2026 IEEE/SICE International Symposium on System Integration (SII)
January 11-14, 2026. Cancún, México

Development of a Software Platform for Software-Defined Robot

Nobuhiko MIYAMOTO[1] and Noriaki ANDO[1] , Yasuharu KUNII[2]

Abstract— **In this paper, we propose a software platform to realize a Software-Defined Robot (SDRobot) for a large number of mobile robots performing exploration and base construction. To develop a platform that adapts to unknown environments by utilizing the redundancy of multi-robot systems and changing functionality through software, we extended the robot middleware OpenRTM-aist with a dynamic module allocation function and an automatic recovery function. The extended functions provide the foundation for a system that adapts to the environment by changing the software configuration depending on the status of resources such as control computers and the state of communication between robots. In this paper, we show an example of constructing a system with multiple RED mobile robots and confirmed the operation of the extended functions.**

I. Introduction

The lunar surface is a harsh environment with extreme temperature differences ranging from -170°C to +110°C, but the underground space of lunar lava tubes has a stable temperature and may be suitable for constructing bases. The research project under Moonshot R&D Program Goal 3, " Intelligent Multi Agents for Exploration and Settlement in Unknown and Unexplored Areas ", aims to realize the exploration of lunar lava tubes and the construction of bases by building a system in which a large number of small robots cooperate and evolve. Robots operating in lunar lava tubes are expected to have limited wireless communication quality and computer resources due to constraints such as power supply. Furthermore, it is difficult for humans to respond on-site when failures or losses occur [1][2]. Therefore, it is considered necessary to use the redundancy of multi-robot systems and establish mechanisms that can flexibly respond to environments and problems.

What is an SDRobot (Software-Defined Robot)? Several definitions have been proposed for Software-Defined Robot or Software-Defined Robotics, but no clear definition exists. Software-defined technologies such as SDR (Software Defined Radio) and SDN (Software Defined Network) achieve flexibility by controlling multifunctional hardware with software. In this paper, we define SDRobot as a mechanism for robots whose behavior can be flexibly changed through software reconfiguration and automatic updates. When realizing fault tolerance and self-recovery mechanisms, robot middleware such as ROS is sometimes used. Middleware provides functionality to ensure availability, allowing developers to focus on developing software components [3][4][5][6].

[1]Intelligent Systems Research Institute(ISRI), National Institute of Advanced Industrial Science and Technology (AIST), Central2, 1-1-1 Umezono, Tsukuba, Ibaraki 305-8568 Japan, n-miyamoto@aist.go.jp
[2]Department of Electrical, Electronic, and Communication Engineering, Faculty of Science and Engineering, Chuo University

In this paper, we propose a software platform to realize SDRobot (Software-Defined Robot), a mechanism that flexibly adapts to environments through software reconfiguration. We demonstrate execution examples in which redundant robot systems dynamically change their configuration according to the environment.

II. Overview of the Software Platform

The functions required of a software platform for SDRobot are as follows:

- A distributed system middleware that builds systems across multiple robots
- An automatic deployment function that enables dynamic launch of robot systems
- An error-handling mechanism that triggers recovery on detection of software, communication, or robot hardware anomalies
- A migration function that moves system functions to other robots
- A monitoring function that observes robot resources, communications, and physical positions

Based on these requirements, we developed a software platform by extending the robot middleware OpenRTM-aist [7] with the functions described below.

A. Basic Functions

The following three functions have been implemented as basic functions.

① Deployment Function

The deployment function provides a mechanism and interface to dynamically launch RT-Components[8][9]. RT-Components can be launched by dynamically loading modules such as dynamic link libraries, Python scripts, and Java class files. In addition, it provides a component deployment framework that can launch RTCs on Docker containers by specifying a Dockerfile.

② Migration Function

The migration function executes failover by switching to standby RT-Components when RT-Components transition to an error state or when a process terminates abnormally[10]. RT-Components transition between active, inactive, and error states according to the OMG RTC standard. With the component observer function, it is possible to switch to standby RT-Components when transitioning to an error state. Moreover, by extending the OpenRTM-aist manager function, failover can be executed when a process started by the manager terminates abnormally.

