predefined rules, limiting its adaptability to dynamic real-world conditions. Buzz [5] [3] introduces a programming language and simulation framework tailored for swarm robotics system development. Offering a high-level language for specifying swarm behaviors and a simulator for testing algorithms, Buzz is hardware-independent, overcoming hardware dependency issues with its BuzzVM virtual machine. While it excels in facilitating coordination and communication among robots to implement complex collective behaviors, Buzz is primarily focused on these aspects, and its scope does not extend to the full spectrum of programmability required for diverse swarm behaviors in dynamic environments.

Research and development efforts on swarm robotics, which takes a bottom-up approach and emphasises decentralised interactions among individual robots, or agents, to produce collective behaviours, have been undertaken by iRobot [6]. Another research done for implementing a programming framework for swarm robots [3] employs a bottom-up design, emphasizing the integration of various built-in swarm robotic behaviours. This framework, designed for minimal resource hardware, includes a virtual pheromone-based communication system and IR sensors, offering the flexibility to create and integrate behaviours ranging from preliminary actions like moveForward and angularTurn to more complex global behaviours such as aggregate and pattern formation, demonstrated through tests on a Java-based simulation platform.

While each tool or framework has its specific features, some common limitations can be identified among them. Some tools may focus on specific swarm intelligence algorithms, such as PSO or ACO while neglecting other techniques. As the size of the swarm rises, the scalability and performance of the tools may become an issue. While many tools provide default parameter settings for the algorithms, the ability to easily customize them and incorporate domain-specific knowledge may vary. Integrating swarm intelligence tools with existing systems or frameworks can be challenging. Compatibility issues, dependencies, or lack of interoperability with other libraries or platforms may require additional effort or workarounds.

Some tools are compatible with either physical or virtual robots but not with both. Therefore, considering these common challenges is essential when evaluating swarm programming tools and frameworks.

B. Swarm Behavioural Algorithms

Flocking is one of the behavioural algorithms that we encountered. Flocks are aggregations of many individuals which move together with cohesion, flexibility, and alignment. Craig Reynolds [8] introduced flocking behaviour to a swarm of simulated birds called Boids, comprising three main rules: collision avoidance, flock centring, and velocity matching. Christoph Moeslinger [9] and his colleagues proposed an algorithm which doesn't require communication, memory or global information. The results showcased that both the mobility of the flock and aggregation time depended on the size of the flock, and the algorithm seemed to work well with comparatively small swarms.

Earlier approaches to pattern formation in swarm robotics have primarily been based on the centralised strategy that requires global knowledge. In contrast, the approach proposed by Mehmet Serdar Güzel and his team [10] introduces an algorithm that allows the swarm to form a circle pattern autonomously while providing collision prevention and adaptability in cluttered environments.

Sakthivelmurugan and his team [11] conducted a study on foraging behaviour, proposing different strategies for item searching including the expanding square, parallel sweep, and divider approaches. Statistical analysis was conducted to compare the strategies and the experiments revealed that the parallel sweep with divider policy achieved the fastest item detection time compared to the other tested strategies.

In parallel with the progress made in analyzing foraging behaviour, Obute and his team [12] introduced the Repulsion-Attraction (Rep-Att) algorithm. Inspired by natural swarm behaviours, the algorithm uses sound signals to enable direct communication among robots. The algorithm uses repulsive signals to move robots away from the nest in search of the target and attraction signals to attract other robots to the detected targets. The simulation results have shown that the Rep-Att algorithm improves the foraging efficiency compared to random walk.

Dynamic task allocation, another important swarm behaviour, refers to the process of assigning tasks among individual robots in a swarm dynamically and adaptively. Based on the decentralised strategy, Wonki Lee and DaeEun Kim have proposed an algorithm [13] to handle dynamic task allocation paired with the object foraging task using the response threshold model. The results have shown that the system can stochastically converge to the equilibrium of the desired task distribution. Furthermore, the swarm can adapt to the dynamic changes of the environment such as adding or removing robots or tasks.

Nedjah and colleagues [14] conducted another research, focusing on the problem of task allocation in robotic systems using evolutionary optimization algorithms. They suggest breaking down complex tasks into simpler ones and coordinating their execution as a solution. The paper highlights the importance of adapting task allocation to changes in the environment and swarm performance instead of using a centralized approach. To enhance the allocation process, different techniques are proposed, including a distributed job allocation algorithm based on Particle Swarm Optimization (PSO).

