http://www.imavs.org/

Swarm-in-Blocks: Simplifying Drone Swarm Programming with Block-Based Language

Agnes Bressan de Almeida
João Aires Correa Fernandes Marsicano
University of São Paulo
São Carlos, Brazil
joao.aires.marsicano@usp.br

***Abstract*—Swarm in Blocks, originally developed for Copter-lack 2022, is a high-level interface that simplifies drone swarm rogramming using a block-based language. Building on the 'lover platform, this tool enables users to create functionalities ke loops and conditional structures by assembling code blocks. n 2023, we introduced Swarm in Blocks 2.0, further refining the latform to address the complexities of swarm management in a ser-friendly way.**

As drone swarm applications grow in areas like delivery, griculture, and surveillance, the challenge of managing them, specially for beginners, has also increased. The Atena team eveloped this interface to make swarm handling accessible ithout requiring extensive knowledge of ROS or programming. 'he block-based approach not only simplifies swarm control but lso expands educational opportunities in programming.

***Index Terms*—swarm robotics, block programming, drone warms, high-level interface, multi-robot coordination, educa-onal robotics**

I. INTRODUCTION

Swarm robotics is a field of multi-robot systems where a arge number of relatively simple robots cooperate to achieve omplex tasks. Inspired by natural systems such as ant colonies nd bird flocks, swarm robotics emphasizes the importance of ecentralized control and local interactions among robots [5]. 'he Swarm-in-Blocks project introduces a high-level interface ased on block programming, designed to make drone swarm rogramming accessible and straightforward, even for users ith limited technical knowledge [1].

II. PROJECT BACKGROUND

Swarm-in-Blocks originated as a project for CopterHack 022, designed as a high-level interface using block language o simplify the programming of drone swarms. Each script n this language represents functionalities such as conditional tructures, loops, or functions that handle swarm instructions. 'he platform is built on existing systems from COEX, specif-ally Clover. In 2023, Swarm-in-Blocks evolved into Swarm-n-Blocks 2.0, aiming to tackle more complex challenges ssociated with swarm robotics in an accessible and polished nanner [2].

III. SYSTEM ARCHITECTURE

The Swarm-in-Blocks framework provides a secure and calable platform for swarm robotics, consisting of the fol-owing components:

- **Block-Based Programming Interface:** A user-friendly, high-level programming interface that allows users to create scripts by fitting together code blocks like puzzle pieces.
- **Robotic Nodes:** Drones equipped with sensors and communication modules to interact with the programming interface and each other.
- **Control System:** Manages the execution of scripts and coordination of drone movements based on the programmed blocks.
- **User Interface:** Tools for monitoring and managing the swarm.

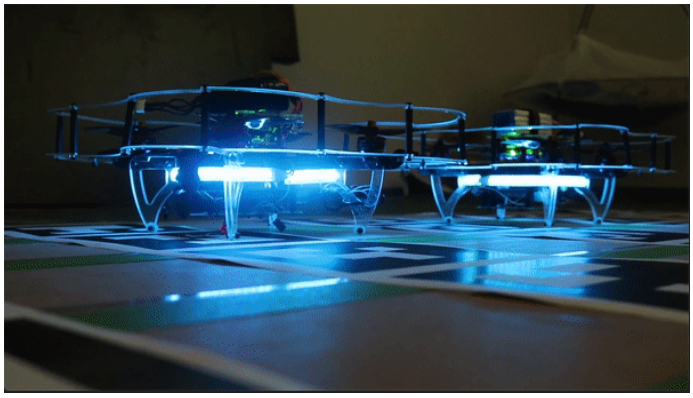


Fig. 1. Simple demo of the platform running.

A. Block-Based Programming Interface

The block-based programming interface is designed to be intuitive, enabling users to create complex swarm behaviors without needing to write traditional code. The interface includes blocks for various functions such as movement commands, conditional logic, and sensor data processing. This approach lowers the barrier to entry for programming drone swarms, making it accessible to a wider audience [3].

B. Robotic Nodes

Each drone in the Swarm-in-Blocks system acts as a robotic node, equipped with sensors for navigation and communication modules to interact with other drones and the control system. The drones are designed to be robust and capable

http://www.imavs.org/

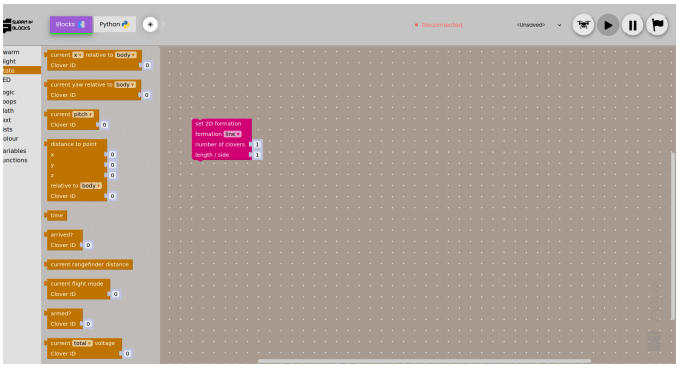


Fig. 2. Swarm Clover Blocks example.

f operating in diverse environments, from indoor spaces to utdoor fields [4].

