Design, Implementation, and Evaluation of an Open-RMF-Based Multi-Robot Middleware Framework on TurtleBot3 Robots

Jerin Peter, *Graduate Student, UCR,* Dr. Konstantinos Karydis, *Faculty Advisor, UCR*

Abstract

This project presents the design, implementation, and evaluation of a multi-robot middleware framework based on the Robotics Middleware Framework (Open-RMF), which was deployed and tested on two TurtleBot3 (TB3) robots. By leveraging the Cartographer library for simultaneous localization and mapping (SLAM), converting the generated occupancy grid to an RMF-compatible navigation graph, and employing the Free Fleet reference implementation, a fully operational multi-robot scheduling and navigation system was demonstrated in both simulation and on physical hardware. The developed framework supports task auctioning, conflict-free resource scheduling, dynamic path negotiation, and real-time fleet management. This report provides detailed instructions for environment setup, RMF-Web integration, and scenario creation, presented alongside performance metrics collected from scaled experiments. Flowcharts, architecture diagrams, and deployment snapshots are used to elucidate system components and data flow. The systems capability to coordinate two TB3 robots for complex patrol and delivery tasks under changing conditions is evaluated, highlighting its strengths in coordination, modularity, and extensibility, while also identifying areas for future improvement.

Index Terms

Multi-Robot Systems, Middleware, Open-RMF, TurtleBot3, Cartographer, ROS 2, Free Fleet, Multi-Agent Pathfinding, Robot Scheduling, Navigation Graphs, RMF-Web.

I. Introduction

MULTI-robot systems (MRS) offer significant advantages over single-robot architectures by enabling collaborative task execution, enhanced operational throughput, and improved system-wide fault tolerance [1], [2], [3]. Foundational work in the field has addressed core challenges like decentralized control, task allocation, and formation keeping [47]. However, the practical integration of heterogeneous robot fleets, often sourced from different vendors, presents considerable challenges in inter-robot coordination, unified path planning, and robust communication [4], [5], [6]. Effective middleware is critical for abstracting hardware differences and providing common services for tasking and navigation [10]. The Robotics Middleware Framework (OpenRMF) emerges as a robust solution to these issues by abstracting fleet management, resource negotiation, and traffic scheduling into a set of high-level, standardized services. This abstraction facilitates seamless interoperability among diverse robotic platforms [8].

This report details the end-to-end design, implementation, and validation of an OpenRMF based middleware on a fleet of two TurtleBot3 (TB3) robots. The methodology involved using Cartographer for mapping, converting the resulting maps into RMF navigation graphs, and employing the Free Fleet (`rmf_demos`) implementation for multirobot scheduling and control. The hardware demonstration validates the framework's capability to coordinate simultaneous and potentially conflicting tasks (e.g., patrol and delivery routes) under dynamic environmental conditions.

The key contributions of this work are as follows:

- The development of a complete mapping pipeline using Cartographer, including a method for converting its output into RMF-compliant navigation graphs.
- The integration of OpenRMFs Free Fleet demonstration packages with physical TB3 hardware through the implementation of a custom fleet adapter.
- The deployment of the RMF–Web visualization suite for real-time fleet management and operational awareness.
- A systematic performance evaluation based on metrics such as task completion time, resource utilization, and conflict resolution efficiency.

This report is structured as follows. Section II reviews the foundational literature and existing technologies pertinent to multi-robot middleware. Section III outlines the primary goal and specific objectives of the project. Section IV details the system setup and configuration. Section V describes the project's architecture and core methodologies. Section VI presents the hardware deployment, task definitions, and performance evaluation. Section VII discusses the results and identifies system limitations and finally concludes the report and suggests avenues for future research and enhancement.

Jerin Peter is with the Department of Electrical and Engineering (Robotics), University of California, Riverside, CA, 92507 USA. (email: jpete085@ucr.edu)
Prof. Konstantinos Karydis is with the Department of Electrical and Computer Engineering, University of California, Riverside, CA, 92507 USA. (e-mail: kkarydis@ucr.edu).

This work was supported in part by the UCR ARCS Lab under Professor Konstantinos Karydis. Date of report: June 12, 2025.

II. State of the Art

This section reviews the foundational literature and existing technologies pertinent to multi-robot middleware and coordination.

A. MultiRobot Systems and Middleware

The challenges of coordinating multiple robots, particularly in shared environments, have led to the development of specialized middleware solutions. Key problems in MRS that middleware aims to solve include inter-robot coordination, path planning in dynamic environments, and the arbitration of common resources such as narrow corridors, doorways, and elevators [9], [10]. Efficiently assigning tasks to the most suitable robots is another critical challenge, with various strategies explored in literature [11]. Foundational research has explored distributed algorithms for cooperation [47] and market-based mechanisms for efficient task allocation [32], which remain central concepts in modern fleet managers.

a) ROS 1 and ROS 2: Among the most influential platforms in robotics is the Robot Operating System (ROS), which has evolved significantly to better support MRS. ROS has become the de facto standard for single-robot applications, providing a rich ecosystem of software libraries, development tools, and simulation environments like Gazebo and RViz [12]. However, the custom TCP–based communication middleware of ROS 1 was found to be less suitable for complex multirobot deployments. Its successor, ROS 2, was built atop the Data Distribution Service (DDS) standard, introducing features such as realtime communication capabilities, enhanced security protocols, component lifecycle management, and fine-grained Quality of Service (QoS) controls. These features make ROS 2 a more robust foundation for developing modern MRS applications [13], [14]. The use of DDS as a communication backend intrinsically addresses many of the service discovery and data transport challenges inherent in distributed systems [15], though performance can vary between specific DDS implementations [16]. For large-scale or geographically distributed deployments, protocols like Zenoh can be layered on top to further optimize data routing over DDS networks [17].

TABLE I: Comparison of Existing Multi-Robot Middleware Frameworks

Middleware	Strengths	Drawbacks
Orocos (Open Robot Control Software) [18]	Low-latency, real-time execution; highly configurable; run-time introspection.	Complex integration across heterogeneous fleets (e.g., ROS-based systems); steep learning curve for real-time scheduling.
CLARAty (Coupled Layer Architecture for Robotic Autonomy) [19]	Proven on NASA robotic missions; modular and flexible component layering.	Discontinued; outdated APIs; limited community support; poor compatibility with modern systems.
MoBeX	Simple, centralized task dispatch; easy deployment in static environments.	Poor scalability in dynamic settings; lacks negotiation, fault tolerance, and resource arbitration.
SPICA	DDS-backed reliable communication; supports distributed shared memory and ontology messaging.	Lacks native resource arbitration; DDS setup complexity; potential latency overhead.

b) Existing MultiRobot Middleware: Beyond the ROS ecosystem, several other middleware frameworks have been proposed to address multi-robot challenges. A selection of notable examples is presented in Table I.

c) Discussion: A comparative analysis reveals that while each of these frameworks offers valuable features, they often lack key functionalities required for modern, large-scale deployments. The primary deficiencies identified are:

- *Resource arbitration* for shared spaces (e.g., corridors, pick-up points).
- *Distributed conflict resolution* mechanisms, such as peer-to-peer bidding or negotiation, which are central to the Multi-Agent Path Finding (MAPF) problem [20].
- *Inherent fault tolerance*, including robust agent decommissioning and recommissioning protocols, which requires dedicated fault detection, identification, and recovery (FDIR) strategies [21].
- *Ease of integration* across heterogeneous platforms without requiring excessive custom wrappers or adapters.