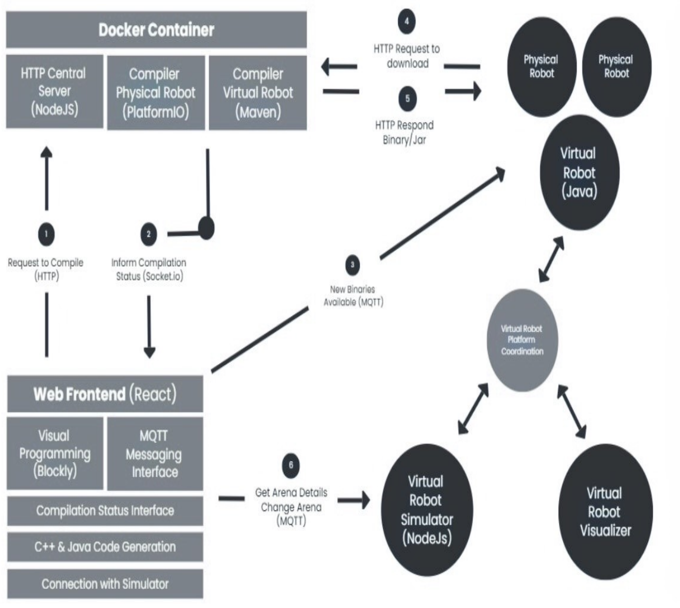

Docker Container
HTTP Central Server (NodeJS)
Compiler Physical Robot (PlatformIO)
Compiler Virtual Robot (Maven)
4
HTTP Request to download
5
HTTP Respond Binary/Jar
Physical Robot
Physical Robot
Virtual Robot (Java)
1
Request to Compile (HTTP)
2
Inform Compilation Status (Socket.io)
3
New Binaries Available (MQTT)
Virtual Robot Platform Coordination
Web Frontend (React)
Visual Programming (Blockly)
MQTT Messaging Interface
Compilation Status Interface
C++ & Java Code Generation
Connection with Simulator
6
Get Arena Details Change Arena (MQTT)
Virtual Robot Simulator (NodeJs)
Virtual Robot Visualizer

Fig. 1. Solution Architecture.

III. Methodology

Our research focuses on demonstrating the effectiveness of a software engineering strategy aimed at overcoming challenges in developing swarm robotic applications. We center our approach on delivering a complete solution via a user-friendly Integrated Development Environment (IDE). This IDE is specifically designed to efficiently create high-quality swarm behaviours. The main objective of our IDE is to enable users, including those without programming expertise, to the smooth creation and compilation of codes customized for both physical and virtual robots. Moreover, the IDE plays a vital role in seamlessly bringing together these platforms, facilitating thorough testing and practical applications in real-world scenarios.

A. Establishment of System Architecture

At the foundation of our approach is establishing a solid system architecture that effortlessly connects physical and virtual robot platforms with the IDE. Fig. 1 shows our solution architecture based on previous research about physical and virtual robot platforms [17].

The physical robot operates binaries written in C++. Its hardware is set up with components for understanding its surroundings and communication. In the front, it has a distance sensor and a color sensor, providing the robot with the capability to sense its surroundings. Additionally, a Neopixel LED is incorporated, allowing for visual indicators or signals. To help it communicate and interact, the robot has four IR sensors in different directions. These sensors allow the robot to send and receive signals using infrared light. Furthermore, the robot has a WiFi module, enhancing its connectivity.

The PeraSwarm Virtual robot platform comprises a Java virtual robot application, a Node.js simulator, and a Visualizer interface. The simulator has the ability to give distance and colour readings for specific angles, change the colour of robots, and configure the virtual platform arena. The virtual robot code is responsible for generating robots, controlling their behaviour based on algorithms, and facilitating inter-robot communication. The research primarily focuses on developing an IDE customized for creating complex swarm behaviours, emphasizing the significance of testing and simulation within a virtual robot platform before transitioning to a physical robot platform.

B. *Development of IDE Components*

The IDE comprises a React frontend and a Docker container containing a Node.js backend, along with separate compilers for the virtual and physical platforms.

1) High-level algorithm composition: Our main goal is to facilitate high-level algorithm composition for users who may not have programming expertise, especially those involved in researching and analyzing swarm behaviours. With a specific focus on the educational sector, we have prioritized the development of a user-friendly interface. To meet these objectives, our system incorporates a block-based visual programming technique. Users can effortlessly design complex swarm behaviours by intuitively dragging and dropping blocks within the IDE. To achieve this user-friendly approach, we have integrated the Google Blockly library into our React frontend, providing an interface that not only ensures ease of use but also allows for the seamless conversion of visual representations into programming code.

Diverse Blocks for Varied Functionalities: When planning our design, we focus on abstraction and high-level modelling methods, following the principles of the Analysis and Design discipline. This includes creating various types of blocks, such as behaviour, I/O (for reading sensors and activating actuators), and general blocks (for programming elements like loops, conditions, and variables). These blocks cover specific functions like reading infrared, colour, distance, and proximity sensors, as well as controlling neopixel LEDs.

Our approach to designing behaviour blocks follows a bottom-up method, as illustrated in Fig. 2, starting with basic (atomic) behaviours and gradually progressing to more complex global-level behaviours.

- Global behaviours: Complex collective behaviours representing the highest level of complexity.
- Cluster behaviours: Behaviours involving multiple entities, forming an intermediate level of complexity.
- Pair behaviours: Behaviours specifically involving two entities, such as two robots or a robot and an object.