.. Control System

The control system is responsible for executing the scripts reated using the block-based programming interface. It interrets the block sequences and translates them into commands hat are sent to the drones. The system also handles coordiation between drones, ensuring that they work together to chieve the desired objectives [5].

). User Interface

The user interface provides tools for monitoring and managng the swarm. Users can view the status of individual drones, ack their movements in real-time, and adjust parameters as eeded. The interface is designed to be intuitive and user-iendly, making it easy for users to manage complex swarm perations [3].

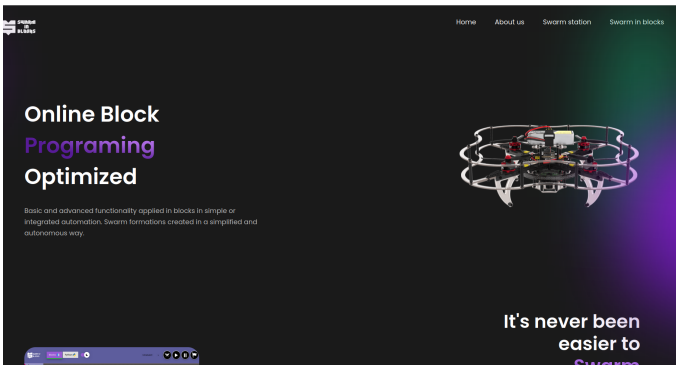


Fig. 3. Swarm in Blocks homepage

IV. APPLICATIONS

Swarm-in-Blocks has several potential applications, includng:

.. Industrial Automation

Swarm robotics can optimize industrial tasks like inventory nanagement and quality control. The Swarm-in-Blocks framework enables autonomous drones to navigate warehouses, perform stocktaking, and detect damaged goods, reducing manual labor and enhancing efficiency [4].

B. Agriculture

In agriculture, drone swarms can handle precision tasks such as crop monitoring, pest control, and irrigation management. The block-based interface simplifies the setup of these operations, improving resource use and crop yields through targeted interventions [5].

C. Entertainment

Swarm-in-Blocks facilitates the creation of aerial light shows by simplifying the programming of drone choreographies. The intuitive interface allows artists to design and execute captivating performances with ease [3].

V. BLOCK PROGRAMMING FOR GENERAL ROBOTICS

The ”Swarm-in-Blocks” project, developed for the Copter-Hack competition and enhanced in 2023 with Swarm-in-Blocks 2.0, offers a block-based programming interface for controlling drone swarms. This project not only advances swarm control technology but also holds significant educational potential, particularly as a teaching tool.

Block-based programming, widely used in educational robotics, simplifies the learning of programming. Platforms like Blockly and Scratch allow users to stack blocks representing code segments, making it easier for students of all ages to learn coding and computational thinking skills.

A. Educational Robotics

Educational robotics uses robots to facilitate learning in STEM education. Tools like Scratch and Blockly make programming accessible to young learners by allowing them to drag and drop blocks representing commands and logic structures [8].

B. Simplifying the Learning of Programming Concepts

Block programming makes learning programming concepts intuitive and accessible. It is particularly effective for introducing students to programming without the complexity of traditional coding languages.

C. Promoting Critical Thinking and Problem-Solving

Programming drones in formation fosters critical thinking and problem-solving skills. Students plan, test, and adjust instructions, considering variables like speed and altitude, enhancing their logical and systematic thinking.

D. Multidisciplinary Integration

Swarm-in-Blocks integrates concepts from mathematics, physics, computer science, and engineering. This multidisciplinary approach helps students apply theoretical knowledge to practical situations, essential in STEM education.

http://www.imavs.org/

. Preparation for the Job Market

By introducing drone programming and blockchain technolgy, Swarm-in-Blocks prepares students for future job marets. Practical experience in these areas offers a competitive dge in robotics and blockchain careers.

. Applications in Educational Settings

Robots like LEGO Mindstorms and VEX Robotics are used ı educational settings to teach programming and robotics. 'hese robots work with block-based coding environments, implifying the learning process [10], [9].

. Impact on Learning

Research shows that block-based programming enhances tudents' understanding of programming, improves problemolving skills, and increases engagement in learning [8], [9].

VI. SWARM IN BLOCKS SYSTEMS IN DETAILS

Due to the complexity of the Swarm in Blocks project, was divided into systems, which represent a macroscopic iew of sets of similar functionality. In this section, we will etail the developed systems, dealing not only with the features eveloped, but also with the motivation and implications for ıis.