Modern alternatives like ROS 2 and Open-RMF are designed to address many of these limitations directly.

d) OpenRMF: To address these gaps, Open-RMF was developed as a comprehensive, open-source solution for interoperability. OpenRMF simplifies heterogeneous fleet integration by defining a narrow and standardized set of messages for fleet and infrastructure (e.g., doors, elevators) integration (`rmf_fleet_msgs`, `rmf_door_msgs`). This enables distributed conflict prevention and resolution, and supports task auctioning (bidding) to assign tasks to the most suitable robot based on availability and capability [8], [22]. Its core modules-`rmf_traffic`, `rmf_task`, `rmf_simulation`, and `rmf_visualization`provide scalable and extensible services for scheduling, path planning, and simulation. Recent enhancements include mutex groups for exclusive resource locking, dynamic robot commissioning/decommissioning, and reservation systems for shared locations like pickup/dropoff points [23].

B. Mapping and Navigation Graphs

A prerequisite for autonomous navigation in any MRS is a robust representation of the environment. This is typically achieved through a two-stage process of mapping and graph generation. Cartographer is a widely-used SLAM library that generates high-fidelity 2D occupancy grids using graph optimization-based techniques [24]. It stands as a modern alternative to classic filter-based SLAM approaches like GMapping [25]. The resulting occupancy grids are then converted into RMF-compatible navigation graphs, which involves extracting lane vertices, edges, and waypoint names, often via the RMF Traffic Editor GUI [22]. These navigation graphs allow RMF to precompute feasible routes and enable collision-free motion planning by constraining robot movement to a network of discrete lanes, a common and effective approach in multi-robot traffic management [26].

C. Free Fleet (rmf_demos) Implementation

To facilitate adoption and testing, the Open-RMF project provides a set of demonstration packages, including a reference implementation known as 'Free Fleet'. The `rmf_demos` package contains example fleet adapters, launch files, and scenario configurations that showcase OpenRMFs capabilities in a simulated environment [27]. These demos utilize the Free Fleet server to manage task bidding, scheduling, and visualization. Adapting `rmf_demos` to real hardware, as was done in this project, involves customizing the provided fleet adapters to translate abstract RMF messages into the robot-specific commands recognized by its native navigation stack.

D. TurtleBot3 Platform

The experimental validation in this work was conducted on the TurtleBot3 platform, chosen for its accessibility and strong ROS 2 support. The TurtleBot3 (TB3) is an affordable, ROS 2-compatible, differential-drive mobile robot that is used extensively in robotics education and research [28]. Its compact design, seamless integration with the ROS 2 navigation stack (Nav2) [29], and official support for SLAM libraries like Cartographer make it an ideal testbed for multirobot experiments. The Nav2 stack itself is highly modular, supporting various planning algorithms, including state-of-the-art planners like the Smac Planner [30]. Furthermore, the simulation models for the TB3 are well-supported and maintained within the Gazebo robotics simulator [31].

E. Summary

In summary, while the landscape of MRS middleware is diverse, Open-RMF distinguishes itself through a unique combination of features designed for interoperability and scalability. These features include:

- A **Unified Traffic Schedule**, which acts as a distributed database for realtime trajectory sharing and deconfliction.
- A **Resource Negotiation** protocol that enables peertopeer conflict resolution [8].
- A **Task Auctioning** system that facilitates auction-based task allocation to the most available and capable robot, a well-studied problem in MRS [32], [33].
- An **Extensible Software Development Kit (SDK)** with stable C++ and Python APIs that enable rapid integration and extension for new robot types.

This project leverages these capabilities and adapts them to the TB3 hardware platform to demonstrate practical, real-world deployment scenarios.

III. Objectives

Based on the identified gaps in existing solutions and the advanced capabilities of Open-RMF, this project was defined by a primary goal and a set of specific, measurable objectives.

A. General Objective

The overarching goal of this research is as follows:

> This project presents the design, implementation, and evaluation of a multi-robot middleware framework based on Open-RMF. The proposed framework facilitates dynamic task scheduling, conflict-free navigation, and real-time fleet management. The system's performance is validated through both simulation and hardware deployments on a team of two TurtleBot3 robots.

B. Specific Objectives

To achieve the general objective, the project was broken down into the following specific aims:

- **Environment Setup & Configuration**: To document the required software and hardware components for a fully functional deployment of ROS 2, OpenRMF, Cartographer, and RMF-Web.
- **Mapping & Navigation Graph Generation**: To utilize Cartographer to map a physical environment and subsequently convert the generated occupancy grid into an RMF-compliant navigation graph via the Traffic Editor tool.
- **Fleet Adapter Development**: To implement and validate a custom TB3 fleet adapter that interfaces RMFs `rmf_fleet_msgs` with the TB3s native Nav2 navigation stack, ensuring real-time odometry and command relay.
- **Simulation Integration**: To adapt the `rmf_demos` (Free Fleet) examples to simulate two TB3 robots in the Gazebo simulator, thereby validating task bidding, traffic negotiation, and RMF-Web visualization prior to hardware deployment.
- **Hardware Deployment**: To deploy the complete software stack on two physical TB3 robots and execute patrol and delivery tasks in a laboratory environment, while measuring key performance metrics.
- **Evaluation & Analysis**: To collect and analyze empirical data from both simulation and hardware experiments to evaluate system effectiveness, identify performance bottlenecks, and propose improvements for future work.

IV. Environment Setup and Configuration

This section details the hardware and software prerequisites, along with the configuration steps required to replicate the experimental environment.

A. System Requirements

The successful deployment of the described system is contingent upon the following hardware and software components:

- **Operating System:** Ubuntu 24.04 LTS
- **Robotics Software:** ROS 2 Jazzy Jellisco
- **Simulation Environment:** Gazebo Harmonic
- **Middleware:** Open-RMF v2.0.3 (`rmf_core`, `rmf_demos`, `rmf_visualization`, `rmf_simulation`)
- **Development Tools:** Python 3.8, `pip`, `colcon`, `rosdep`
- **Web Dashboard Tools:** Node.js 16, `pnpm` (for RMF-Web)
- **Hardware:** TurtleBot3 Burger (2 units), each with an onboard Raspberry Pi 4, and a shared Wi-Fi network.

B. ROS 2 and Open-RMF Installation

The foundational software layers consist of the Robot Operating System 2 and the Open-RMF packages. Installation should proceed by following the official ROS 2 Jazzy documentation to install ROS 2 and its core packages [34]. Subsequently, Open-RMF v2.0.3 is installed by cloning the `rmf_core` and `rmf_demos` repositories into a colcon workspace and building from source [22]. It is crucial to source both the ROS 2 environment script (`/opt/ros/jazzy/setup.bash`) and the new RMF workspace script (`~/rmf_ws/install/setup.bash`) in the shell environment.

C. Cartographer and Map Generation

Environment mapping was performed using the Cartographer SLAM package to produce a 2D occupancy grid. The ROS 2 branch of Cartographer is used to generate a high-fidelity 2D map of the laboratory environment [24]. After building the Cartographer workspace, the SLAM node is launched on a TB3, which is then teleoperated throughout the environment. The primary outputs of this process are:

- `map.pgm`: An image file representing the 2D occupancy grid.
- `map.yaml`: A metadata file containing the map's resolution, origin, and other properties.

These files serve as the input for the Traffic Editor in the next stage.

D. Traffic Editor and Navigation Graph Generation

The generated occupancy grid was then manually converted into a semantic navigation graph using the RMF Traffic Editor. This GUI-based tool facilitates the creation of RMF navigation graphs by guiding the user through the following steps:

1) Creating a new building model (e.g., Lab1 - CRIS).
2) Importing the `.pgm` occupancy grid as a floor plan.
3) Adding vertices at key navigational points, such as corridor intersections and room entrances.
4) Drawing environmental features like walls and floor polygons for visualization.
5) Placing functional elements such as doors and setting their motion directions.
6) Defining lanes as directed edges between vertices to represent valid paths.