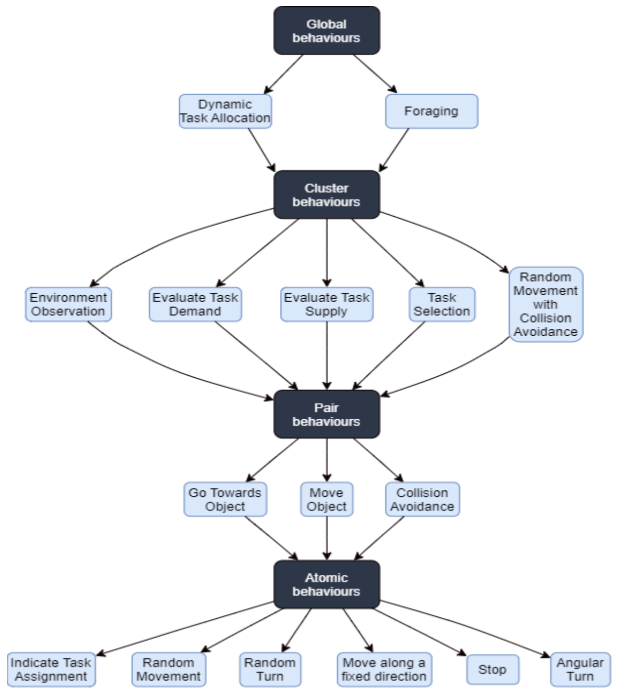


Fig. 2. Bottom-up Approach.

- Atomic behaviours: Basic behaviours executed by a single robot without interaction or sensing involvement with others.

This structured hierarchy allows users to smoothly combine low-level behaviours, building complex collective swarm behaviours. By using low-level behaviour blocks to construct high-level behaviours, we ensure consistency between the behaviour in both physical and virtual platforms. Atomic behaviours, being the basic and straightforward units of behaviour, are a common factor in both platforms, guaranteeing uniformity in their execution.

2) Conversion of Visual Representations to Code: The graphical-level algorithms created through block-based visual programming are converted into programming code using the Google Blockly library. Each block represents C++ and Java code, allowing users to visually design and manipulate code blocks to generate executable code for swarm algorithms.

3) Remote Cross-Compilation Support : The IDE combines remote cross-compilation support by using the PlatformIO (PIO) CLI and Maven. The backend handles tasks such as compiling the generated code, generating binaries for the physical platform using the PlatformIO CLI, and producing jar files for the virtual platform using Maven. This comprehensive approach lets the tool create programs that work on different platforms, giving us flexibility in deploying.

Moreover, the IDE offers developers the ability to manage version control for the compiled binaries and class files. This ensures that the latest versions are consistently available for deployment across the swarm of robots. The compiled binaries and class files are stored in a centralised repository, accessible by the robots through HTTP requests. This centralised storage mechanism provides efficient version control and seamless access for the deployment of the most up-to-date executables to the swarm of robots.

4) Over-the-Air (OTA) Code Upload and Execution: The system enables the download of new executables to the virtual and physical robots wirelessly using WiFi modules and a central server. This central server is configured to communicate with the swarm robots using the MQTT messaging protocol. When the swarm robots are ready for new programs, the central server sends them a message through MQTT to start the download. Then, the new binaries and jar files are packaged and sent back to the swarm robot using MQTT messaging. In the case of physical robots, the WiFi module receives the binaries over the air and stores them in its memory. For virtual robots, the downloaded jar file triggers the initiation of a new application, simultaneously terminating the existing one. This smooth transition ensures that the robot is ready to execute the new code, which may involve updated algorithms or new features. Throughout this process, the central server monitors the progress of the download and installation, making sure that each robot receives the correct executables and confirming the successful completion of the process. This approach allows for updates to be applied to the swarm robots efficiently and smoothly, eliminating the need for physical connections or manual help.

5) Enhancement of IDE Features : As we continued working on the development, we made the IDE even better by incorporating more interesting features. These additional features include the capacity to program new behaviours, manipulate and visualize the virtual arena, and concurrently generate executables for multiple robots. Furthermore, the IDE includes a collection of built-in algorithms, giving users the option to use them directly without the need to create new ones from scratch. Moreover, the connectivity between the IDE and both virtual and physical robot platforms is enhanced through the integration of the MQTT message-sending feature. This feature makes it easy for the IDE to communicate and control the robots smoothly, ensuring efficient coordination and interaction. The integration of these additional features and strengthened connectivity further enables users in creating, testing, and refining swarm behaviours within the developed IDE.

6) Validation through Experiments : To ensure the effectiveness and performance of the IDE, a series of experiments are formulated and executed. These experiments validate the tool's functionality in different scenarios, from foundational setup to feature enrichment, ensuring its adaptability and efficiency in creating complex swarm behaviours.