. Formations

The "Formations" of the Swarm-in-Blocks project details ıe use of 2D and 3D formations for drone swarms. The 'ython script formation.py calculates drone positions for forıations such as lines, circles, squares, triangles, cubes, pyraıids, and spheres. Using the Numpy library, it generates arrays f coordinates for each drone. The swarm.py file handles the pplication of these formations, allowing users to select and isualize formations through a menu interface. This facilitates ıe creation and management of complex drone formations ith ease.

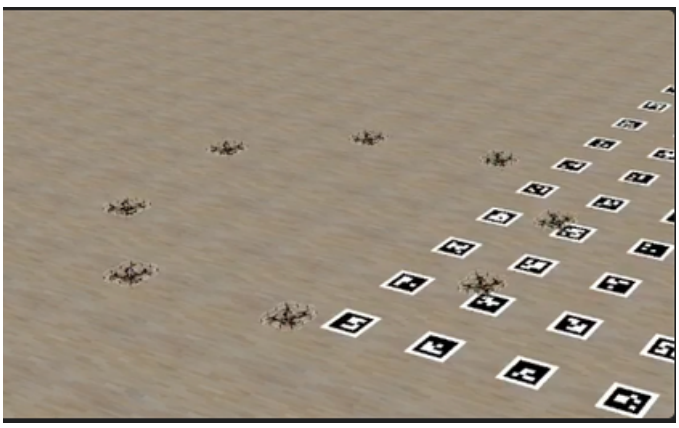


Fig. 4. Circle formation.

B. Transformations

The "Transformations" section of the Swarm-in-Blocks project covers operations that allow editing the drones' arrangement and the formation itself. Using matrix operations, procedures for translation, scaling, and rotation of the entire formation are developed. These transformations adjust the current coordinates of the drones, with special care for rotation, which requires centralizing the formation at the origin of the map. The collision prevention algorithm ensures that potential collisions are avoided during transformations.

C. Collision Avoidance

The "Collision Avoidance" section of the Swarm-in-Blocks project details an algorithm designed to prevent drone collisions within a swarm. The algorithm addresses three scenarios: one stationary and one moving drone, both drones moving on parallel paths, and both drones moving on non-parallel paths. It utilizes matrix operations and ROS (Robot Operating System) functionalities to manage drone positions and ensure safe distances are maintained, preventing collisions during swarm operations.

D. LED Effects

In the Swarm-in-Blocks project, the LED effects system allows for visual enhancements of drone formations. Ready-to-use effects include options for all drones, random drones, even and odd drones, and 2D formations. These effects can be controlled via user input, offering various modes like fill, fade, flash, blink, blink fast, wipe, rainbow, and rainbow fill. Each effect alters the LED behavior to enhance visibility and aesthetics, with some effects tailored to specific formations.

E. First Person View (FPV)

Part of the Swarm-in-Blocks 2.0, the First Person View (FPV) system enhances user experience by allowing real-time control of drones via a camera feed. Developed using roslibjs for web integration with ROS, it enables users to view and control drones through a web interface using JavaScript, HTML, and CSS. Users can select drones, view live footage, and control drone movements with keyboard inputs, improving flight safety and control.

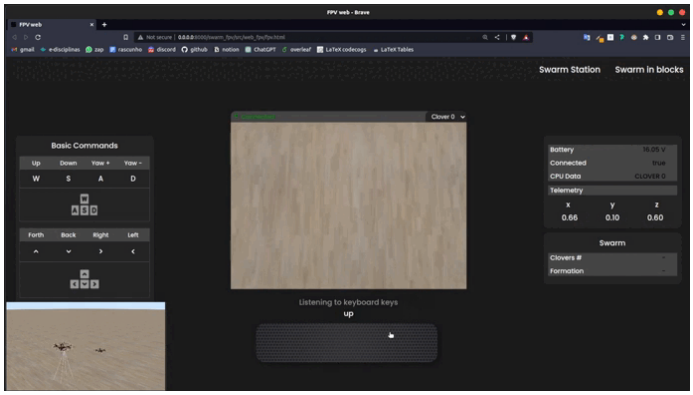


Fig. 5. Swarm in Blocks 2023 FPV.

http://www.imavs.org/

. *Smart Preview*

Swarm Preview in the Swarm-in-Blocks project enables sers to visualize swarm behavior in simulations or real life ith minimal computational power. It's used primarily in lanning Mode but also in Simulation and Navigation Modes. Jsers can choose 2D or 3D previews via the Python or Blocks PI. Functions include resuming, saving, and canceling simuations, and navigating through operations with control arrows identify issues.

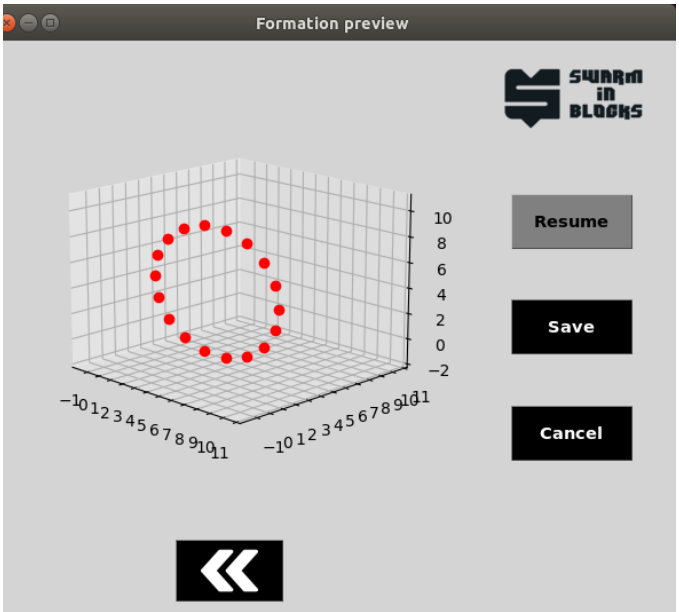


Fig. 6. 3D Plot Preview menu.

VII. CODE REVIEW

In this section, we delve deeper into the codebase that underins the core components of our Swarm-in-Blocks platform. he primary focus will be on examining the code responsible or the key pillars of our application: the **Swarm Clover locks**, the **homepage**, the **First Person View (FPV)** and e **Swarm Station**. These pillars form the foundation of our latform, each contributing unique functionality and ensuring eamless interaction between users and drone swarms.

The subsections that follow will present detailed explanaons and code snippets to illustrate how these pillars are onstructed, focusing on the logic and structure that enable e Swarm-in-Blocks platform to function effectively.

. *Swarm Clover Blocks*

The `swarm_clover_blocks` is a crucial component f the Swarm-in-Blocks platform, serving as the backend at enables block-based programming for drone swarms. his backend application extends the functionalities of the riginal `clover_blocks` package, integrating support for ontrolling multiple drones simultaneously, which is essential or swarm operations.

1) Code Structure: The `swarm_clover_blocks` package is organized into several directories and files, each playing a specific role in the overall functionality of the platform:

a) `msg/`: This directory contains the message definitions used within the system. These messages are essentially the data structures that encapsulate user requests and responses, along with any necessary identifiers. The use of ROS messages allows for standardized communication between different nodes in the ROS ecosystem.

b) `programs/`: In this directory, pre-configured examples of block-based programs are stored. These examples are complete with the necessary blocks to perform specific tasks, providing users with ready-made templates that can be easily customized through the frontend interface.

c) `src/`: This is the core of the backend, where the primary logic of the `swarm_clover_blocks` resides. The key file here is `clover_blocks.py`, which orchestrates all backend operations, including the execution of user-defined programs and management of communication between the frontend and the drones.

d) `srv/`: The `srv/` directory defines the service files, which specify the structure of requests and responses for operations like loading, running, and storing programs. These services are essential for the dynamic interaction between the frontend interface and the backend logic.

e) `www/`: This directory forms the heart of the frontend. It includes the core Blockly files as well as custom adaptations specific to the Swarm-in-Blocks platform. Key files include `blocks.js`, which defines the available blocks and their functionalities, `index.html` for the web interface, and `main.css` for styling the interface. Additionally, `ros.js` and `roslib.js` are used for communication with the ROS backend.

2) Node Functionality: The `swarm_clover_blocks` node is the operational backbone of the block-based programming interface. It implements all the necessary services and topics to run Blockly-generated Python scripts. The node's functionality includes:

- **Service** `run` – Executes the Blockly-generated program, interpreting the block sequence as a Python script and commanding the drones accordingly.
- **Service** `stop` – Provides the ability to terminate an ongoing program, ensuring that drone operations can be halted safely and immediately.
- **Service** `store` – Allows users to save their block-based programs for future use, facilitating easy access and reusability of code.
- **Service** `load` – Retrieves stored programs from the `programs/` directory, making them available for execution through the frontend.

3) Parameters and Topics: The `swarm_clover_blocks` node also manages several parameters and topics that govern its operation:

a) Parameters: The parameters control various aspects of the drone's behavior, such as navigation tolerances, yaw angle precision, and confirmation prompts before running a

http://www.imavs.org/

rogram. These parameters can be adjusted directly in the node onfiguration or via URL GET-parameters when accessing the /eb interface.

b) Published Topics: The node publishes several topics, ncluding:

- **`running`:** Indicates whether a program is currently being executed.
- **`block`:** Provides updates on the current block being executed, which is particularly useful for monitoring program progress.
- **`error`:** Reports any errors or exceptions that occur during program execution, ensuring that users are informed of issues in real time.
- **`prompt`:** Manages user input requests during program execution, allowing for interactive control of the drone swarm.