③ Monitoring Function

978-1-6654-5784-2/26/$31.00 ©2026 IEEE

The monitoring function launches an SNMP manager and SNMP agent, enabling monitoring of resource metrics such as CPU usage and memory usage via SNMP (Simple Network Management Protocol) communication[11]. Each robot extends OpenRTM-aist with active monitoring via polling and passive monitoring in which the SNMP agent sends traps to the SNMP manager when abnormalities occur. Robots actively monitor each other to share metric information and can receive traps when another robot experiences an anomaly.Based on information obtained from active monitoring or traps, the software component configuration can be changed.

B. Extended New Functions

Based on the basic functions, we implemented extended functions to realize SDRobot. Existing frameworks that combine ROS 2 with Kubernetes [5] primarily attempt recovery by restarting processes after crashes or timeouts. For distributed robot systems with constraints on resources and power, however, we also need mechanisms for dynamically changing inter-module connections, such as the following.

ⓐ **Dynamic Module Allocation Function** The dynamic module allocation function (Fig. 1) assigns software modules dynamically based on information such as load per robot, network conditions, and physical positions, when multiple robots are working in parallel. Specifically, CPU usage, memory usage, and estimated communication traffic between components of the module to be launched are used, together with the current information of each robot, to automatically allocate RT-Components to robots. Table I shows the constraints that each robot must satisfy for module allocation, and Table II shows the constraints that must be satisfied when communication occurs between components upon module launch.

Regarding the outline of the module allocation algorithm, one robot first acquires configuration information, including the RT components to be started, port connection information, parameter settings, and so on. The configuration information may be sent from an external server, or it may be selected from multiple configurations stored on the robot according to the environment. If there are standby modules that have already been started on robots, and those modules are included in the configuration, the system uses these standby modules.

When all RT components cannot be started on a single robot due to resource constraints, the system requests the startup of RT components on robots that satisfy the necessary conditions. When multiple robots satisfy the conditions, one of the following algorithms can be used:

- Check the information of robots in order from a list and select the first robot that satisfies the conditions.
- Check all robots in the list and select the robot that maximizes an objective function computed from multiple pieces of information.

Examples of possible allocations include:

- When all modules cannot be launched on a single robot due to insufficient memory, the system automatically finds a robot that (a) is physically close, (b) has good communication status, and (c) has sufficient memory, and requests it to launch the module.
- When it is difficult to launch a module with high CPU usage on a robot with insufficient power, the system automatically finds a robot that (a) is physically close, (b) has good communication status, and (d) has sufficient remaining battery, and requests it to launch the module.

TABLE I: Constraint conditions for each robot

Information	Constraint condition
CPU usage	Below threshold
Free memory	Above threshold
Battery level	Above threshold

TABLE II: Constraint conditions between robots

Information	Constraint condition
Signal strength (RSSI)	Above threshold
Transmission jitter	Below threshold
Distance between robots	Below threshold

Estimates of CPU usage, memory usage, and communication traffic between components for the modules to be launched must be measured in advance. For this purpose, we added a function to the OpenRTM-aist manager to collect and save to file the CPU usage and memory usage of its own process, as well as the data sent and received by RT-Component data ports. Thus, measurement can be completed once the system is constructed and executed from RT-Components. Furthermore, the automatic allocation process can be extended with plugins.

ⓑ **Automatic Recovery Function** The automatic recovery function (Fig. 2) switches failover processing depending on the situation when a problem occurs in the robot system, and is implemented by extending the migration function.

Specifically, on the robot where a module is launched, module switching is executed according to the conditions shown in Table III. For failures that can be resolved by restarting the module, the standby module within the same robot is switched in, or the module is restarted. If the problem is not resolved, the system switches to the standby module on another robot.

Switching conditions can be changed via configuration files, and more complex conditions can be defined by extending plugins.

III. System Demonstration

In this chapter, we present an example of constructing a robot system using multiple small mobile robots RED[12], as well as execution examples of the automatic module allocation function and the automatic recovery function. RED is controlled by the onboard Raspberry Pi 4 Model B. The Raspberry Pis are connected to a Wi-Fi access point. The devices and settings related to the wireless LAN are shown in Table IV.