IV. Experiments

We conducted a series of experiments using the developed IDE to ensure the efficacy of our work. We started out with a simple behaviour of random movement while object detection using a single physical robot. Then, the experiments were expanded to dynamic task allocation and object-finding behaviours using both virtual and physical swarm robot platforms.

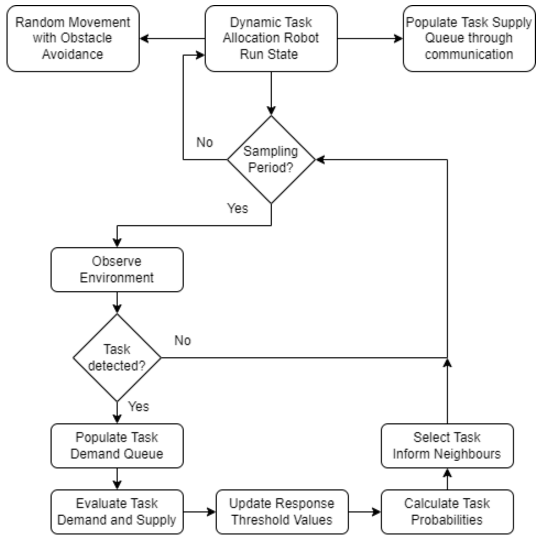
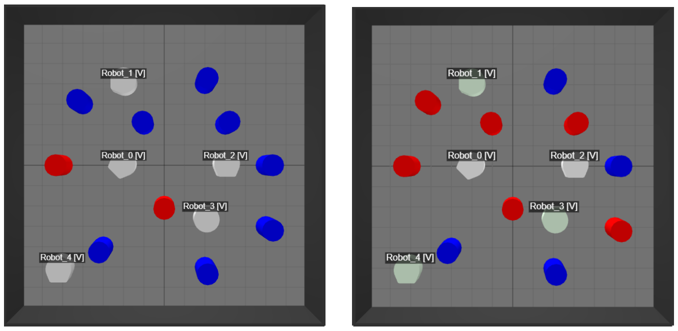

Random Movement with Obstacle Avoidance
Dynamic Task Allocation Robot Run State
Populate Task Supply Queue through communication
No
Sampling Period?
Yes
Observe Environment
Task detected?
No
Yes
Populate Task Demand Queue
Select Task Inform Neighbours
Evaluate Task Demand and Supply
Update Response Threshold Values
Calculate Task Probabilities

Fig. 3. Dynamic Task Allocation Behaviour Flow.

A. Dynamic Task Allocation Behaviour

The dynamic task allocation behaviour was tested in the virtual robot platform and the algorithm was inspired by the research conducted by Wonki Lee and DaeEun Kim [13]. In this experiment, initially, red and blue objects are placed on the robot arena randomly. The red and blue coloured objects denote two different tasks (red task, blue task). Then, the swarm robots which are initially assigned to task red are also placed in the arena randomly and they start moving around while avoiding the obstacles. The final goal is to converge and stabilise the swarm robots to the desired task distribution among the robots. The desired task distribution is defined by the distribution of the coloured objects.

1) Task demand and task supply estimation: As per the algorithm, each robot maintains two fixed-length queues to store the recent history that it sees with regard to task demand and task supply. The task demand corresponds to the colours of the neighbouring objects and the task supply corresponds to the tasks that are assigned to the neighbouring robots that it sees. The task demand information is collected by using the colour sensor readings in a specific time interval and the task supply information is collected by inter-robot communication. When a robot assigns a task to itself, it communicates that to the neighbouring robots such that the neighbouring robots can update their task supply queues. The reason to read the colour sensor readings in a specific time interval is to avoid updating the queues with duplicate information. Then, the global task demand and the supply of the environment for each type of task are estimated using the local history stored in the queues. For example, if the queue length is 10, and in the task demand queue, 6 elements correspond to the colour red, the task demand estimation for task red is 0.6 for that specific robot.



(a) Scenario 1 with 20% red and 80% blue.

(b) Scenario 2 with 60% red and 40% blue.

Fig. 4. Two Scenarios with Two Different Task Proportions.

2) Response threshold model: After calculating the task demand and supply estimations for each task type, the response threshold value for each task type is updated. This is a value between 0 and 1. The response threshold model is explained in detail in the research conducted by Wonki Lee and DaeEun Kim [13]. The logic behind the response threshold model is that if the task demand is higher than the task supply the response threshold for that certain task type decreases. Once the response threshold decreases, the probability of selecting that task type increases as per their proposed task selection probability function. This implies that when the task demand is higher than the supply for a specific task type, the robot becomes more likely to select it. When all the swarm robots behave in this same manner depending on their local information, the swarm collectively converges to the desired task distribution over time. This is an iterative process and proof is given that their response threshold model stochastically converges to the equilibrium of the desired task distribution. Fig. 3 shows the flow of the dynamic task allocation algorithm.