This comprehensive structure ensures that the `warm_clover_blocks` package can effectively manage ne complex task of programming and controlling drone warms, providing both a user-friendly frontend and a owerful backend.

For more detailed information about the Swarm Clover locks, please refer to the official repository available at warm-in-Blocks Repository.

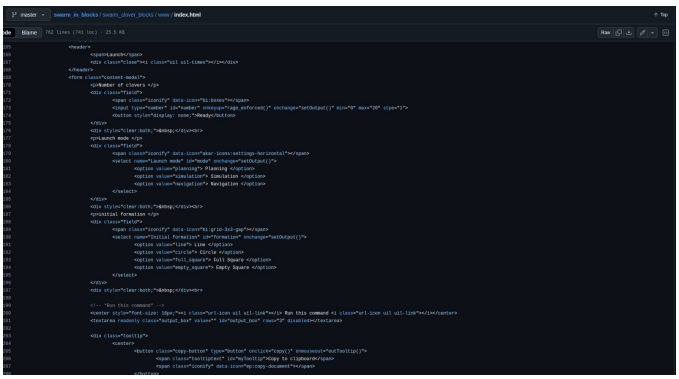


Fig. 7. Swarm Clover Blocks Code.

. homepage

The `Clover_UI` serves as the main user interface for the warm-in-Blocks platform, providing users with a comprehenive homepage to launch and manage both individual drones nd drone swarms. This interface is accessible through a web rowser by navigating to the localhost address configured via apache, making it easy to control and monitor drones directly rom any device on the same network.

1) Code Structure: The `Clover_UI` package is organized nto a ReactJS project, with a clear structure that facilitates evelopment, deployment, and maintenance. The following are ne key components of the project:

a) `cloverUI/`: This is the root directory of the ReacJS project. It contains all the essential files and folders needed o run the Clover UI, including the main HTML file and source ode.

b) `cloverUI/index.html`: This file is a crucial part of the ReactJS configuration. It acts as an entry point for the application, calling the main project components but not containing the actual application code.

c) `cloverUI/src/`: The `src/` directory is where the core logic and algorithms of the Clover UI reside. This is the primary source folder, containing all the React components and configuration files that drive the application's functionality.

d) `cloverUI/src/App.jsx`: The `App.jsx` file is the heart of the Clover UI project. It orchestrates the entire application by calling all the component pages. Each specific component's code is located in the `Components` folder, and `App.jsx` brings these components together to form the full user interface.

e) `cloverUI/src/Componentes/`: This folder contains the essential components that make up the application. Each component represents a specific part of the user interface, such as the header, carousel, or action buttons.

f) `cloverUI/src/style.js`: The `style.js` file contains Vite configuration settings that apply the defined styles dynamically. Vite is used to manage the project's build process, ensuring that styles are correctly applied during both development and production builds.

2) Frontend Technologies: The Clover UI leverages the ReactJS framework to create a responsive and interactive user interface. ReactJS was chosen due to its popularity, fast learning curve, and active community, making it an ideal choice for developing a robust and scalable frontend.

Additionally, the Clover UI takes advantage of complex visual effects, such as the Slider Carousel, to enhance the user experience. ReactJS's flexibility and ease of use enabled the development of a platform that is both functional and user-friendly, even though the page does not have its own backend.

To further streamline the development process, Vite, a ReactJS project builder, was used. Vite simplifies the creation and configuration of ReactJS projects, providing a pre-configured environment that is ready to use. This allowed the development team to focus on building the user interface rather than dealing with complex setup processes.

For more detailed information about the homepage, please refer to the official repository available at Swarm-in-Blocks Repository.

C. First Person View (FPV)

The development of the Swarm First Person View (sFPV) system allows drone pilots to control flights in real time through a camera installed on the drone. This feature has been restructured to run entirely on the web, integrating seamlessly with the Swarm Station. The sFPV system enhances drone operation by providing real-time visuals and data, such as battery levels, telemetry, CPU status, and flight state, thereby improving flight safety and control.

1) Code Structure: The `sFPV` package is organized into two main parts: the backend and the frontend. Each plays a critical role in ensuring the system's functionality:

a) msg/: This directory contains the messages used to andle user requests and manage communication within the ystem. These messages are essential for relaying commands nd responses between the frontend interface and the backend ervices.

b) src/: The `src/` directory houses the core script les needed to run the FPV application. These scripts are esponsible for capturing the drone's camera feed, processing , and relaying it to the user interface for real-time display.

c) launch/: The `launch/` directory contains the necssary launch files to execute the FPV system. These files llow users to initiate the application from the terminal, setting p the environment and connecting the necessary ROS nodes or the operation of sFPV.

2) Frontend: The frontend of the FPV application is built sing `roslib.js` to facilitate communication with the ROS ackend. The web interface, developed with HTML and CSS, llows users to control the drone through their browser. This nterface displays the live video feed from the drone's camera, long with essential telemetry data, enabling precise and safe rone operation. The user can interact with the drone using omputer keyboard inputs, adding another layer of control irectly from the web interface.

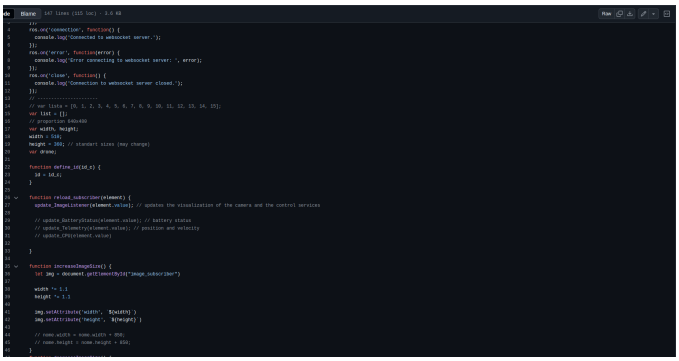


Fig. 8. First Person View Code.

For more detailed information about the sFPV, please refer o the official repository available at Swarm-in-Blocks Reposory.

). *Swarm Station*

The `swarm_station` package is responsible for runing the backend of the Swarm Station, which provides a ully integrated control center for managing drone swarms. 'his package utilizes the `tf2_web_republisher` package longside the `roslibjs` and `ros3djs` libraries to serve a D web visualizer. The system runs almost entirely on the veb, allowing users to access the Swarm Station from any evice connected to the same network as the host.

1) Code Structure: The `swarm_station` package is oranized into several key directories, each playing a crucial role n the overall functionality of the Swarm Station:

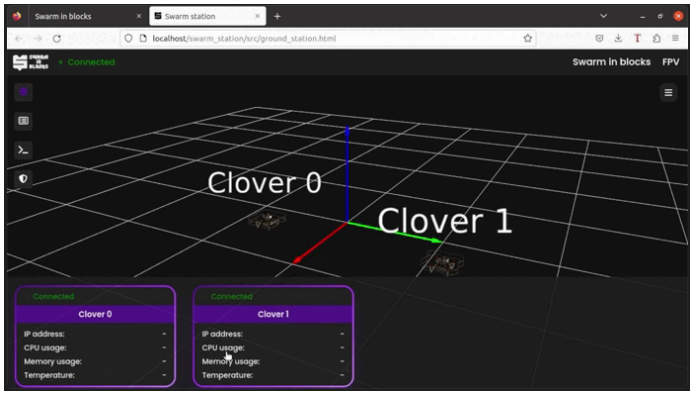


Fig. 9. Swarm in Blocks Swarm Station.

a) launch/: The `launch/` folder defines and initializes nodes, services, and parameters necessary to start the application. It also includes specific configurations required for the package.

b) meshes/: This directory contains definitions of the physical and dynamic characteristics of the Clover's 3D model used for simulation purposes.

c) src/: The `src/` directory is the core of the package, where source code is developed and compiled. It includes the definitions responsible for creating nodes, services, and other functionalities related to the frontend of the Swarm Station.

2) Main Features: The Swarm Station includes several integrated features designed to streamline swarm management:

a) Information Center: A central hub for displaying essential data on each drone, including an emergency "land all" button for quick response.

b) Drone's Process: Displays hardware information from each Clover's Raspberry Pi, making it easier to identify and compare potential issues.

c) Topic List: A feature that lists currently active topics, allowing users to monitor and analyze ongoing processes without needing an additional terminal.

d) Web Terminal: A web-based terminal that allows users to send commands directly from the web interface, avoiding the need for multiple screens. Multiple terminals can be opened simultaneously for handling various processes.

e) Safe Area: A feature that lets users define a safe operating area for the drones. If a drone exits this area, it is automatically landed, enhancing safety.

For more detailed information about the Swarm Station, please refer to the official repository available at Swarm-in-Blocks Repository.