7) Setting scale measurements to ensure the graph corresponds to real-world dimensions.
8) Adding robot spawn points and charger locations.
9) Defining mutex groups to control access to single-occupancy zones like narrow passages.
10) Exporting the final building model (`*.building.yaml`) and its associated navigation graph (`nav_graphs/1.yaml`).

The exported files are stored in a dedicated project directory, as shown in the example workspace structure in Appendix B. The relationship between the Traffic Editor's graphical representation and the resulting simulation world is illustrated in Figure 1.

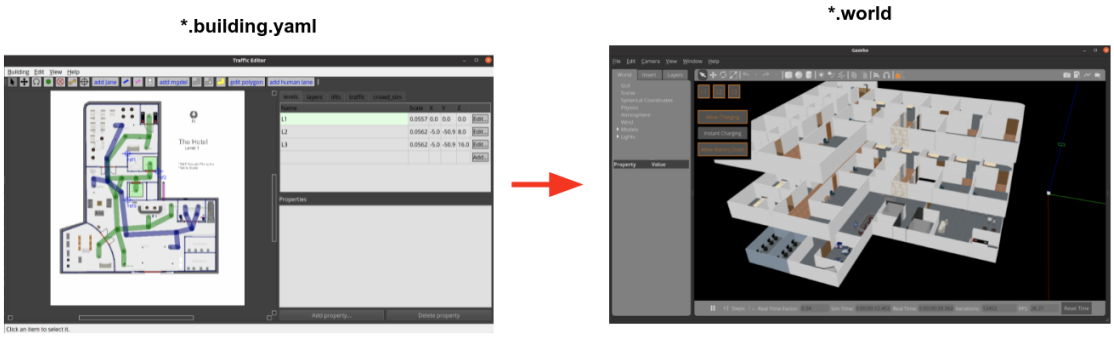
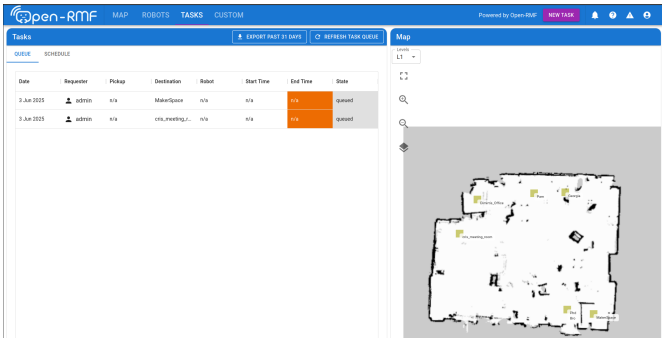


Fig. 1: The RMF Traffic Editor GUI (left) showing the creation of a `*.building.yaml` file, and the corresponding Gazebo world (`*.world`) on the right.

E. RMF-Web Dashboard

For real-time monitoring and task management, the RMF-Web visualization dashboard was employed. RMF-Web provides a web-based interface for live fleet management. After installing Node.js and `pnpm`, the `rmf-web` repository is cloned, its dependencies are installed via `pnpm install`, and the server is launched with `pnpm start`. The dashboard becomes accessible at `http://localhost:3000`, displaying robot positions, active tasks, and the building layout in real time, as shown in Figure 2. The interface provides dedicated views for managing the task queue (Figure 2a) and visualizing the fleet on the map (Figure 2b). The design of such an interface is a key aspect of Human-Robot Interaction (HRI), as it directly impacts an operator's situational awareness and cognitive load [35].



(a) The task queue view in the RMF-Web GUI.

(b) The map view in the RMF-Web GUI, showing waypoints and the map overlay.

Fig. 2: Screenshots of the RMF-Web Dashboard, illustrating the task and map views.

F. TurtleBot3 Environment

The physical robot setup for each TurtleBot3 unit required specific onboard configurations. Each TB3 (Burger model) is equipped with a Raspberry Pi 4 running ROS 2 Jazzy. The `TURTLEBOT3_MODEL` environment variable must be set to `burger`. For multi-robot communication, each TB3 must be on the same Wi-Fi network as the host PC, and either static IP addresses or unique ROS 2 domain IDs must be configured to prevent network conflicts. Remote access to each robot via SSH is recommended for efficient remote launching of nodes.

V. System Architecture and Methodology

This section provides a detailed description of the system's architecture, the data flow between components, and the methodologies used for mapping, control, and tasking.

A. Overall System Architecture

Figure 3 presents a high-level overview of the multi-robot system architecture. The system is composed of the following primary components:

- **Host PC:** Runs ROS 2 Jazzy, Cartographer for localization, the core Open-RMF services (`rmf_core`, `rmf_demos`), the RMF-Web dashboard, and the Free Fleet server node.
- **TB3 Robots (TB31, TB32):** Each is equipped with a Raspberry Pi 4 running ROS 2 Jazzy, the Nav2 stack for autonomous navigation, and a custom-developed fleet adapter.
- **Traffic Schedule Database (`rmf_traffic`):** A distributed ledger that enables unified trajectory sharing and proactive deconfliction between robots.
- **Reservation Service:** Arbitrates access to shared resources and locations, such as pickup/drop-off points or charging stations.
- **RMF-Web Dashboard:** A graphical interface for real-time fleet visualization and user-initiated task dispatching.

The communication between the Host PC and the robot fleet is managed by a Zenoh router, as depicted in Figure 3. The Free Fleet adapter on the server publishes commands using robot-specific namespaces (e.g., `/robot1/navigate_to_pose`). The Zenoh router directs these messages to the appropriate robot, where a Zenoh bridge translates them into native ROS 1 or ROS 2 topics. This decoupling allows the system to seamlessly manage a heterogeneous fleet, including robots running different ROS distributions, enhancing scalability and flexibility [17].

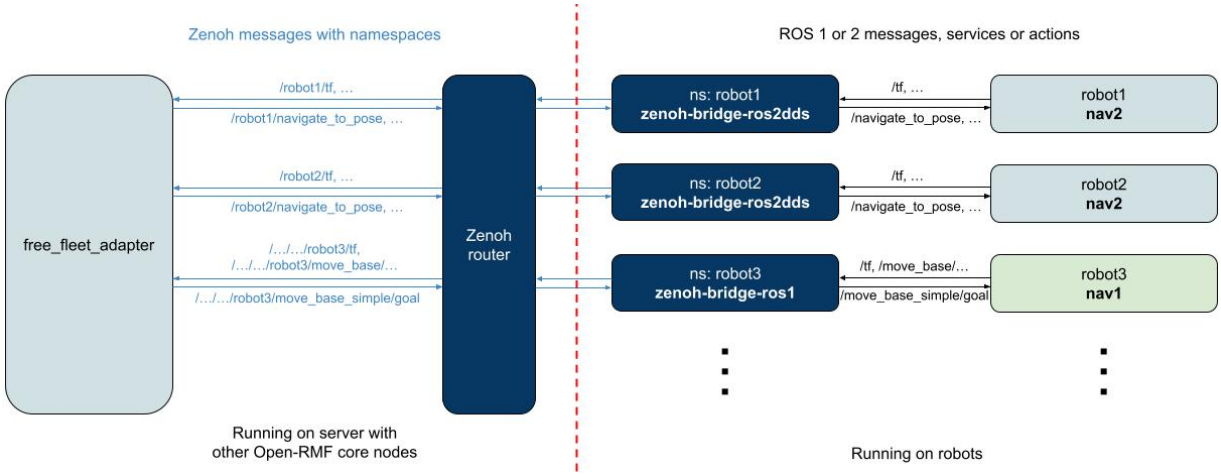


Fig. 3: The communication architecture illustrating data flow from the Free Fleet adapter via the Zenoh router to individual robots. The use of Zenoh bridges enables interoperability between ROS 1 and ROS 2 agents in a single fleet.