3) Characteristics of the behaviour: This behaviour does not depend on any global knowledge or a controller rather the control happens in a decentralised manner which preserves the qualities of the swarm behaviours. Each robot makes its own decisions following a simple set of rules based on the local information that it sees and collectively the swarm achieves the complex behaviour. As mentioned before, the algorithm has the ability to make the swarm robot system converge and stabilise in the desired task distribution. Furthermore, the robots have the ability to dynamically adjust themselves according to the changing environmental conditions such as adding or removing coloured objects or robots. The change of the response threshold values over time shows how certain robots tend to get specialised in doing a specific task type over time. These characteristics are discussed further along with the experiments conducted and the test results obtained.

4) The experimentation process : The first step is programming the dynamic task allocation behaviour using the block-based visual programming interface. The IDE provides multiple approaches for the user. The user has the ability to program the entire algorithm from scratch using the fundamental blocks. Another approach is using the in-built dynamic task allocation behaviour option directly. The third approach is using the level-based comprehensive set of behavioural blocks specifically designed for this behaviour. In this case, cluster behavioural blocks such as random movement with obstacle avoidance, observe environment, evaluate task demand, evaluate task supply, select task and atomic blocks such as show task which indicates the selected task by lighting up the neo pixels and sends messages to the neighbouring robots are designed to be used when programming the dynamic task allocation behaviour. In the experimentation process, the third approach was used for programming. Once the programming was completed, the Java option was selected to generate Java codes since the goal was to execute the program in the virtual robot platform. Then, the generated code was reviewed and the taskSet arena type was selected as the simulation environment. Afterwards, the robots were added by selecting the preferred locations on the arena grid displayed on the user interface. Subsequently, the code was compiled and once the compilation status was indicated as successful, the option was selected to execute the program and the newly programmed behaviour started executing on the virtual robot platform automatically. By evaluating the results obtained by the executed behaviour, conclusions can be drawn about the successful programming, compilation and execution process of the IDE.

B. Object Finding Behaviour

The object finding behaviour was tested on both virtual robot and physical robot platforms. The object finding algorithms employed in virtual robot and physical robot platforms differ due to available facilities, showcasing adaptability to the distinct challenges and capabilities of each environment.

1) Implementation of object finding algorithm in virtual robot platform : In this experiment, the virtual robot used object detection with obstacle avoidance to find an object of a specific colour in the environment and the robot employed proximity sensors to gather distance and colour information from various angles in its surroundings. The robot is programmed to detect objects with a predefined colour by exploring its surroundings and determines whether an object is present by comparing the colour of the object (predefined colour) with the colour it detected using proximity sensors. Upon detecting the target colour, the robot dynamically responds based on the proximity of the object. If the object is within a specified distance threshold, the robot transitions to a ”WAIT” state, signalling that the object has been found within the acceptable range. If the object is detected but is beyond the acceptable distance, the robot adjusts its position by turning toward the object’s direction and moving forward to reduce the distance. This adaptive response is contingent on the angle at which the object is detected. The robot changes its colour to object colour when the object is detected, aiding in the observation of its state.

2) Implementation of object finding algorithm in physical robot platform : The object finding behaviour of physical

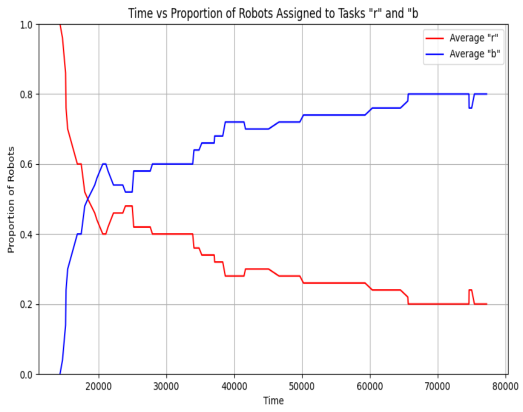


Fig. 5. Time vs Task distribution proportion of scenario 1.

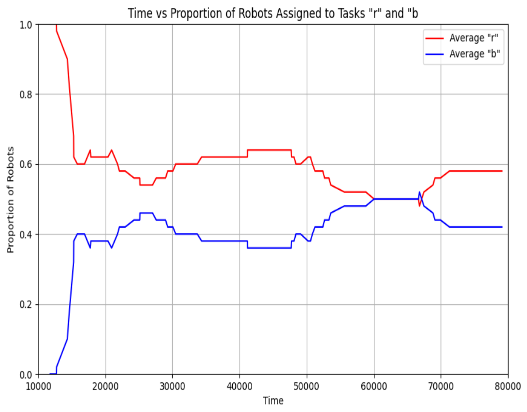


Fig. 6. Time vs Task distribution proportion of scenario 2.

robots involves the integration of a colour sensor and a distance sensor, both positioned at the front of the robot, which enables the robot to exclusively detect objects situated in its forward path. The robot utilizes a distance sensor to measure the distance between itself and potential obstacles. If the measured distance is less than the distance threshold, indicating the presence of a potential obstacle, the algorithm proceeds to randomly choose a direction (left or right) for corrective actions. Then it reads the color information from a color sensor to identify the color of the detected object. If the detected colour is indicative of the target object, the robot changes its colour to the object colour, and the robot comes to a stop, signalling that the object has been found within the acceptable range. If the detected colour does not match the target colour, the robot performs a backward movement for 1 second. It then enters a loop where it rotates until the distance to the obstacle is less than the distance threshold and the detected colour is not indicative of the target object. This loop has a maximum iteration count of 5. Regardless of the colour detection outcome, the robot colour visualization is adjusted accordingly. If no obstacle is detected within the threshold distance, the robot moves forward for 1 second.