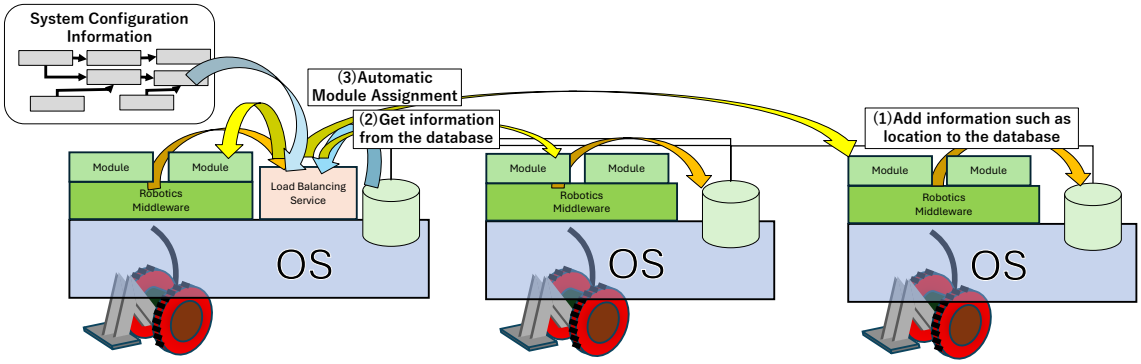


Fig. 1: Overview of the dynamic module allocation function

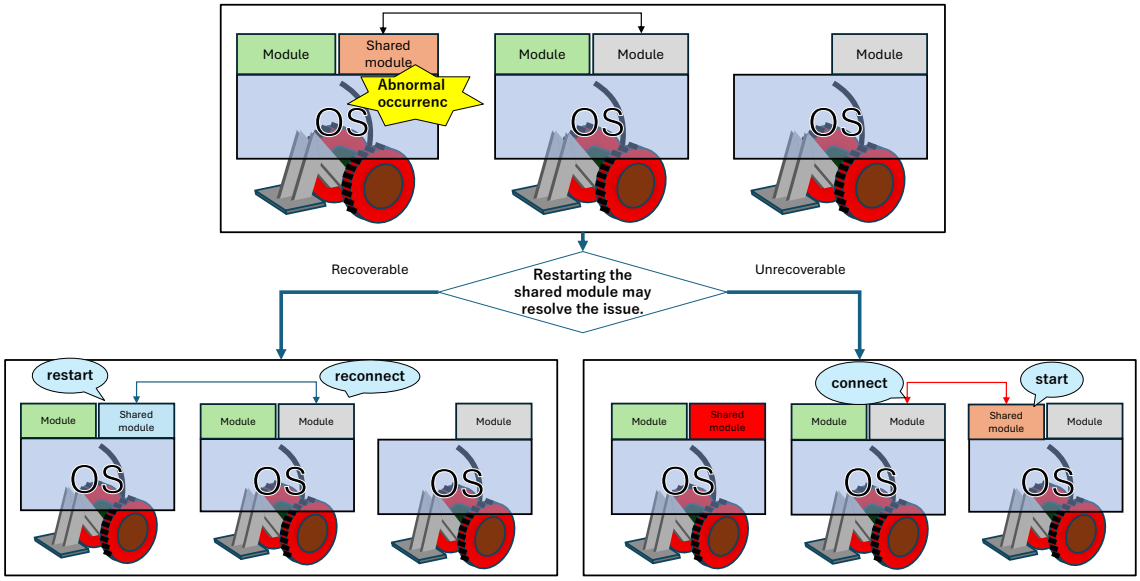


Fig. 2: Overview of the automatic recovery function

TABLE III: Switching conditions and processing

Cause	Switching condition	Switching process
Memory shortage	Free memory below threshold	If a standby module is running on the same robot, terminate the faulty module and switch to the standby. Otherwise, attempt restart. If still unresolved, switch to a standby on another robot; if none, launch the module on another robot.
Abnormally high CPU usage	CPU usage above threshold	
Component transition to error state	Detect hardware failure or disconnection Detect abnormal robot behavior Detect emergency stop	
Low battery	Battery level below threshold	Switch to a standby module on another robot. If none, launch the module on another robot.
Abnormally high CPU usage	CPU usage above threshold	
Communication failure	RSSI below threshold Delay above threshold Jitter above threshold Packet loss rate above threshold	

TABLE IV: Wireless LAN environment

Item	Device/Setting used
Wi-Fi access point	Buffalo AirStation Pro WAPM-1750D
Wi-Fi adapter	cypress cyw43455
Frequency	2.457GHz
Protocol	IEEE 802.11n

A. System Overview

An overview of the system is shown in the figure 3. This system uses a perspective projection camera and a fisheye camera mounted on RED, and is configured to stop under the following conditions, while otherwise continuing to move:

- When the camera detects a specific object and determines it to be dangerous
- When RED moves outside the designated area

When the system configuration information is loaded into the OpenRTM-aist manager, the automatic module allocation function attempts to assign RT-Components to Robot-1 as shown in Fig. 3-(1). However, due to memory capacity limitations, it requests Robot-2 and Robot-3 to launch RT-Components. On Robot-1, the fisheye camera component (OmnidirectionalCameraRTC1) and anomaly detection component (AnomalyDetectionRTC1) are launched, on Robot-2, the camera component (CameraRTC2) and motion control component (REDControllerRTC2) are launched, and on Robot-3, the position estimation component (PositionEstimationRTC3) is launched. The system is then formed by exchanging data among these components.

After executing the automatic allocation and allowing a certain amount of time to pass, as shown in Fig. 3-(2), one RED is covered with an electromagnetic wave shield mesh to reproduce a weak electric field environment. The system recognizes the weak field environment as an abnormal condition, and the automatic recovery function migrates the fisheye camera component and anomaly detection component originally running on Robot-1 to Robot-3, which has sufficient resources, thereby maintaining overall system operation.

The free memory, RSSI, and RT-Component states for each robot are shown in Figs. 4, 5, and 6. For functional verification, each component allocates a 500 MB memory region on startup and frees it on termination. The orange dashed lines in Figs. 4–6 indicate when dynamic module assignment was executed; the purple dash–dot lines indicate when automatic recovery was executed. Constraint settings for dynamic assignment are listed in Table V, and switching conditions for automatic recovery in Table VI.

TABLE V: Constraint condition settings

Information	Constraint condition
Free memory capacity	500MB or less

We present time-series plots of the timings of RT-Component activation and deactivation, the free memory of each robot, and the RSSI of Robot-1. From Fig. 4c, Fig. 4d, Fig. 5b, Fig. 5c, and Fig. 6b, it can be confirmed that five RT-Components are launched by the module allocation

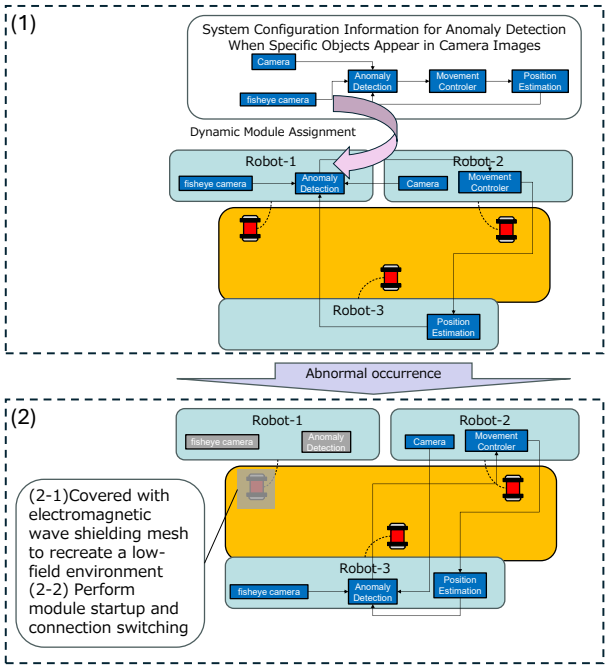


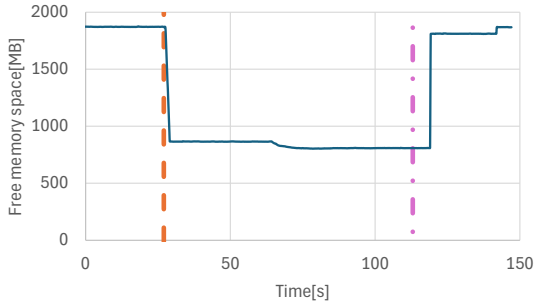
Fig. 3: Example of system construction

TABLE VI: Switching condition settings

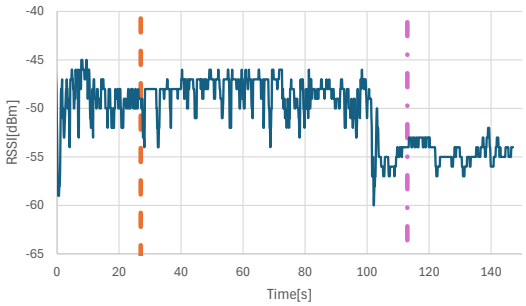
Cause	Switching condition
Memory shortage	Free memory capacity of 200MB or less
Communication failure	RSSI of -52.5 dBm or less sustained for 6 seconds

function at around 27 seconds from the start. It takes about 16 seconds to activate all RT-Components. This is because each component allocates 500 MB at startup and performs initialization during activation, without processing in the activation callback, startup takes about one second.

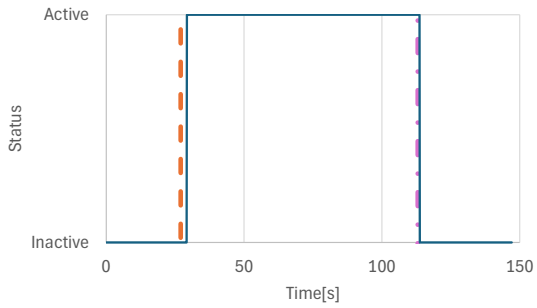
From Fig. 4b, the RSSI of Robot-1 decreases from approximately -47 dBm to approximately -55 dBm at around 107 seconds. The weak-field state is detected as an anomaly, and at around 113 seconds the RT-Components running on Robot-1 terminate, while the corresponding RT-Components launch on Robot-3. Although launching three RT-Components on Robot-3 causes its free memory to exceed the constraint of the automatic module allocation function, it does not violate the switching condition of the automatic recovery function; thus, automatic recovery completes by migrating RT-Components to Robot-3. Automatic recovery takes about 11 seconds, primarily because the startup request is issued via the load-balancing service on Robot-1 under the weak-field condition. By satisfying the constraints of the automatic module allocation function, high reliability can be ensured. When troubles such as communication failures occur, migrating functions to maintain system operation may reduce reliability, but availability is secured. These results show that, compared with approaches that only detect crashed/unresponsive processes and restart them, our platform enables more environment-adaptive automatic module assignment and recovery.