B. Environmental Representation

The foundation of the system's methodology is a robust, multi-layered representation of the environment. This was created using the pipeline detailed in the Environment Setup section, which includes generating a 2D occupancy grid with Cartographer (see Sec. IV-C) and converting it into a semantic navigation graph using the RMF Traffic Editor (see Sec. IV-D). This graph serves as the primary data structure for all RMF scheduling and path-planning services.

C. Free Fleet Integration and Control

The Free Fleet approach enables RMF to command custom mobile robots. Figure 4 illustrates the generalized architecture, where a Full-Control Fleet Adapter and a ROS 2 Free Fleet Server communicate with multiple robot clients. This design is agnostic to the underlying robot's software stack (e.g., ROS 1, ROS 2, or a non-ROS system), provided a compatible client is running on the robot. This project uses specific fleet and robot configuration files to define the capabilities and properties of the TB3 fleet. Example snippets of these configurations are provided in Appendix B.

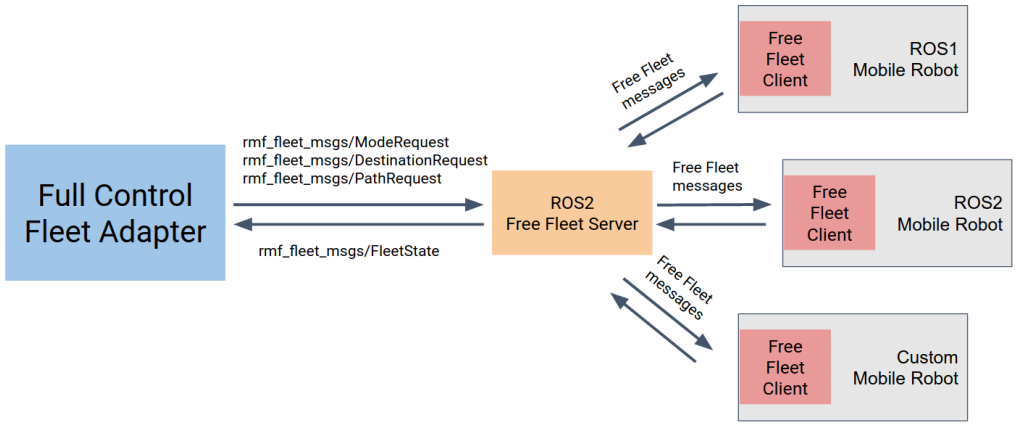


Fig. 4: A generalized block diagram showing the Full-Control Fleet Adapter and ROS 2 Free Fleet Server communicating with multiple Free Fleet Clients (ROS 1, ROS 2, or custom).

D. Fleet Adapter Logic

The custom TB3 fleet adapter serves as the critical bridge between the abstract commands of RMF and the concrete execution layer of the Nav2 stack. Its highlevel logic is organized as follows:

- **Initialization**: A ROS 2 node named `tb3_fleet_adapter` is initialized.
- **Subscriptions**: The adapter subscribes to RMF command topics.
  - `/lab1/destination_requests` (Message: `rmf_fleet_msgs/DestinationRequest`)
  - `/lab1/mode_requests` (Message: `rmf_fleet_msgs/ModeRequest`)
- **Publications**: The adapter publishes robot state information at a regular interval (e.g., 10 Hz).
  - `/lab1/fleet_states` (Message: `rmf_fleet_msgs/FleetState`)
- **Callbacks and Actions**:
  - On receiving a `DestinationRequest`, the adapter extracts the target waypoint, converts its graph coordinates to map coordinates, creates a `NavigateToPose` action goal, and sends this goal to the Nav2 action server.
  - On receiving a `ModeRequest`, the adapter pauses or resumes navigation. A `PAUSE` command cancels the current Nav2 goal, while a `RESUME` command re-sends the last goal.
  - Periodically, the adapter queries the current TB3 pose from Nav2 feedback, populates a `RobotState` message with its name, battery level, location, and velocity, and publishes this information in a `FleetState` message.

The logical flow of the fleet adapter is depicted in Figure 5.

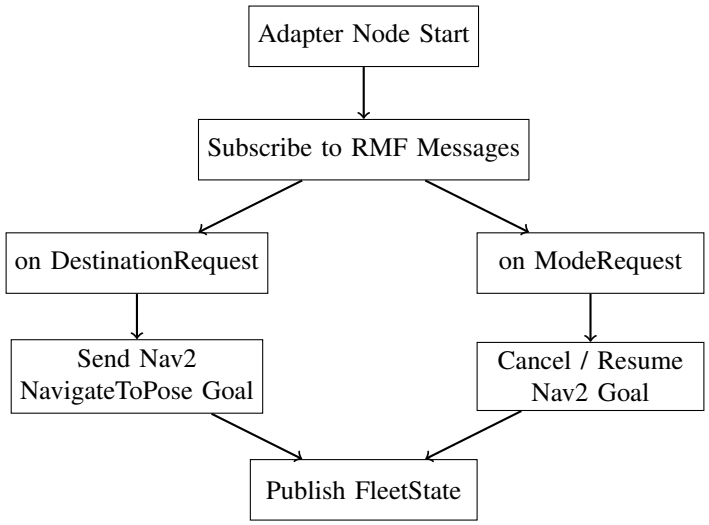


Fig. 5: A flowchart illustrating the Fleet Adapter's primary workflow.

E. Simulation-based Validation

To validate all configurations prior to hardware deployment, the system was first run in simulation. This was achieved by adapting the `rmf_demos_gzoffice.launch.xml` Free Fleet example to load two TB3 models into the Gazebo simulator and start all required Open-RMF services. While simulation is invaluable for accelerating development, it is critical to acknowledge the "reality gap," where discrepancies between simulated and real-world physics and sensor models can lead to divergent behaviors on physical hardware [36]. An example of the Gazebo simulation, overlaid with the RMF-Web user interface for monitoring, is shown in Figure 6. Techniques such as domain randomization can be employed to help mitigate these effects [37].

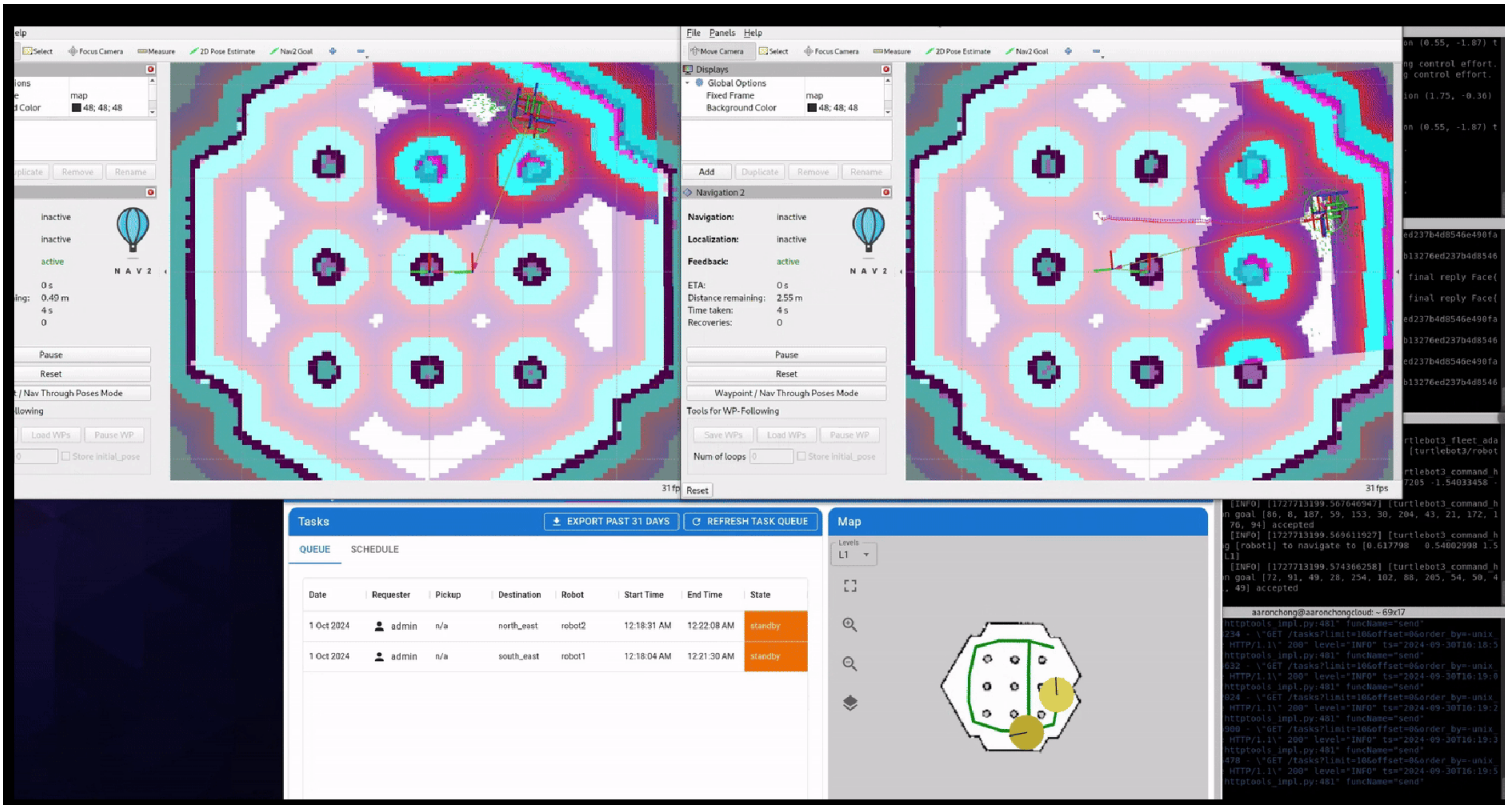


Fig. 6: The simulation environment showing robot navigation, with the RMF-Web UI overlay indicating task and robot status.

F. Task Definition and Dispatching

Tasks within the RMF ecosystem are defined and dispatched as ROS 2 actions using nodes from the `rmf_task_ros2` package. For this project, the following task types were configured:

- **Patrol Task:** A task requiring a robot to visit a sequence of waypoints in a continuous loop (e.g., TB31 patrols between `start_pose` and `cris_meeting_room` for a total of 2 loops).
- **Delivery Task:** A multi-stage task requiring a robot to move from a starting location to a pickup point, then to a drop-off point, and finally back to a designated charger location.
- **GoTo Task:** A simple task involving single waypoint navigation (e.g., travel to `cris_meeting_room`).

The assignment of these tasks is handled via a market-based bidding mechanism, a common and effective strategy for dynamic task allocation in MRS [32], [38], as shown in Figure 7.

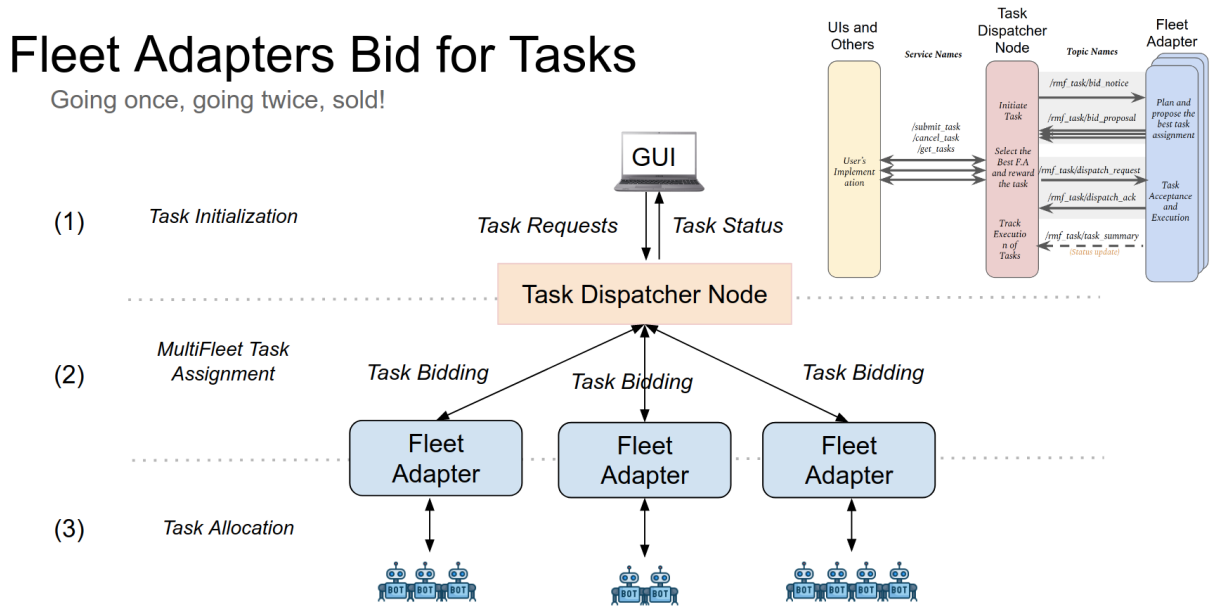


Fig. 7: The task bidding process: The Task Dispatcher issues a bid notice; each Fleet Adapter evaluates the task and returns a cost proposal; the dispatcher then selects the best bid and assigns the task accordingly.

VI. Hardware Deployment and Evaluation

This section describes the physical setup, the process for launching the system on hardware, and the results of an experimental evaluation scenario.

A. Physical Setup

The hardware deployment utilized the following physical setup:

- **TB3 Robots:** The fleet consisted of two TurtleBot3 units, designated TB31 and TB32. Each robot was equipped with an LDS-01 LiDAR sensor and an onboard Raspberry Pi 4. Each ran its respective navigation stack (Nav2 for ROS 2) and the custom fleet adapter node. The physical robots used for the hardware validation are shown in Figure 8.
- **Network:** All devices, including the host PC and both TB3 units, were connected to a single 5GHz Wi-Fi network with static IP assignments to ensure stable communication.

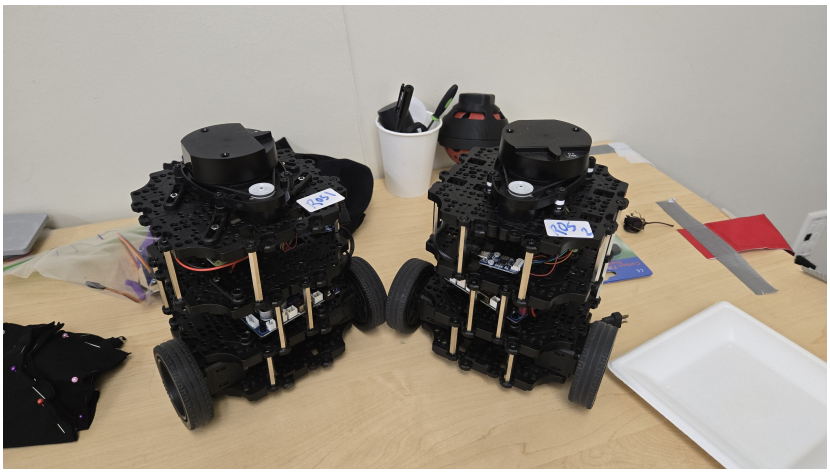


Fig. 8: The physical TurtleBot3 robots (TB3–1 and TB3–2) used in the experiments.

B. Launching RMF on Hardware

Deployment on the physical hardware was initiated using the following procedure. An SSH connection was established to each TB3 unit to set the ROS 2 domain ID and source the required ROS 2, Cartographer, and RMF workspaces. On the host

PC, the Open-RMF core nodes and the RMF-Web dashboard were launched. Concurrently, on each TB3, the Nav2 stack and the custom fleet adapter were started. Successful initialization was verified by confirming that both TB3 robots appeared in the ”Robots” list within the RMF-Web dashboard. The specific commands used for this process are detailed in Appendix C.

C. Scenario 1: Parallel Patrol with Shared Space

An experiment was designed to evaluate the system's ability to manage simultaneous tasks that involve a shared, contested space.

a) Task Setup: The following two tasks were dispatched concurrently from the RMF-Web interface:

- **TB3–1 Task:** Navigate between waypoint `start_pose` and waypoint `cris_meeting_room`, completing 5 round-trips.
- **TB3–2 Task:** Navigate between waypoint `start_pose` and waypoint `maker_space`, completing 5 round-trips.

The paths for these two tasks overlap in a central corridor, which was defined as a mutex-protected zone in the navigation graph. Figure 9 provides a visual representation of the robots operating within the physical environment, with the RMF navigation graph lanes and waypoints superimposed.

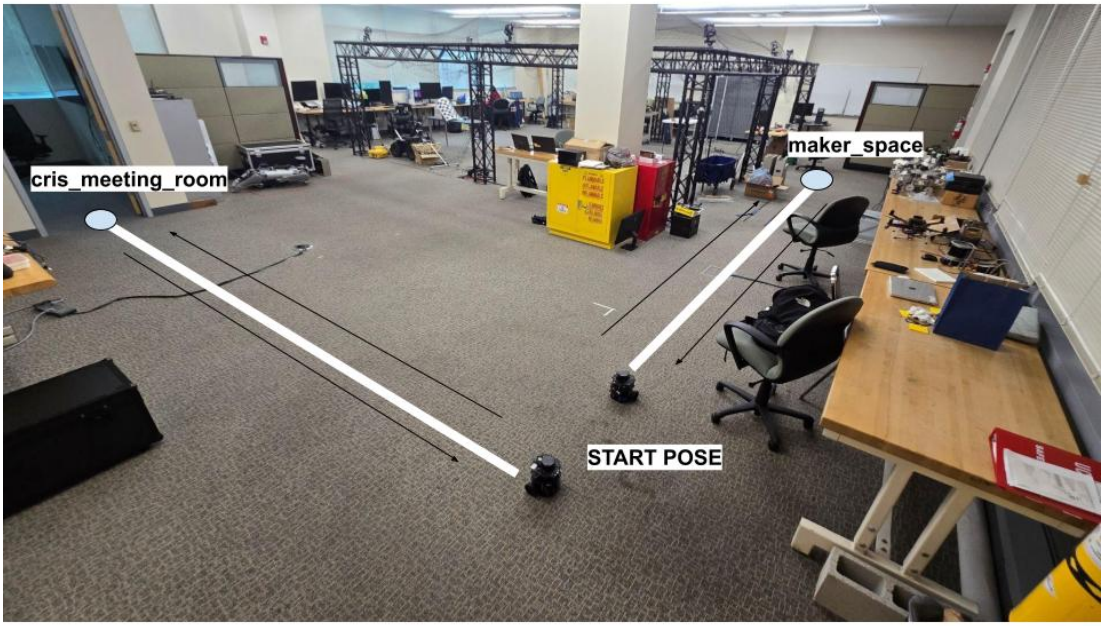


Fig. 9: The TurtleBot3 robots operating in the physical lab environment, with an overlay of the RMF navigation graph showing waypoints and travel lanes.

b) Expected Behavior: The expected sequence of events for this scenario was as follows:

1) TB3–1 is assigned its patrol task, while TB3–2 is assigned its own patrol task.
2) Both robots commence their respective tasks simultaneously and follow paths along the predefined navigation graph.
3) The central corridor, which is traversed by both robots, is protected by a mutex group to prevent simultaneous entry and potential collision.
4) If both robots approach the mutex-protected corridor at the same time, RMF's `rmf_traffic` scheduler facilitates a distributed negotiation. The robot that is granted priority proceeds, while the other waits at the entrance to the mutex zone.
5) Each robot successfully completes its 5 round-trips without collision or deadlock, demonstrating effective conflict resolution and path coordination.

c) Performance Metrics: The following performance metrics were collected during the experiment:

- **Task Completion Time:** The total time elapsed from task dispatch to final completion for each robot.
- **Wait Time:** The average and maximum time a robot spent waiting at the entrance to the corridor mutex zone.
- **Negotiation Overhead:** The number of negotiation messages exchanged and the average duration of a negotiation event.
- **System Load:** The CPU utilization on the host PC and the average network latency to the TB3 units during the experiment.

d) Results: The quantitative results from the experiment are summarized below:

- The average round-trip time for TB3–1 (start $\leftrightarrow$ cris_meeting_room) was 185 seconds (3 minutes, 5 seconds).
- The average round-trip time for TB3–2 (start $\leftrightarrow$ maker_space) was 193 seconds (3 minutes, 13 seconds).
- The average wait time at the corridor mutex was 2.4 seconds.
- A total of 12 negotiation message sets were exchanged (6 per robot), with an average negotiation duration of 0.08 seconds.
- The host PC CPU usage remained between 25-35%, and the average network latency to the TB3s was 15 ms.

VII. Discussion and Conclusion

This section synthesizes the projects key findings, draws overarching conclusions, and outlines promising directions for future work.

The implemented Open-RMF system exhibited several noteworthy *strengths*. Its modular and extensible architectureespecially the fleet-adapter abstractionenabled the seamless integration of new robot types with minimal changes to the core, nurturing a vendor-agnostic ecosystem [8]. The combination of mutex groups, a central traffic scheduler, and advanced reservation logic delivered proactive, smooth conflict resolution that outperformed purely reactive avoidance schemes [39]. Real-time visualisation through the RMF-Web dashboard further bolstered operator trust by presenting live robot positions, task progress, and resource claims [40]. Finally, the built-in decommission/recommission workflow provided a first line of fault tolerance: when a robot was removed, its active task was automatically re-auctioned and reassigned within seconds, illustrating baseline system robustness [41].

Despite these advantages, several *limitations* surfaced. Generating navigation graphs with Traffic Editor remains a tedious, error-prone manual exercise for large sites; automating topological-graph extraction from metric maps is therefore still an open research problem [42]. The current prototype is also tightly coupled to Nav2; platforms lacking a Nav2-compatible interface would require substantial additional adapter layers, epitomising the last-mile integration challenge in robotics middleware. Moreover, the negotiation protocol proved sensitive to network conditions: round-trip latencies beyond roughly 50 ms noticeably slowed negotiations and occasionally induced deadlocks, echoing known DDS issues on lossy networks [43]. Finally, the Raspberry Pi 4 onboard the TurtleBot3 constrained simultaneous SLAM and navigation, underscoring the cost of deploying resource-heavy components on low-power compute.