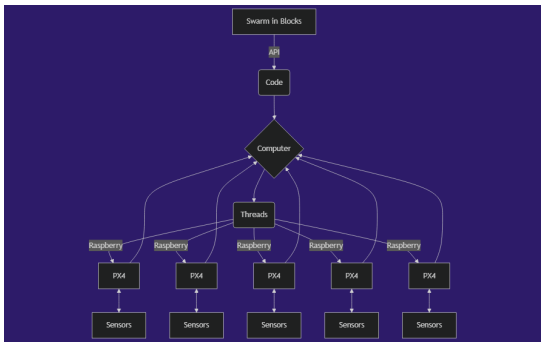
VIII. COMMUNICATION GRAFICS

The Swarm-in-Blocks platform effectively integrates MAVROS, ROS, and its web-based interface to manage and control multiple drones. MAVROS acts as a bridge between the drone's flight controllers and the ROS ecosystem, translating drone-specific commands and telemetry into a ROS-compatible format.

http://www.imavs.org/

ROS serves as the central hub, coordinating the flow of ıformation between the drones and the user interface. The warm-in-Blocks interface, built using `roslibjs`, commu-icates with ROS nodes in real-time, allowing users to control rones through an intuitive block-based programming environ-ıent.

Commands created in the interface are converted into ROS ommands and sent to MAVROS, which executes them across ıe connected drones. The drones' telemetry and status updates re fed back through this communication chain, ensuring that sers can monitor and adjust operations as needed.



ig. 10. Schematic of communication flow between MAVROS, ROS, and warm-in-Blocks.

This streamlined communication framework ensures cohe-ive and synchronized control of the entire drone swarm.

IX. SIMULATION ANALYSIS

Simulation analysis is essential for validating and opti-ıizing robotic systems before real-world deployment. By sing advanced tools like Gazebo, we can recreate complex cenarios, evaluate system performance, and identify potential ssues in a virtual environment. This section focuses on ıtegrating two systems within Gazebo, covering the setup rocess, system interaction, and outcome analysis to refine esign and functionality.

This method provides a thorough evaluation of system ehavior under various conditions, allowing developers to nhance reliability and efficiency. Whether testing a single obot or a complex multi-agent scenario, the insights from ıese simulations are crucial for guiding development and nsuring successful deployment.

. *The Systems*

The first system features an Nvidia RTX 3060 mobile GPU, n Intel i7-11800H processor, and 16GB of DDR4 RAM, ffering a strong blend of graphical and computational power or smooth, high-resolution simulations in Gazebo. The second ystem, with an AMD Ryzen 5 3600 CPU, a Radeon RX 5700 XT OC GPU, and 16GB of DDR4 RAM, provides robust rocessing and excellent graphical rendering, making it well-ıited for intensive simulation tasks requiring both CPU and GPU performance.

B. *Real Time Factor*

The Real-Time Factor (RTF) in Gazebo serves as an indi-cator of how efficiently the simulation runs compared to real-time. Ideally, an RTF of 1.0 signifies that the simulation is running exactly in real-time, while an RTF below 1.0 indicates that the simulation is running slower than real-time, typically due to increasing computational demands.

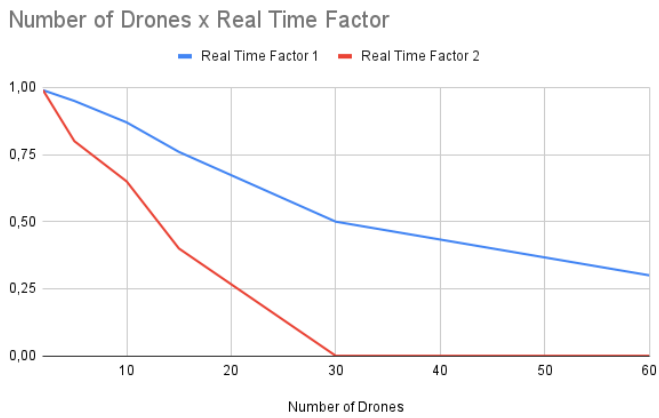


Fig. 11. Comparisson of RTF between 2 systems.

.

In the graph provided, the blue line represents the perfor-mance of the AMD-based system (AMD Ryzen 5 3600 with Radeon RX 5700 XT OC), while the red line represents the Nvidia-based system (Intel i7-11800H with Nvidia RTX 3060 mobile). As the number of drones increases, the RTF decreases for both systems, reflecting the higher computational load.

However, a critical observation is that when the number of drones exceeds 10, the RTF for the Nvidia-based system (red line) drops precipitously and eventually crashes the system, as indicated by the RTF reaching near zero. This crash highlights the Nvidia-based system's inability to handle the simulation workload as effectively as the AMD-based system, which, although also experiencing a decline in RTF, manages to sustain the simulation for a larger number of drones. This suggests that the AMD system provides better stability and performance under high-load conditions in Gazebo, whereas the Nvidia system struggles and ultimately fails when the number of drones increases beyond 10.