C. *Contrasting object finding behaviour in virtual robot and physical robot: Navigating limitations*

When comparing the object-finding behaviours of virtual and physical robots, both systems have the same primary objective of discovering objects based on predefined object colours. However, their implementations diverge significantly. The virtual robot provides an adaptable method by modifying its movement in response to simulated sensor inputs. It is outfitted with a virtual proximity sensor at multiple angles. The physical robot, on the other hand, can only perceive one direction because it is dependent on front-facing colour and distance sensors. The virtual robot's adaptability is curtailed only by simulated conditions, while the physical robot faces real-world challenges, such as rotating to identify unobstructed directions.

V. Results and Discussion

A. *Dynamic Task Allocation Behaviour*

1) Testing of convergence to the desired distribution: Multiple tests were conducted to test different characteristics of the dynamic task allocation behaviour. The main characteristic tested was the convergence to the desired task distribution. The dimensions of the robot arena were defined as 70×70 units^2, with the objects characterised by a radius set to 5 units. For this, ten red and blue coloured objects were placed on the arena randomly. The simulation environment was set with five robots which were initially assigned to task red and placed around the arena randomly. The tests were conducted using two scenarios with two different task proportions as shown in the Fig. 4. In the first scenario (Fig. 4(a)), the task proportion was set to 20% red, 80% blue which indicates that two objects are red and 8 objects are blue out of the ten objects placed. In the second scenario (Fig. 4(b)), the task proportion was set to 60% red, 40% blue which indicates that six objects are red and four objects are blue out of the ten objects placed. Each scenario was executed 10 times to present the average results.

The average results obtained for each of the scenarios are shown in Fig. 5 and Fig. 6. The results shown indicate the change in the task distribution proportion among the robots over time. In both scenarios, the proportion starts as 100% red, 0% blue, since all the robots are assigned to task red initially. As time elapses, the robots start switching between the two task types which makes the distribution proportion fluctuate. It is evident from the observations that, in every scenario, the robots have consistently converged to the intended task distributions and stabilised. In the initial scenario, this distribution is 20% red and 80% blue, while in the subsequent scenario, it is 60% red and 40% blue. The evidence presented supports the conclusion that in this experiment, the dynamic task allocation behaviour is executed as the user intended which implies the successful programming, compilation and execution process of the IDE.

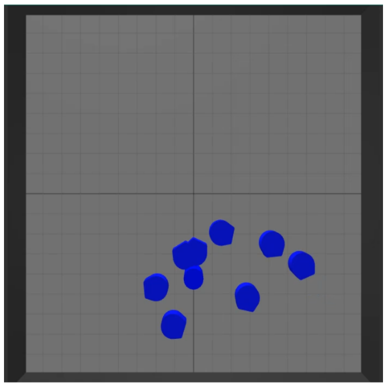


Fig. 7. Convergence of Eight Robots to Identify the Target Object.

2) Testing task specialisation of the robots: By monitoring the change of the threshold values for each task type over time, it can be observed that some of the robots get strongly specialised in performing a certain task type. If the response threshold of task red is very high (closer to 1) and the response threshold of task blue is very low (closer to 0), it implies that the specific robot is strongly specialised in performing task blue. This phenomenon can be identified as a replication of the division of labour in natural swarms where some individuals become more specialised in performing a specific task over time. For example, in an ant colony, some ants can be specialised in specific task types such as cleaning, foraging, and mound building. Individuals with strong specialisations typically switch tasks rarely. In contrast, individuals with weak specialisations can shift between tasks flexibly, according to the environmental conditions.

When obtaining the test results in Fig. 8 and Fig. 9, the dimensions of the robot arena were defined as 90×90 units^2, with the objects characterised by a radius set to 5 units. For this, ten red and blue coloured objects were placed on the arena randomly. The simulation environment was set with ten robots which were initially assigned to task red and placed around the arena randomly. The task distribution proportion was set to 40% red, 60% blue. Fig. 8 indicates the initial and the final threshold values for tasks red and blue for each robot. The results demonstrate that the robots with IDs 0, 4, and 7 exhibits a substantial increase in their final red threshold values, approaching approximately 1, accompanied by a corresponding decrease in blue threshold values, reaching nearly 0. According to that observation, it can be concluded that robots 0, 4, and 7 are strongly specialised in performing task blue. Supporting the same conclusion, Figure 9 illustrates the change in the response threshold values of the robot 4 over time.