Operational experience crystallised three major *lessons*. First, rigorous time synchronisation across all nodesfleet adapter, Nav2, RMF core, and Cartographeris essential; any clock drift between simulated and wall time manifests as erroneous position estimates and failed plans. Second, mutex zones must be defined with appropriate granularity: corridor-wide zones induced needless queuing, whereas finer segmentation markedly improved throughput. Third, the weighting used during task bidding (e.g., travel distance versus battery level) strongly affects load balance; careless tuning can starve low-battery robots, whereas calibrated weights promote equitable task distribution [10].

Overall, the project *validated* the feasibility of deploying Open-RMF on resource-constrained TurtleBot3 robots. Real-time scheduling, proactive conflict avoidance, reservation-based execution, and clear operator oversight were achieved in both simulation and hardware trials. Performance tests showed efficient task completion, modest negotiation overhead, and graceful recovery from single-robot faults, confirming Open-RMFs maturity for scalable, interoperable multi-robot deployments.

Looking forward, several *avenues* merit exploration. (i) Automating navigation-graph generation from occupancy grids would drastically reduce setup time [44]. (ii) A Rust-based re-implementation of key RMF modules (e.g., `rmf_traffic`, `rmf_task`) could yield memory-safe, high-performance concurrency [23]. (iii) Broadening platform supporte.g., ClearPath Husky, MiR, or AGVs conforming to VDA 5050will enable heterogeneous-fleet studies and accelerate industry adoption [45]. (iv) Adaptive QoS settings and predictive deadlock-avoidance techniques could make negotiations more resilient to high or variable network latency. (v) Finally, large-scale experiments with 810 robots in warehouse-like environments are essential to stress-test the traffic-negotiation algorithms under the dense conditions characteristic of modern multi-agent path-finding scenarios [46].

Acknowledgment

The author gratefully acknowledges the support of Professor Konstantinos Karydis and the Autonomous Robotic and Control Systems (ARCS) Laboratory at the University of California, Riverside, for providing the resources and guidance necessary for this project. Special thanks are extended to the Open-RMF open-source community for their exemplary documentation and demonstration packages, which served as an invaluable foundation for this work.

APPENDIX A
LIST OF ABBREVIATIONS

- DDS: Data Distribution Service
- FDIR: Fault Detection, Identification, and Recovery
- GUI: Graphical User Interface
- HRI: Human-Robot Interaction
- MAPF: Multi-Agent Path Finding
- MRS: Multi-Robot System
- PGM: Portable Gray Map
- QoS: Quality of Service
- RMF: Robotics Middleware Framework
- ROS: Robot Operating System
- SDK: Software Development Kit
- SLAM: Simultaneous Localization and Mapping
- TB3: TurtleBot3
- YAML: YAML Aint Markup Language

APPENDIX B
EXAMPLE CONFIGURATION SNIPPETS

This appendix provides examples of the YAML configuration files and the overall file structure used in the project.

A. RMF Project Workspace Structure

A structured project workspace is recommended for managing the various configuration and asset files. A typical structure under `~/rmf_ws/src/` is as follows:

```
rmf_ws/src/
 project_assets/           % Copied from rmf_demos/rmf_demos_assets
 project_fleet_adapter/    % Custom TB3 fleet adapter source code
 project_maps/             % Custom maps and navigation graphs
    Lab1/
        Lab1.building.yaml
        nav_graphs/
           1.yaml
        floor_plans/
 project_config/           % Launch and parameter configuration files
    launch/
       common.launch.xml
       Lab1.launch.xml
       simulation.launch.xml
    config/
        Lab1/
            tb3_fleet_config.yaml
            tb3_robot1_config.yaml
 project_simulation/         % Gazebo simulation models and launch files
    launch/
        simulation.launch.xml
 project_tasks/            % Custom task dispatching scripts
 rmf_ws_installation_readme.md % Project documentation
```

B. TB3 Fleet Configuration (fleet.yaml)

The following YAML snippet shows a minimal TB3 fleet configuration used by the RMF Free Fleet server.

Listing 1: Example fleet configuration file.

```yaml
fleet_name: tb3_fleet
fleet_manager:
 user: "some_user"
 password: "some_password"
```

```
  server_uri: "http://localhost:8000"
task_capabilities:
  loop: true
  delivery: false
  cleaning: false
```

C. TB3 Robot Configuration (robot.yaml)

This robot-specific YAML file describes the properties and capabilities of an individual TB3 robot for use by the fleet adapter.

Listing 2: Example robot configuration file.

```
name: tb3_1
initial_pose:
  x: 0.0
  y: 0.0
  yaw: 0.0
  map_name: "L1"
rmf_fleet:
  name: "tb3_fleet"
  robot_type: "tb3_burger"
  motion:
    linear_velocity: 0.22 # m/s
    angular_velocity: 1.0 # rad/s
  battery:
    capacity: 300.0 # Wh
    charging_waypoint: "lab_charger1"
    charge_duration: 1800 # seconds
```

APPENDIX C
LAUNCH COMMANDS FOR HARDWARE DEPLOYMENT

This appendix details the commands required to launch the full hardware stack, broken down by machine. These commands should be executed in separate terminals.

A. Commands for the Host PC

Execute the following commands on the central host PC that runs the core RMF services.

Listing 3: Commands to launch the core RMF services on the host PC.

```
# Navigate to your RMF workspace
cd ~/rmf_ws

# Source the workspace
source install/setup.bash

# Launch the main project configuration for the hardware test
ros2 launch project_config Lab1.launch.xml use_sim_time:=false failover_mode:=false
```

REFERENCES

[1] J. J. Roldn, J. del Cerro, A. Barrientos, and J. A. Rosario, "Special Issue on Multi-Robot Systems," *Sensors*, vol. 21, no. 23, p. 7894, 2021.

[2] L. E. Parker, "Multiple mobile robot systems," in *Springer Handbook of Robotics*, B. Siciliano and O. Khatib, Eds. Springer, 2012, pp. 921941.

[3] I. Navarro and F. Mata, "An introduction to multi-robot systems," *IEEE Third International Conference on Intelligent Systems and Knowledge Engineering*, 2012, pp. 1-6.

[4] G. Kurt, "Towards Intelligent Robotic Systems," *Applied Sciences*, vol. 10, no. 4, p. 1368, 2020.

[5] M. Brambilla, E. Ferrante, M. Birattari, and M. Dorigo, "Swarm robotics: a review from the swarm engineering perspective," *Swarm Intelligence*, vol. 7, no. 1, pp. 1-41, 2013.

[6] Y. Rizk, M. Awad, and E. W. Tunstel, "Cooperative heterogeneous multi-robot systems: A survey," *ACM Computing Surveys (CSUR)*, vol. 52, no. 2, pp. 1-31, 2019.

[7] L. Almeida, P. Pedreiras, and J. A. G. Fonseca, "The FTT-CAN protocol: why and how," *IEEE Transactions on Industrial Electronics*, vol. 49, no. 6, pp. 1189-1201, 2002.

[8] M. Yadunandana, G. S. Lee, A. H. Q. H. Dang, and M. Rusli, "Open-RMF: A flexible and scalable framework for managing multi-fleets of robots," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021, pp. 3201-3208.

[9] A. Meseguer Valenzuela and F. Blanes Noguera, "State of the Art in Robot Fleet Management: From Task Allocation to Motion Planning," 2023. [Online]. Available: https://ruc.udc.es/dspace/bitstream/handle/2183/33658/2023_Meseguer_Valenzuela_Andres_State_of_the_Art_in_Robot_Fleet_Management.pdf

[10] B. P. Gerkey and M. J. Matari, "A formal analysis and taxonomy of task allocation in multi-robot systems," *The International Journal of Robotics Research*, vol. 23, no. 9, pp. 939-954, 2004.

[11] B. I. Ahmad and M. N. A. Khan, "A survey on multi-robot task allocation," in *2019 International Conference on Robotics and Automation Sciences (ICRAS)*, 2019, pp. 1-6.

[12] M. Quigley et al., "ROS: an open-source Robot Operating System," in *ICRA Workshop on Open Source Software*, vol. 3, no. 3.2, 2009.

[13] S. Macenski et al., "Robot Operating System 2: Design, architecture, and uses in the wild," *Science Robotics*, vol. 7, no. 66, p. eabm6074, 2022.

[14] Y. Maruyama, S. Kato, and T. Azumi, "Exploring the performance of ROS2," in *Proceedings of the 13th International Conference on Embedded Software (EMSOFT)*, 2016, pp. 1-10.

[15] P. Gerardo, R. V. van Nieuwpoort, and H. E. Bal, "The Data Distribution Service: a new connectivity standard for high-performance distributed systems," *Concurrency and Computation: Practice and Experience*, vol. 22, no. 15, pp. 2115-2127, 2010.

[16] L. Esteves, J. C. B. de Faria, and R. de Oliveira, "An experimental evaluation of DDS for real-time robotic systems," in *2021 IEEE 24th International Symposium on Real-Time Distributed Computing (ISORC)*, pp. 83-92.

[17] A. Michel, P. G. Rapin, F. Ndiaye, and C. Zanolin, "Zenoh: A Scalable, Unified, and Location-Transparent Protocol for Modern Distributed and Edge Computing," ZettaScale Technology, Tech. Rep., 2021.

[18] H. Bruyninckx, "Open robot control software: the OROCOS project," in *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation*, vol. 3, 2001, pp. 2523-2528.

[19] R. Volpe, I. Nesnas, T. Estlin, D. Mutz, R. Petras, and H. Das, "The CLARAty architecture for robotic autonomy," in *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation*, vol. 3, 2001, pp. 2468-2473.

[20] R. Stern, N. R. Sturtevant, A. Felner, S. Koenig, J. Walker, T. K. S. Kumar, C. Ma, and L. C. T. Brito, "Multi-agent pathfinding: Definitions, variants, and benchmarks," in *Proceedings of the International Symposium on Combinatorial Search*, vol. 10, 2019, pp. 101-108.

[21] R. Caccavale, A. Giglio, G. Giglio, and F. Pierri, "A Survey on Fault Management in Multi-Robot Systems," *Journal of Intelligent & Robotic Systems*, vol. 95, pp. 799816, 2019.

[22] Open Robotics, "Open-RMF Documentation," *https://open-rmf.readthedocs.io/*, 2024.

[23] Open Robotics, "The State of OpenRMF," *ROSCon 2024 Slides*, 2024. [Online]. Available: https://roscon.ros.org/2023/presentations/The_State_of_Open-RMF.pdf

[24] W. Hess, D. Kohler, H. Rapp, and D. Andor, "Real-time loop closure in 2D LIDAR SLAM," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 1271-1278.

[25] G. Grisetti, C. Stachniss, and W. Burgard, "Improved techniques for grid mapping with Rao-Blackwellized particle filters," *IEEE Transactions on Robotics*, vol. 23, no. 1, pp. 34-46, 2007.

[26] W. Wang, J. Pan, and D. Manocha, "A review of multi-robot coordination with deep reinforcement learning," *Current Robotics Reports*, vol. 1, no. 4, pp. 257-268, 2020.

[27] Open Robotics, "rmf_demos GitHub Repository," *https://github.com/open-rmf/rmf_demos*, 2024.

[28] ROBOTIS, "TurtleBot3," *https://www.robotis.us/turtlebot-3/*, 2024.

[29] S. Macenski, F. Martn, R. White, and J. Gins, "The Marathon 2: A navigation system," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020, pp. 2757-2764.

[30] S. Macenski, "The Smac Planner: A Versatile and Efficient Planner for Navigation in 2D," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 2021, pp. 11956-11962.

[31] N. Koenig and A. Howard, "Design and use paradigms for Gazebo, an open-source multi-robot simulator," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, vol. 3, 2004, pp. 2149-2154.

[32] M. B. Dias, R. M. Zlot, N. Kalra, and A. Stentz, "Market-based multirobot coordination: A survey and analysis," *Proceedings of the IEEE*, vol. 94, no. 7, pp. 1257-1270, 2006.

[33] G. A. Korsah, M. M. B. Dias, and A. Stentz, "A comprehensive taxonomy for multi-robot task allocation," *The International Journal of Robotics Research*, vol. 32, no. 12, pp. 1495-1512, 2013.

[34] Open Robotics, "ROS 2 Documentation," *https://docs.ros.org/en/jazzy/*, 2024.

[35] J. L. Drury, J. Gerhardt, and E. Scholtz, "Design principles for human supervision of multi-robot teams," in *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, vol. 48, no. 3, 2004, pp. 462-466.

[36] J. Collins, S. Chand, A. Vanderkop, and D. Howard, "A Review of Physics Simulators for Robotic Applications," *IEEE Access*, vol. 9, pp. 51439-51453, 2021.

[37] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017, pp. 23-30.

[38] R. M. Zlot, A. Stentz, M. B. Dias, and S. Thayer, "Multi-robot exploration controlled by a market economy," in *Proceedings 2002 IEEE International Conference on Robotics and Automation*, vol. 3, 2002, pp. 3016-3023.

[39] J. van den Berg, S. J. Guy, M. Lin, and D. Manocha, "Reciprocal n-body collision avoidance," in *Robotics Research*, 2011, pp. 3-19.

[40] T. B. Sheridan, *Telerobotics, Automation, and Human Supervisory Control*. MIT press, 1992.

[41] A. Avizienis, "The N-version approach to fault-tolerant software," *IEEE Transactions on Software Engineering*, vol. SE-11, no. 12, pp. 1491-1501, 1985.

[42] S. Thrun, "Robotic mapping: A survey," in *Exploring artificial intelligence in the new millennium*, 2002, pp. 1-35.

[43] D. Casini, T. D. C. Fin, G. Nelissen, and G. Buttazzo, "A study on the real-time performance of ROS 2," in *2022 IEEE 25th International Symposium on Real-Time Distributed Computing (ISORC)*, 2022, pp. 11-20.

[44] S. Thrun, "Learning metric-topological maps for indoor mobile robot navigation," *Artificial Intelligence*, vol. 99, no. 1, pp. 21-71, 1998.

[45] VDA - Verband der Automobilindustrie, "VDA 5050: Communication interface for automated guided vehicles (AGV) and the master control," 2022.

[46] G. Sharon, R. Stern, A. Felner, and N. R. Sturtevant, "Conflict-based search for optimal multi-agent pathfinding," *Artificial Intelligence*, vol. 219, pp. 40-66, 2015.

[47] Y. U. Cao, A. S. Fukunaga, and A. B. Kahng, "Cooperative mobile robotics: Antecedents and directions," *Autonomous Robots*, vol. 4, no. 1, pp. 7-27, 1997.