Leevi Linnovaara

Improving Operational Flexibility in AMR Systems Through External Scheduling and Performance Tracking

A Case Study on Extending MiR Fleet Capabilities via REST API Integration

School of Technology and Innovations
Robotics
Master's Programme in Smart Energy

Vaasa 2026

UNIVERSITY OF VAASA
School of Technology and Innovations
Author: Leevi Linnovaara
Title of the thesis: Improving Operational Flexibility in AMR Systems Through External Scheduling and Performance Tracking: A Case Study on Extending MiR Fleet Capabilities via REST API Integration
Degree: Master of Science in Technology
Discipline: Robotics
Supervisor: Jani Boutellier
Year: 2026 **Pages:** 65

ABSTRACT:

Autonomous mobile robots are increasingly used in industrial and intralogistics environments due to their flexible operation in dynamic environments. However, practical deployments often reveal limitations in task scheduling and monitoring, especially in environments without external automation systems, such as programmable logic controllers or warehouse management systems. In these contexts, mission execution tends to be reactive and lacks structured support for time-based or recurring task planning.

The objective of this thesis is to improve the operational flexibility of AMR systems by developing an external scheduling and performance monitoring application. The research extends the capabilities of a commercial AMR platform by using a standardized application programming interface, without modifying the robot system's internal control logic. A constructive research approach is employed to address a practical problem identified in real-world deployments by designing, implementing, and evaluating a system.

The developed solution is implemented as a standalone Python application with a graphical user interface built with the KivyMD framework. Communication with the robot occurs through a REST-based interface, which enables mission execution and system data retrieval. The application provides time-based scheduling, supports both manual and automated mission triggering, and enables the collection and storage of mission execution data. Local data storage is achieved using a lightweight embedded database, enabling independent operation without external infrastructure.

The system is evaluated in an industrial workshop environment with a single autonomous mobile robot. Testing verifies scheduling functionality, execution reliability, and data collection under both standard and exceptional operating conditions. Results demonstrate that time-based mission scheduling can be implemented through an external application layer. The system effectively collects and structures execution data, enabling calculation of performance indicators such as mission success rate and execution duration.

The findings demonstrate that external applications can effectively extend the functionality of AMR systems at the supervisory level. These results support the conclusion that lightweight, API-based solutions offer value in industrial AMR deployments without necessitating modifications to existing robot systems.

KEYWORDS: Autonomous Mobile Robots, Industrial Robotics, Fleet Management, Task Scheduling, Performance Monitoring

VAASAN YLIOPISTO
Tekniikan ja innovaatioiden akateeminen yksikkö
Tekijä: Leevi Linnovaara
Tutkielman nimi: Operatiivisen joustavuuden parantaminen AMR-järjestelmissä ulkoisen aikataulutuksen ja suorituskyvyn seurannan avulla: tapaustutkimus MiR-robottien kyvykkyyksien laajentamisesta REST-API-integraation avulla
Tutkinto: Diplomi-insinööri
Opintosuunta: Robotiikka
Työnohjaaja: Jani Boutellier
Valmistumisvuosi: 2026 **Pages:** 65

Tiivistelmä:

Mobiilirobotteja käytetään yhä enemmän teollisuus- ja sisälogistiikkaympäristöissä, johtuen niiden joustavasta toiminnasta dynaamisissa olosuhteissa. Käytännön toteutuksissa paljastuu kuitenkin usein rajoituksia tehtävien aikataulutuksessa ja seurannassa, etenkin ympäristöissä, joissa ei ole ulkoisia automaatiojärjestelmiä, kuten ohjelmoitavia logiikkaohjaimia tai varastonhallintajärjestelmiä. Tällaisissa tilanteissa tehtävien suorittaminen on yleensä reaktiivista, eikä siinä ole jäsenneltyä tukea aikaperusteiselle tai toistuvalle tehtävien suunnittelulle.

Tämän tutkimuksen tavoitteena on parantaa AMR-järjestelmien toiminnallista joustavuutta kehittämällä ulkoinen aikataulutus- ja suorituskyvyn seurantasovellus. Tutkimus laajentaa kaupallisen AMR-alustan ominaisuuksia käyttämällä vakiintunutta sovellusohjelmointirajapintaa ilman, että robottijärjestelmän sisäistä ohjauslogiikkaa muutetaan. Käytännön sovelluksista tunnistettuun ongelmaan, vastataan rakentavalla tutkimuslähestymistavalla suunnittelemalla, toteuttamalla ja arvioimalla kehitetty ratkaisu.

Kehitetty ratkaisu on toteutettu itsenäisenä Python-sovelluksena, jonka käyttöliittymä on rakennettu KivyMD-kirjastolla. Yhteydenpito robotin kanssa tapahtuu REST-pohjaisen rajapinnan kautta, mikä mahdollistaa tehtävien suorittamisen ja tiedon hakemisen. Sovellus mahdollistaa tehtävien ajastamisen, tukien automaattista ja manuaalista tehtävien käynnistämistä sekä suoritustietojen keräämistä ja tallentamista. Tietojen tallennus toteutetaan kevyellä sulautetulla tietokannalla, mikä mahdollistaa sovelluksen itsenäisen toiminnan ilman riippuvuutta ulkoisesta infrastruktuurista.

Järjestelmää arvioidaan teollisessa ympäristössä, jossa on käytössä yksi robotti. Testeissä tarkastellaan aikataulutustoimintoja, luotettavuutta sekä tiedonkeruuta normaaleissa ja poikkeuksellisissa käyttöolosuhteissa. Tulokset osoittavat, että tehtävien aikataulutus voidaan toteuttaa ulkoisen sovelluksen kautta. Järjestelmä kerää ja jäsentää suoritustietoja tehokkaasti, mahdollistaen esimerkiksi tehtävien onnistumisasteen ja suoritusaikojen laskemisen.

Tulokset osoittavat, että ulkoiset sovellukset voivat tehokkaasti laajentaa AMR-järjestelmien toiminnallisuutta valvontatasolla. Nämä tulokset tukevat johtopäätöstä, että kevyet, API-pohjaiset ratkaisut tarjoavat lisäarvoa teollisissa AMR-käyttöönotoissa ilman, että olemassa oleviin robottijärjestelmiin tarvitsee tehdä muutoksia.

Avainsanat: Autonomiset mobiilirobotit, Teollisuusrobotiikka, Robottikalustonhallinta, Tehtävien aikataulutus, Suorituskyvyn seuranta

Contents

Abbreviations

AGV – Automated Guided Vehicle
AI – Artificial Intelligence
AMR - Autonomous Mobile Robot
API – Application Programming Interface
DDoS – Distributed denial-of-service
HMI – Human-machine interface
HTTP – Hypertext Transfer Protocol
IIoT – Industrial Internet of Things
IoRT – Internet of Robotic Things
IoT – Internet of Things
JSON – JavaScript Object Notation
KPI – Key Performance Indicator
LiDAR – Light Detection and Ranging
MiR – Mobile Industrial Robots
MRTA – Multi-Robot Task Allocation
PLC – Programmable Logic Controller
REST – Representational State Transfer
RFM – Robot Fleet Management
SLAM – Simultaneous Localization and Mapping
UI – User Interface
WMS – Warehouse Management System

1 Introduction

Autonomous Mobile Robots have become an increasingly important component of modern industrial and logistics systems. Unlike traditional automated guided vehicles that rely on predefined physical guidance infrastructure, AMRs can navigate dynamically in partially structured environments using onboard perception, localization, and decision-making capabilities (Kagan et al., 2019). This technological advancement has enabled more flexible material-handling solutions in industrial environments, warehouses, hospitals, and distribution centres.

The adoption of AMRs is closely connected to the broader development of Industry 4.0 and the increasing need for flexible, data-driven production systems. Industrial case studies show that although AMRs can improve workflow integration and reduce non-value-added tasks, their successful deployment depends on organizational readiness, system integration, and effective communication between suppliers and end users (Grover & Ashraf, 2024, pp. 1168-1169). Market analyses also show that the use of AMR solutions will continue to grow in various sectors, reflecting their strategic importance in modern automation ecosystems (Next Move Strategy Consulting, 2025).

The complexity of design and control increases considerably as AMR systems scale from individual robots to coordinated fleets. Intralogistics research has shown that multi-robot systems need coordinated decision-making across several interdependent planning domains, including fleet size, scheduling, dispatch, zone allocation, and route planning (Fragapane et al., 2021). A notable portion of the research has focused on optimization-based approaches to task allocation and production scheduling for multi-robot systems. In the manufacturing contexts, these approaches often aim to minimize manufacturing time under complex resource constraints (Shakeri et al., 2025), while broader MRTA research emphasizes scalable and robust coordination strategies, including dynamic task allocation (Chakraa et al., 2023).

At the same time, AMR fleets are operating in increasingly networked digital environments where robots exchange data with control systems, edge platforms, and other IoT components. The emergence of the Internet of Robotic Things (IoRT) emphasizes the convergence of robotics, IoT, artificial intelligence, and distributed edge/cloud computing, highlighting the importance of intelligent connectivity, interoperability, and platform-based integration in collaborative robotic ecosystems (Vermesan et al., 2020). In such environments, autonomous decision-making at the device level is complemented by distributed coordination mechanisms across edge and cloud layers.

Although significant progress has been made in algorithmic task allocation and traffic optimization, practical industrial applications often involve hybrid human-robot environments in which operators, planners, and supervisors interact with AMR fleets. Studies on the cooperation between humans and robots in intralogistics systems show that coordination between robots and human workers introduces further constraints on operations and scheduling (Löffler et al., 2023). Therefore, effective fleet management requires not only autonomous coordination mechanisms but also structured supervisory and monitoring tools that allow operators to influence and monitor the operation of the system.

These developments raise important questions about how commercial AMR systems support temporal task planning, operational flexibility, and performance monitoring at the supervisory level. While much of the academic literature focuses on optimization algorithms and decentralized allocation strategies, relatively little attention has been paid to practical scheduling extensions that improve the usability and operational management of deployed fleet management systems. The present thesis aims to address this gap by investigating supervisory-level scheduling support in commercial AMR environments through the design, implementation, and evaluation of a time-based mission planning application. The thesis is positioned within the field of Industrial Automation and Robotics, with a particular emphasis on the convergence of

autonomous mobile robotics and IoRT. It investigates the application of standardized communication protocols to enhance the functionality of proprietary robotic platforms.

1.1 Background and motivation

This thesis was conducted in collaboration with JTA Connection OY, a Finnish automation systems integrator delivering turnkey industrial automation solutions. In recent years, AMRs have increasingly been deployed both as components of larger automation systems and as stand-alone transport solutions in less infrastructure-intensive environments. Through practical AMR implementation projects across diverse deployment contexts, limitations in temporal mission scheduling and supervisory-level control were identified.

In highly automated industrial environments, AMR missions are often triggered by PLCs, warehouse management systems, or other higher-level background automation layers based on predefined conditions. In cases like these, AMR mission execution is typically reactive and event-driven. However, when such infrastructure is limited, unnecessary, or absent, task scheduling must be performed primarily using the AMR fleet management system's own features. In these situations, the lack of structured scheduling tools is even more pronounced, emphasizing the need for supervisory-level scheduling mechanisms that support recurring and time-based mission execution.

While many industrial deployments rely on event-driven triggering through external automation systems, time-driven and recurring mission patterns are less systematically supported within existing fleet management platforms. These observations form the practical motivation for the present thesis.

1.2 Objectives and research questions

The objective of this thesis is to improve the operational flexibility of MiR autonomous mobile robot systems by developing and deploying an external scheduling and performance monitoring application. The developed solution extends the capabilities of the existing fleet management system through REST API integration, enabling time-based mission scheduling and structured collection of task execution data. The work follows a constructive research approach, in which a practical problem observed in real-world implementations is solved through system design, implementation, and evaluation.

The aim of this thesis is to design a software application that enables time-based mission automation without modifying the internal functionality of the AMR system. The thesis also seeks to demonstrate how mission execution data retrieved from the robot can be structured and analysed to support performance tracking and operational assessment.

Based on these objectives, the thesis addresses several research questions. The first research question considers how time-based scheduling can technically be implemented in an AMR system that does not natively support this functionality. Secondly, the thesis examines performance metrics that can be extracted from the MiR API and how this data could be used to evaluate the success and efficiency of task execution. Another key question is which architectural and user interface design principles support practical deployment in industrial environments. Finally, the thesis considers whether external scheduling can improve operational flexibility or efficiency in real-world AMR deployments.

1.3 Scientific challenges and contributions

A significant challenge in modern industrial robotics is the reliance on off-board systems, such as external schedulers or Edge compute nodes, to manage robot operations via

wireless networks. This reliance introduces non-deterministic latency resulting from wireless channel fading and variable network traffic, which can disrupt the operational duty cycle or mission synchronization of robots. Although Eisen et al. (2023) addresses this issue at the navigation level through joint resource scheduling, a scientific gap remains in ensuring mission-level determinism when employing high-level communication protocols such as REST API. This thesis aims to reduce non-deterministic behaviour in mission execution by implementing a REST-based middleware that incorporates robust error handling and persistent logging to mitigate network-induced synchronization failures in AMR fleets.

The security and dependability of distributed robotic architectures are another significant challenge within industrial automation. As robotic systems evolve from isolated units to interconnected elements of larger networks, their susceptibility to diverse cyber-physical threats increases. The expanded connectivity of multi-robot systems, especially those that depend on cloud-based services for coordination or storage, substantially increases the potential attack surface. Consequently, industrial fleets become vulnerable to distributed DDoS attacks and other exploits that may disrupt mission-critical operations or result in severe real-world consequences (Papoutsakis et al., 2025, pp. 1-4). By implementing a localized REST API and an on-device SQLite database for data persistence, the proposed system eliminates dependence on external internet connections or cloud-based storage. This architecture ensures mission scheduling and performance tracking during external network outages and enhances resilience against security risks associated with extensive external connectivity.

Another central challenge in the advancement of Industry 4.0 is the shift from basic robot deployment to sustainable fleet management, which necessitates a comprehensive understanding of energy efficiency and operational traceability. Research by Dobrzanska & