100-Mouse System: a Swarm Robotics Platform and its Intuitive State Management User Interface

Ryusei Matsumoto[1,2] , Shota Yamamoto[1,3], Yoko Sasaki[1], Keisuke Okumura[1,4]

***Abstract*— This paper introduces a multi-robot coordination testbed called *the 100-mouse system* and its state management user interface (UI). *The 100-mouse system* is an actual multi-robot platform with 100 robots implemented with ROS2 for testing Multi-Agent Path Finding (MAPF) algorithms. To identify the defective parts of a multi-robot system at a glance, we propose a user-friendly state management interface for real-time monitoring. It is intuitive, lightweight, and minimally invasive. We demonstrate its effectiveness through a multi-agent pathfinding-based pattern formation experiment on a large-scale swarm robotics platform. The UI successfully monitored 70 robots, detected anomalies, and provided real-time feedback while minimizing its impact on system performance.**

I. Introduction

Recent advancements in multi-agent planning algorithms have enabled coordination of thousands of agents [1]–[3]. While these algorithms are primarily designed for robot deployment, the transition from simulation to real-world applications, particularly for large-scale teams remains challenging due to issues such as communication, localization, and system integration. Real robot platforms play a crucial role in bridging the gap between simulation and reality, revealing practical challenges of multi-robot coordination. While many platforms have been developed for small systems with a few dozen robots [4], our focus has shifted to much larger-scale systems, such as those involving 100 robots. At this scale, managing the system becomes significantly more complex, requiring substantial human effort to handle tasks like battery management, communication monitoring, and system reliability.

In this system, we present the development of a swarm robotics platform, termed *the 100-mouse system*, using Robot Operating System 2 (ROS2), highlighting the challenges of large-scale deployment. We focus on the human management aspects, including the development of a user interface (UI) and visualization tools, which have been largely overlooked in existing studies. As a proof of concept, we utilized the developed UI to implement unlabelled multi-agent pathfinding [5] with 70 robots, which solves pattern formation specified by user instructions. In what follows, we describe the system architecture, technical challenges, our state management system, and a platform demonstration.

II. Literature Review

There are many scalable MAPF algorithms, but only a few have been implemented on a real hardware platforms [6]–[8]. Even in these studies, the number of robots deployed remains at a few dozen, which does not fully capture scalability issues. Closely related to our work in terms of robotic platform are Robotarium [9], [10] and DOTS [11]. Robotarium is a remotely accessible swarm robotics platform that remains freely accessible to anyone. The platform features safety measures that combine simulation-based verification of user commands with collision avoidance [9], [10]. DOTS is a relatively new open-access platform for industrial swarm robotics platform. The robots used in this platform have higher performance than other platforms. They have high computational performance due to their on-board GPUs, high sensing performance, and real-world payload capability. However, both platforms are limited to 20 robots and do not support larger swarm robot systems. *The 100 mouse system* is distinguished by its scale of order 100. The uniqueness of the platform is the scalability of the system scale with inexpensive robots and ROS2, and a UI that detects anomalies that occur with a large number of robots.

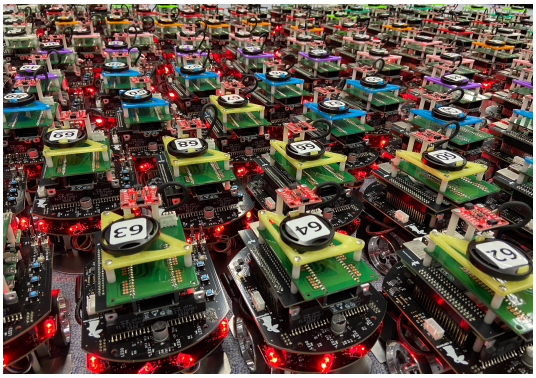

Fig. 1: 100-mouse system.

III. 100-mouse system

The 100-mouse system is a versatile swarm robotics platform based on ROS2, carefully designed to control large-scale robot fleets consisting of 100 Raspberry Pi Mouse robots (Fig. 1). This platform can be applied to various autonomous tasks, such as coordinating robots to form a specified formation. ROS2 was chosen for its simplicity in distributed communication, enabling the simultaneous control of multiple robots. Its flexibility in program modification is facilitated by node combinations.

A. System design and architecture

The 100-mouse system consists of mobile robots, a central server, and a positioning system (Fig.2). The ROS2 node configuration is broadly divided into two components: the central server and the mobile robots (Fig.3). The central

[1]National Institute of Advanced Industrial Science and Technology, Japan. [2]Institute of Science Tokyo, Japan. [3]Waseda University, Japan. [4]University of Cambridge, United Kingdom. {matsumoto.ryusei, yamamoto.shota, y-sasaki, okumura.k}@aist.go.jp

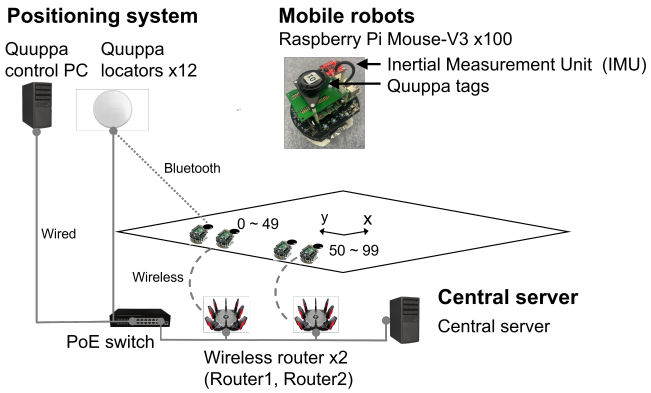

Fig. 2: System hardware design.

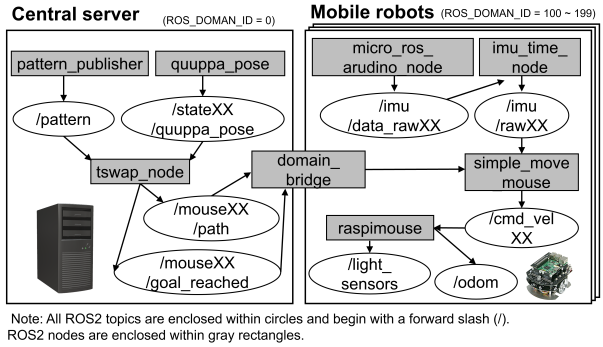

Fig. 3: ROS2 software design.

server aggregates the location data from the positioning system and computes the routes for all robots with multi-robot path planning algorithms. This paper takes the TSWAP algorithm [5] as an example of such a planner, a scalable unlabelled MAPF algorithm with completeness guarantees. By replacing the ROS2 node implementing this algorithm with another, different MAPF algorithms can be easily integrated into the system. Each robot handles posture estimation via its onboard Inertial Measurement Unit (IMU), motion generation based on the TSWAP instructions, and motor control.

a) Mobile robots: The mobile robot uses a Raspberry Pi Mouse V3 running ROS 2 Humble on Ubuntu 22.04. It is equipped with an ICM-20948-based IMU for attitude estimation and a Quuppa QT-1 tag for localization. Each robot is assigned a unique ROS_DOMAIN_ID within the range of 100 to 199, ensuring independent communication. The robots communicate only with the central server not with each other. To reduce communication overhead when managing up to 100 robots, each robot is connected to two separate Wireless Local Area Networks (Fig.2). The robot's posture (`/imu/rawXX`) is published as a quaternion, which is derived from the Digital Motion Processor in the IMU (`/imu/data_rawXX`) and combined with the initial posture data. The simple_move_mouse node generates velocity commands (`/cmd_velXX`) to control the robot's movement in up, down, left, and right directions. These commands are based on the posture (`/imu/rawXX`) and movement command (`/mouseXX/path`) (Fig.3 Mobile robots). Rotational motion is controlled proportionally, while translational motion is executed at a constant speed of $0.25\,\mathrm{m/s}$.

b) Positioning system: Our positioning scheme is based on the Quuppa Intelligent Locating System, a high-precision Bluetooth low-energy-based system. It consists of tags transmitting radio signals, locators receiving them, and software managing the system. In *the 100-mouse system* setup, 12 locators were installed on the ceiling to track the positions of 100 tags at $10\,\mathrm{Hz}$ within an area of approximately $10\,\mathrm{m} \times 20\,\mathrm{m}$. The location data for all 100 tags, as calculated by the Quuppa control software, was transmitted to a central server via UDP communication.

c) Central server: An Ubuntu 22.04 PC with ROS2 Humble served as the central server, aggregating location data from each robot and computing routes. The server runs ROS2 nodes including quuppa_pose, pattern_publisher, and tswap_node (Fig.3 Central server). The quuppa_pose node publishes location data as a ROS2 topic (`/stateXX/quuppa_pose`) received from the Quuppa control software. The pattern_publisher node publishes the robots' goal points (/pattern) defined in a configuration file. The tswap_node invokes the C++ TSWAP algorithm [5] and publishes movement command (`/mouseXX/path`) for all robots, as well as goal state topic (`/mouseXX/goal_reached`) each time a robot reaches its goal point. Although the central server uses a different ROS_DOMAIN_ID (set to 0), it runs domain_bridge to communicate minimally with the robots on the topics (`/mouseXX/path, /mouseXX/goal_reached`).

IV. Proposal for System State Management UI

It is difficult to ensure that all technical components are executed without anomalies in a multi-agent system managing numerous robots. Such anomalies include sensor errors, communication delays, and actuator response lags that can occur independently in each robot. Monitoring for these unwanted events is critical. Otherwise, accumulating these small errors will lead to degraded system performance or, worse, catastrophic failure of the entire system. However, managing both the individual robots' states and the overall system is non-trivial, especially with 100 robots, as it is inherently information overflow for human users. An organized UI is therefore essential for the deployment of large-scale multi-robot systems; In the following, we first characterize the requirements for such UI systems and then present our solution.

1) The visual status should be easily interpretable.
2) The system should be lightweight and capable of displaying real-time information yet capable of handling large multi-robot teams.
3) The operation of the multi-agent system should not be affected.
4) Tracking of the robot's movements and status should be possible.

A. Implementation of state management UI

To satisfy the above requirements, the proposed state management UI employs the following.

1) **Intuitive visual status monitoring**
   Individual robot status is monitored through table, while a top-down view of the multi-agent system is provided using a positioning system and an overhead camera. The table and positioning system view are linked, enabling a simultaneous understanding of each robot and the system's overall state (Fig.4).
2) **Efficient real-time display**
   The tables and top-down view are updated every few seconds. To minimize communication delays, the UI server is on the same ROS2 network as the central server (Fig.4). when communication with robots in different ROS2 networks is required, data from multiple topics are aggregated into a single topic (`/mouse1XX/alive_judge_publisher`) on the robot side, reducing both communication load and the need for cross-network communication.
3) **Minimize interference with multi-agent systems**
   To maintain system performance, a Quality of Service policy controls communication reliability through several parameters and is configured for communication between the multi-agent system and the state management UI. The History policy is set to "Keep Last" and the Reliability policy to "Best Effort", ensuring that real-time status display does not disrupt the system operation.
4) **Robot status and movement tracking**
   The operational status is inferred from data across various topics. For instance, by comparing the velocity command (`/cmd_velXX`) and the actual movement velocity (`/odom`), it can be determined whether the velocity command is correctly transmitted to the motor. Inferences are also made regarding potential collisions and posture accuracy based on relevant data.

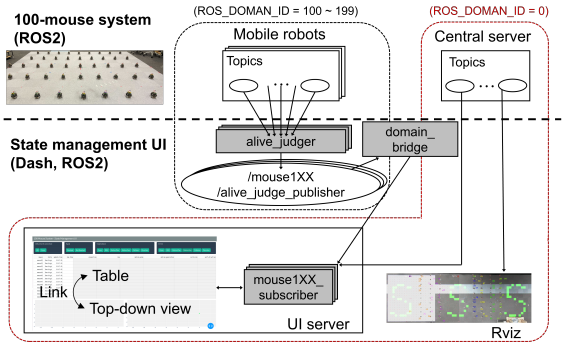

Fig. 4: State management UI architecture.

The state management UI was implemented using Dash (https://github.com/plotly/dash), a Python data visualization framework integrated with ROS2. Rviz, a 3D visualizer associated with ROS2, displayed the robots' positional data via an overhead camera. Each mobile robot included an alive_judger node aggregating multiple internal topics and publishing them as a single topic (`/mouse1XX/alive_judge_publisher`). The mouse1XX_subscriber node receives topics from both the robot and server, which are reflected in the UI.

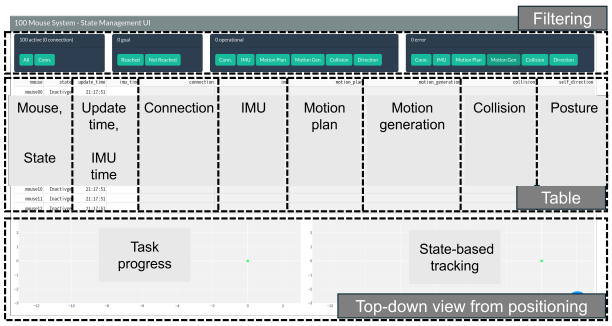

Fig. 5: State management UI layout and appearance.

B. Display of Status Information for Each Robot

The state management UI table presents a row of records for each robot, organizing its state into six categories as shown in Figure 5: Connection, IMU, Motion plan, Motion generation, Collision, and Posture. It also displays the record's update time and the internal robot time. Each category's normality or abnormality is determined by the presence of the relevant topics and the inferred values from these topics. The table's structure is as follows:

- **Mouse and State** section displays the ID and status of each robot, with possible statuses including Inactive, Operational, Error, and Goal. Robots that have not communicated with the system are classified as Inactive. Robots that have subscribed the goal state topic (`/mouseXX/goal_reached`) are marked as Goal. If abnormalities are detected in any columns, the robot's status is set to Error; otherwise, it is Operational.
- **Update time and IMU time** section displays the update time for each record and the robot's last communication time (internal robot time), respectively.
- **Connection** section displays the results of ping tests.
- **IMU** section displays the presence of the topic (`/imu/data_rawXX`) related to its operation.
- **Motion plan** section displays the presence of the topics (`/imu/rawXX`, `/mouseXX/path`) related to the simple_move_mouse node's subscription.
- **Motion generation** section displays the availability of the topics (`/cmd_velXX` and `/odom`) related to the velocity command and odometry. If the velocity command does not accurately affect the odometry, an error message is displayed in the corresponding cell.
- **Collision** section displays the collision occurrences. The robots are considered to be operating normally if the values detected by their four light sensors are below a threshold, averaged over 20 topic receipts.
- **Posture** section displays posture estimation based on the robot's movement command and localization. When a movement command (`/mouseXX/path`) is published over time with waypoints path_1, path_2, ..., the robot is considered to be operating normally if the angle between the vector of path_1 and the vector from the starting point of path_1 to robot's position prior to the publication of path_2 is within ± 90 degrees.