B. Object Finding Behaviour

In this experiment to assess the object finding behavior in a virtual robot platform, eight robots were strategically positioned within an arena alongside a singular object designated to be identified. The object, initially assigned the colour blue, served as the target for the robots. As the algorithm initiated, the robots dynamically navigated the environment, leveraging object detection coupled with obstacle avoidance. After a period of execution, all eight robots successfully converged in proximity to the object and changed their colours to blue as illustrated in Fig. 7. The final outcome shows how well the algorithm works achieving the collective objective of identifying and localizing the object.

In this experiment evaluating object-finding behaviour on a physical robot platform, five robots were strategically positioned at different locations within an arena, with all being designated as targets represented by the colour blue. The algorithm was initiated, prompting the robots to dynamically navigate the environment using a combination of object detection and obstacle avoidance strategies. As the execution progressed, all five robots successfully converged in close proximity to the object, changing their colours to blue. The final outcome demonstrates the algorithm's effectiveness in achieving the collective goal of identifying and localizing the target object. However, an observed issue emerged during the experiment. At times, instead of converging near the target, robots tended to form a chain by closely following the nearest robot. This behaviour represents a challenge in the algorithm's optimization, and addressing this issue will be a focus for further refinement in future iterations of the experiment. Identifying and resolving such challenges is integral to ensuring the algorithm's robustness and enhancing its performance in real-world scenarios.

C. Qualitative Performance Analysis

This section discusses the qualitative performance analysis of the developed IDE in terms of three main characteristics; usability of the graphical programming interface, flexibility and re-programmability of swarm behaviours, and compatibility with both physical and virtual robot platforms.

1) Usability of the graphical programming interface : The IDE introduces a user-friendly graphical block-based interface, a significant enhancement to programming usability. Since this is a code-less approach, it is highly beginner-friendly and users with no prior programming knowledge can easily adapt to the platform and experiment with swarm behaviours. It stands out for its ability to mitigate the learning curve traditionally associated with programming frameworks using Domain-Specific Languages (DSL) and other code-based approaches. The intuitive visual approach facilitates quicker adoption and proficiency, making swarm robotics development more accessible to a diverse range of programmers.

2) Flexibility and re-programmability of behaviours: A pivotal aspect of the IDE is its capacity to foster flexibility in programming swarm behaviours, coupled with re-programmability. The users are given multiple options to programming swarm behaviours such as programming new swarm behaviours from scratch using the fundamental blocks, experimenting with in-built swarm behaviours such as dynamic task allocation, object finding, and random movement with obstacle avoidance, or using the provided behaviour-specified blocks to program them easily. Therefore, through its

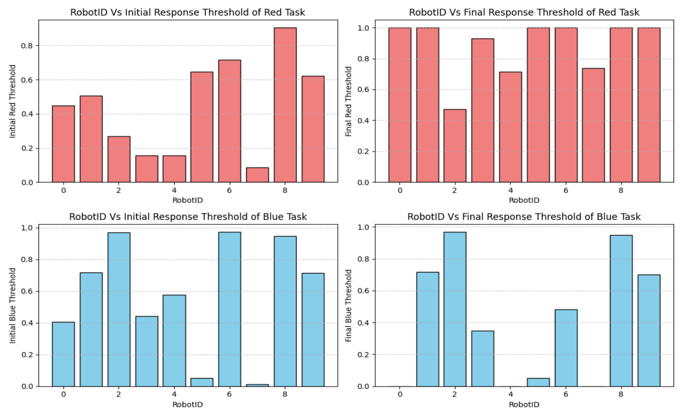


Fig. 8. Robot Id Vs Initial/Final Response Threshold Values.

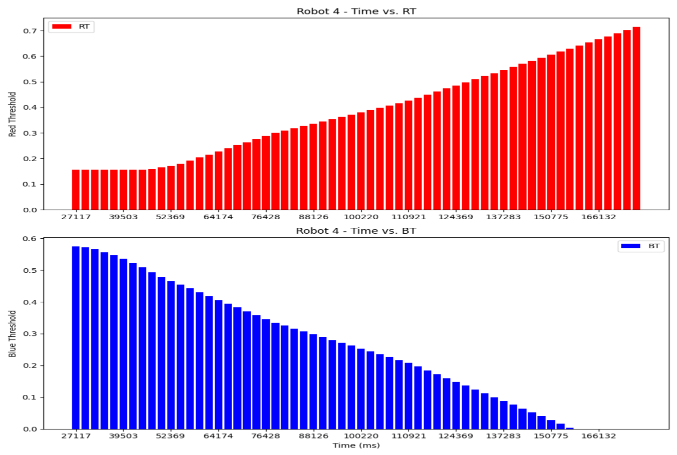


Fig. 9. Response Threshold Change of the Robot 4, over Time.

modular and scalable graphical blocks, users enjoy the freedom to experiment with and re-program behaviours seamlessly without limiting themselves to a fixed set of swarm behaviours.

3) Compatibility with Both Physical and Virtual Robot Platforms : The IDE transcends conventional limitations by offering compatibility with both physical and virtual swarm robot platforms. Once the behaviours are programmed, with only a few clicks, the users can easily execute them and observe the outcome in both virtual and physical robot platforms. This versatility facilitates a seamless transition between simulated environments and real-world implementations. Providing a unified platform for development and testing, the IDE ensures that the algorithms created can be readily applied and validated on both virtual and physical robot platforms.

VI. Conclusion

In this study, we have introduced a framework for developing and executing swarm behaviours in a unified environment that has focused on the integration of block-based visual programming with a user-friendly integrated development environment (IDE) across both virtual and physical platforms. The IDE with features like block-based visual programming, code generation, compilation, and OTA code upload, enhances the user experience and deployment flexibility. Significantly advancing swarm robotics, our decentralized dynamic task allocation and object-finding behaviours demonstrated the convergence of swarm robots and adaptability through extensive experiments. Inspired by the division of labour observed in natural swarms like ants and bees, our dynamic task allocation behaviour employs a decentralized approach, enabling each robot to make decisions based on local information. Our object-finding algorithms, experimented on both virtual and physical robot platforms, showcased a collective ability to identify and localize target objects in simulated and real-world scenarios. Overall, multiple experiments with swarm behaviours using the developed IDE have conclusively observed the accuracy and reliability of the programming, compiling, and execution processes and it can be identified as a useful tool for the educational and research sectors in the context of programming complex swarm behaviours.

References

[1] P. H. V. Lima, E. Ferrante, A. E. Turgut, and M. Dorigo, "Swarm-bench: A bench- marking toolkit for swarm robotics," Robotics and Autonomous Systems, vol. 97, pp. 10–20, 2017.

[2] Madhushanka, H. M. K., & Perera, A. L. H. E. (2023). *Swarm Intelligence Programming Framework* Literature Review. February.

[3] Dassanayaka, M., Bandara, T., Adikari, N., Nawinne, I., & Ragel, R. (2020, July). A Programming Framework for Robot Swarms. In 2020 Moratuwa Engineering Research Conference (MERCon) (pp. 578-583). IEEE.

[4] Spears, W.M., Spears, D.F., Heil, R., Kerr, W., & Hettiarachchi, S. (2005). An overview of physicomimetics. Lecture Notes in Computer Science, 3342, 84–97.

[5] Pinciroli, C., Lee-Brown, A., & Beltrame, G. (2015). Buzz: An Extensible Programming Language for Self-Organizing Heterogeneous Robot Swarms. http://arxiv.org/abs/1507.05946.

[6] McLurkin, J., Kaelbling, L.P. (1999). Stupid Robot Tricks: A Behavior-Based Distributed Algorithm Library for Programming Swarms of Robots.

[7] Yi, W., Di, B., Li, R., Dai, H., Yi, X., Wang, Y., & Yang, X. (Year not specified). An Actor-based Programming Framework for Swarm Robotic Systems.

[8] Reynolds, C.W. (1987). Flocks, herds, and schools. Computer Graphics, 21(4), 25–34.

[9] Moeslinger, C., Schmickl, T., & Crailsheim, K. (n.d.). LNAI 5778 - A Minimalist Flocking Algorithm for Swarm Robots.

[10] Güzel, M.S., Gezer, E.C., Ajabshir, V.B., & Bostancı, E. (Year not specified). An Adaptive Pattern Formation Approach for Swarm Robots.

[11] Sakthivelmurugan, E., Senthilkumar, G., Prithiviraj, K.G., & Tinu Devraj, K.R. (Year not specified). Foraging behavior analysis of swarm robotics system.

[12] Obute, Simon O., Dogar, Mehmet R., Dogar, Mehmet R. (2019): Simple Swarm Foraging Algorithm Based on Gradient Computation.

[13] Lee, W., & Kim, D.E. (2019). Adaptive approach to regulate task distribution in swarm robotic systems. Swarm and Evolutionary Computation, 44, 1108–1118. https://doi.org/10.1016/j.swevo.2018.11.005

[14] Nedjah, N., De Macedo Mourelle, L. (2015). Pso-based distributed algorithm for dynamic task allocation in a robotic swarm

[15] Brambilla, M., Ferrante, E., Birattari, M., & Dorigo, M. (2013). Swarm robotics: A review from the swarm engineering perspective. Swarm Intelligence, 7(1), 1–41.

[16] Navarro, I., & Matía, F. (2013). An Introduction to Swarm Robotics. ISRN Robotics, 2013, 1–10.

[17] Dilshani Karunarathna, Nuwan Jaliyagoda, Ganindu Jayalath, Janaka Alawatugoda, Isuru Nawinne, Roshan Ragel. Mixed-Reality based Multi-Agent Robotics Framework for Artificial Swarm Intelligence Experiments, IEEE Access, vol. 11, 2023