C. *Analysis*

The scalability of the system is directly influenced by the performance of its central node. The AMD-based system (Ryzen 5 3600 with Radeon RX 5700 XT OC) demonstrates better scalability, as it can manage an increasing number of drones without crashing, making it suitable for expanding the simulation's complexity. In contrast, the Nvidia-based system (i7-11800H with RTX 3060 mobile) shows limited scalability, struggling with more than 10 drones, which leads to system crashes and hinders the ability to scale the simulation effectively.

http://www.imavs.org/

http://www.imavs.org/

To achieve greater scalability, upgrading to more powerful nd modern components would enhance the system's capacity ɔ handle larger simulations and real life number of drones. 'his would ensure that the central node maintains performance nd stability as the number of drones and the complexity of the imulation increase, allowing for a more scalable and robust ɔlution in demanding environments.

X. RESULTS AND DISCUSSION

The Swarm-in-Blocks framework has shown significant nprovements in drone swarm coordination and ease of use. :s block-based interface allows for quick prototyping and ɛrative testing, enabling users with minimal programming xperience to manage drone swarm operations effectively [3].

.. *Scalability and Flexibility*

Swarm-in-Blocks is highly scalable, supporting large numers of drones without added complexity. The modular blocks ffer flexibility in defining and modifying swarm behaviors, ssential for large-scale applications like environmental mon-:oring and disaster response [4].

'. *Safety and Reliability*

Decentralized control mechanisms enhance the system's afety and reliability. Autonomous operation and redundancy nsure the swarm continues functioning even with drone ıalfunctions [5].

'. *Performance Evaluation*

Swarm-in-Blocks handles real-time operations with minimal ıtency. The block-based interface simplifies programming, al-ɔwing users to focus on high-level strategies. Benchmark tests onfirm efficient performance across various environments [3].

). *User Feedback*

Users have praised the intuitive design and ease of use. Ed-cational institutions find it particularly valuable for teaching warm robotics, bridging the gap between theory and practice 4].

XI. FUTURE WORK

Swarm-in-Blocks is continuously evolving, with future work ɔcusing on:

.. *Integration with Other Platforms*

Integrating with other robotic platforms for seamless in-ɛroperability, enabling comprehensive solutions for complex ısks [3].

'. *Enhanced User Interface*

Improving the user interface with more customization, real-me feedback, and monitoring tools to simplify large swarm ıanagement [4].

'. *Field Testing and Validation*

Conducting extensive field testing in diverse environments ɔ ensure robustness and identify areas for improvement [5].

XII. CONCLUSION

The Swarm-in-Blocks project simplifies the programming of drone swarms using a block-based language, making it accessible to users with limited technical knowledge. The Swarm-in-Blocks framework provides a user-friendly and scalable platform for communication and coordination, with potential applications in areas such as environmental monitoring, disaster response, industrial automation, agriculture, and entertainment. Future work will focus on further optimizing the framework and exploring additional applications in various domains.

ACKNOWLEDGMENTS TO THE ATENA TEAM

We would like to express our deep gratitude to the Atena team, part of the SEMEAR Group (Solutions in Mechatronic Engineering and Robotics Application) at the University of São Paulo (USP), São Carlos campus.

Comprising over 100 members, including undergraduate and graduate students, and supported by professors, the Atena team participated in competitions such as CopterHack. Team members include Agnes Bressan de Almeida, João Aires Correa Fernandes Marsicano, Felipe Andrade Garcia Tommaselli, Gabriel Ribeiro Rodrigues Dessotti, José Carlos Andrade do Nascimento, Lucas Sales Duarte, Matheus Della Rocca Martins, and Nathan Fernandes Vilas Boas. Their dedication and collaboration were essential for the project's success.

For more information, visit the SEMEAR website.

XIII. REFERENCES

REFERENCES

[1] Swarm-in-Blocks GitBook. (n.d.). Retrieved from https://swarm-in-blocks.gitbook.io/swarm-in-blocks
[2] Swarm-in-Blocks GitHub Repository. (n.d.). Retrieved from https://github.com/Grupo-SEMEAR-USP/swarm_in_blocks
[3] DroneBlocks. (n.d.). Retrieved from https://droneblocks.io
[4] Droneblog. (2023). The Making of A Drone Swarm: Everything You Need To Know. Retrieved from https://www.droneblog.com
[5] U.S. GAO. (2022). Science & Tech Spotlight: Drone Swarm Technologies. Retrieved from https://www.gao.gov
[6] Control.com. (2022). How to Train a Robot: Block-Based Programming. Retrieved from https://control.com
[7] Robots.net. (2023). What Is Block Coding. Retrieved from https://robots.net
[8] International Journal of STEM Education. (2022). The effects of educational robotics in STEM education. Retrieved from https://stemeducationjournal.springeropen.com
[9] Springer. (2021). Teaching Digital, Block-Based Coding of Robots to High School Students. Retrieved from https://link.springer.com
[10] iRobot Education. (2023). Coding Robots, Learning Library STEM Outreach. Retrieved from https://edu.irobot.com