C. Task progress and anomalies via Positioning

The bottom part of the UI tracks progress and anomalies in the system with a top-down view. The locations of all robots are displayed based on their positioning data, with robots that have reached their goals marked (Fig. 5 and Fig. 7, bottom-left). This provides an overview of each robot's task progress. A filtering function allows the display of only robots in a specific state, enabling users to track their behavior visually (Fig. 5 and Fig. 7, bottom-right). An arrow indicating the robot's travel direction is shown on the current movement command (`/mouseXX/path`), allowing real-time visualization of whether the robot is following the designated path.

D. Global view visualization with overhead camera

Apart from Fig. 5, we further display the real-time positions of the mobile robots, in Fig. 6. Positioning data (numerical values), goal points (green squares), movement trajectories (line graph), and goal decision points (red squares) are also shown. This provides a top-down image of the final pattern formation, allowing visual evaluation of how closely the actual configuration matches the intended goal.

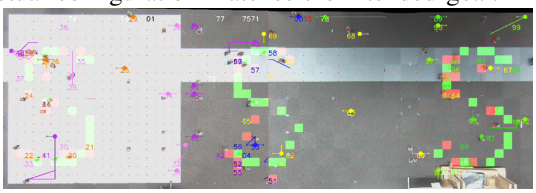

Fig. 6: Overhead camera view.

V. Experiment

A pattern formation experiment was conducted with *the 100-mouse system*, targeting the pattern "SSS." Seventy robots were deployed, and their behavior was continuously monitored through the developed state management UI.

A. Performance evaluation of the state management UI

Figure 7 presents the state management UI display during the operation of 70 robots. The performance of the UI in managing the system is described below.

1) **Intuitive visual status monitoring**
   The status of each robot was conveyed through color-coded cells indicating normal and abnormal conditions. Each row in the table represented a robot. For normally operating robots, all cells were green (Fig.7 mouse51). When a robot reached its goal, the row was turned orange (Fig.7 mouse52). Anomalies were indicated by red cells, with the issue described in the text (Fig.7 mouse50).
2) **Efficient real-time display**
   The table was updated every 3 seconds, and the top-down view was refreshed every second. For robots with established communication (Fig.7 mouse50-56), the internal clock (IMU time) was synchronized within approximately 3 seconds, confirming correct updates.
3) **Minimize interference with multi-agent systems**
   The 100-mouse system operated effectively with the state management UI active, showing no significant impact on performance impact.
4) **Robot status and movement tracking**
   Robots identified as in collision on the UI were confirmed to be in collision. However, no clear distinction was observed in the posture inference display between correct and misaligned initial postures. Positioning error may have influenced inference by positioning, indicating the need for a more accurate positioning system or alternative methods.

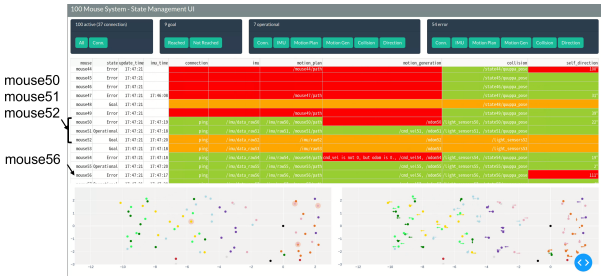

Fig. 7: State management UI in operation.

B. Case studies on abnormality detection

The following events represent anomalies detected by the state management UI in *the 100-mouse system*.

1) Detection of communication instability: A communication failure occurred simultaneously with robot IDs 50 to 99, connected to Router 2, within 10 seconds. During this period, robots with stable communication continued to move, whereas those with disrupted communication remained stationary (Fig.8A, Fig.2). After losing communication, the robots gradually reconnected. However, as more robots became available, the connection to either Router 1 or Router 2 was lost again.

2) Detection of robot collisions: Collisions were detected between robots in operation and those that had already reached the goal (Fig.8B mouse84). Although the TSWAP algorithm in *the 100-mouse system* theoretically prevents collisions, the results suggest that the system design must account for positioning errors and communication delays in real-world scenarios.

3) Detection of IMU malfunctions: Some robots were detected communicating, but their IMUs were deactivated. When the IMU is deactivated, no subsequent movement commands are published, resulting in these robots remaining stationary for several seconds (Fig.8C mouse81, 92, 94).

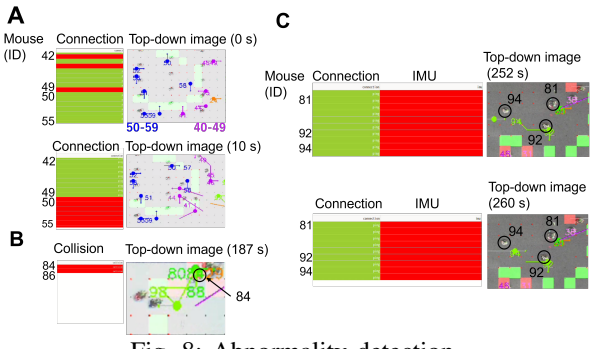

Fig. 8: Abnormality detection.

VI. Conclusion

This paper presents *the 100-mouse system*, an actual multi-robot platform for testing MAPF, and its state management user interface designed to visualize both individual robot status and the overall state of a multi-agent system. The UI provides real-time, lightweight updates with minimal impact on performance. In the 100-mouse system, the UI successfully identified critical issues such as communication instability, collisions, and IMU malfunctions, highlighting key operational challenges. This tool will bridge the gap between simulation and real-world deployment in multi-agent path planning, providing valuable insights to improve future system performance.

References

[1] Keisuke Okumura, Manao Machida, Xavier Défago, and Yasumasa Tamura. Priority inheritance with backtracking for iterative multi-agent path finding. *Artificial Intelligence*, 310:103752, September 2022.

[2] Keisuke Okumura. Improving lacam for scalable eventually optimal multi-agent pathfinding. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*, IJCAI '23, 2023.

[3] He Jiang, Yulun Zhang, Rishi Veerapaneni, and Jiaoyang Li. Scaling lifelong multi-agent path finding to more realistic settings: Research challenges and opportunities, 2024.

[4] Jan Blumenkamp, Ajay Shankar, Matteo Bettini, Joshua Bird, and Amanda Prorok. The cambridge robomaster: An agile multi-robot research platform, 2024.

[5] Keisuke Okumura and Xavier Défago. Solving simultaneous target assignment and path planning efficiently with time-independent execution. *Artificial Intelligence*, 103946, 2023.

[6] Keisuke Okumura, François Bonnet, Yasumasa Tamura, and Xavier Défago. Offline time-independent multiagent path planning. *IEEE Transactions on Robotics*, 39(4):2720–2737, 2023.

[7] Keisuke Okumura and Xavier Défago. Quick multi-robot motion planning by combining sampling and search, 2023.

[8] Jungwon Park, Junha Kim, Inkyu Jang, and H. Jin Kim. Efficient multi-agent trajectory planning with feasibility guarantee using relative bernstein polynomial, 2020.

[9] Sean Wilson, Paul Glotfelter, Li Wang, Siddharth Mayya, Gennaro Notomista, Mark Mote, and Magnus Egerstedt. The robotarium: Globally impactful opportunities, challenges, and lessons learned in remote-access, distributed control of multirobot systems. *IEEE Control Systems Magazine*, 40(1):26–44, 2020.

[10] Daniel Pickem, Paul Glotfelter, Li Wang, Mark Mote, Aaron Ames, Eric Feron, and Magnus Egerstedt. The robotarium: A remotely accessible swarm robotics research testbed. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 1699–1706, 2017.

[11] Simon Jones, Emma Milner, Mahesh Sooriyabandara, and Sabine Hauert. Dots: An open testbed for industrial swarm robotic solutions, 2022.