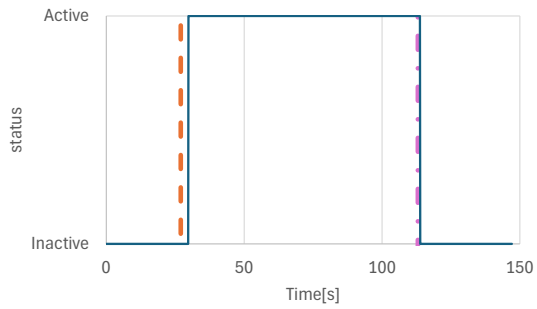
(a) Free memory capacity of Robot-1



(b) WiFi status of Robot-1

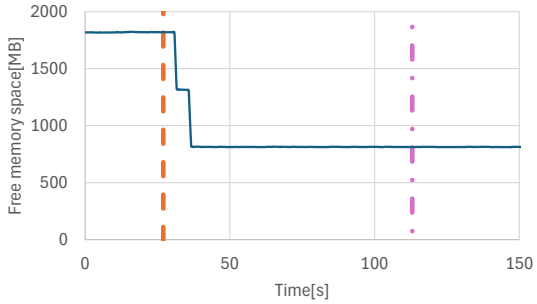


(c) Status of OmnidirectionalCameraRTC1

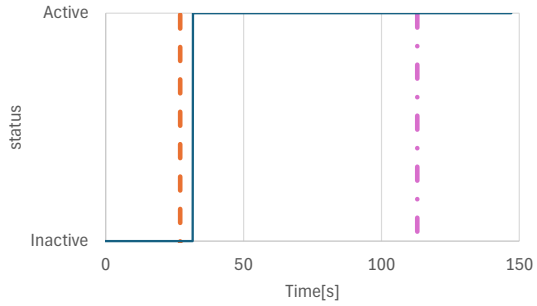


(d) Status of AnomalyDetectionRTC1

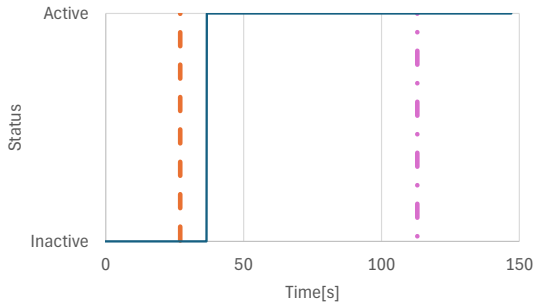
Fig. 4: Status of Robot-1



(a) Free memory capacity of Robot-2



(b) Status of CameraRTC2



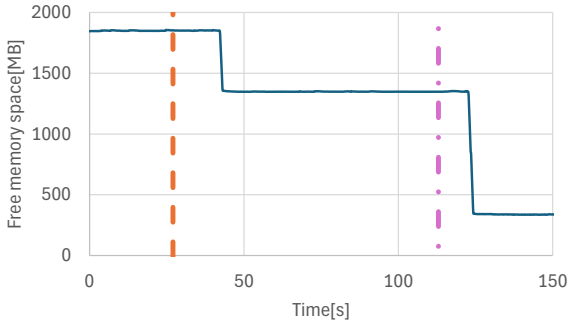
(c) Status of REDControllerRTC2

Fig. 5: Status of Robot-2

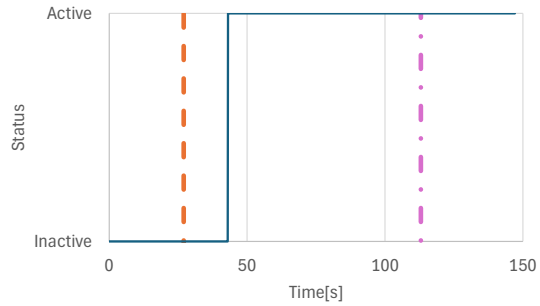
IV. Conclusion

We proposed a software platform to realize Software-Defined Robots, in which redundant robot systems flexibly adapt to the environment by reconfiguring software, and implemented an automatic module allocation function and an automatic recovery function based on OpenRTM-aist. Assuming exploration and base construction by large robot teams, we demonstrated a robot system constructed with multiple RED mobile robots and confirmed the operation of the automatic module allocation and automatic recovery functions. While reliability is emphasized at system startup through automatic module allocation, when problems occur the system switches to availability-oriented module placement, enabling environmental adaptation and continuous operation.

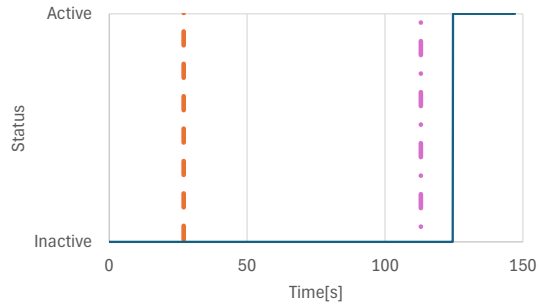
In our test system, recovery took longer because the robot running the load-balancing service was placed in the weak-field environment. Therefore, we consider it necessary to



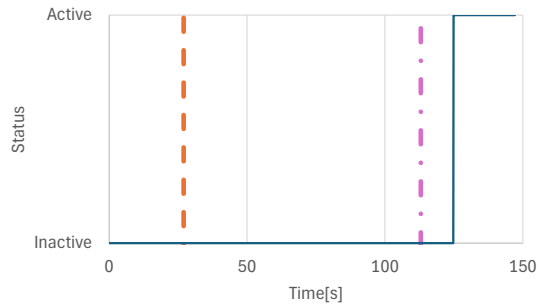
(a) Free memory capacity of Robot-3



(b) Status of PositionEstimationRTC3



(c) Status of OmnidirectionalCameraRTC3



(d) Status of AnomalyDetectionRTC3

Fig. 6: Status of Robot-3

extend functionality to migrate the load-balancing service to other robots. As future work, assuming the deployment of RED in lunar lava tubes, we aim to apply the software platform to embedded real-time operating systems.

ACKNOWLEDGMENT

This research was supported by JST Moonshot R&D Program [JPMJMS2238].

REFERENCES

[1] C. Valcázar, et al., "Compact Omnidirectional Antenna for Lunar Lava Tube Robots", *Proceedings of the 42nd ESA Antenna Workshop*, 2024.

[2] H. Kalita and J. Thangavelautham, "Communication and Power Transfer for Cooperative Multi-Robot Exploration in Lunar Lava Tubes", *Sensors*, vol. 21, no. 5, pp. 1514, 2021.

[3] D. Kirchner, S. Niemczyk, and K. Geihs, "RoSHA: A Multi-Robot Self-Healing Architecture", in *RoboCup 2013: Robot World Cup XVII*, Lecture Notes in Artificial Intelligence, Springer, vol. 8371, pp. 329–340, 2014, .

[4] M. Li, K. Su, W. Liu, L. Cui, and G. Liu, "ALLIANCE-ROS: A Software Architecture on ROS for Fault-Tolerant Cooperative Multi-Robot Systems", in *PRICAI 2016: Trends in Artificial Intelligence*, Lecture Notes in Computer Science, Springer, vol. 9810, pp. 499–511, 2016.

[5] J. Zhang, X. Yu, and T. Westerlund, "Enhancing the Resilience of ROS 2-Based Multi-Robot Systems with Kubernetes", *Sensors*, vol. 25, no. 2, p. 451, 2025.

[6] M. Kim, S. Garg, J. Gonzalez, and K. Goldberg, "FogROS2-FT: Fault-Tolerant Cloud Robotics", in *Proc. The 2022 IEEE/RSJ International Conference on Intelligent Robots and Systems(IROS2022)*, IEEE, pp. 11553–11560, 2022.

[7] N. Ando, T. Suehiro, K. Kitagaki, T. Kotoku, M. S.. Couceiro, "RT-Middleware: Distributed Component Middleware for RT(Robot Technology)", *Proc. The 2005 IEEE/RSJ International Conference on Intelligent Robots and Systems(IROS2005)*, pp.3555–3560, 2005.

[8] N. Ando and N. Miyamoto, "Component Deployment Framework Based on Container Environment," *Proceedings of the 24th SICE System Integration Division Annual Conference (SI2023)*, pp. 3224–3226, 2023 (in Japanese).

[9] N. Miyamoto, N. Ando, and Y. Kunii, "Development and Performance Evaluation of a Deployment Automation Framework by Containerization of Robot Software Components," *Proceedings of the 42nd Annual Conference of the Robotics Society of Japan (RSJ2024)*, 3C2-06, 2024 (in Japanese).

[10] N. Miyamoto and Y. Ando, "Proposal of a Framework for Software Component Redundancy in Distributed Robotic Systems," *Proceedings of the Robotics and Mechatronics Conference 2024 (ROBOMECH2024)*, 2P1-H01, 2024 (in Japanese).

[11] N. Miyamoto, Y. Ando, and Y. Kunii, "Proposal of an Automatic Software Component Deployment Framework for Improving Task Efficiency in Distributed Robotic Systems," *Proceedings of the Robotics and Mechatronics Conference 2025 (ROBOMECH2025)*, 2A1-G02, 2025 (in Japanese).

[12] T. Sato, K. Sakamoto, T. Maeda, and Y. Kunii, "Exploration System for Distributed Swarm Robots Using Probabilistic Action Decisions", in *Distributed Autonomous Robotic Systems (DARS 2022)*, Springer Proceedings in Advanced Robotics, Springer, pp.453–465, 2024.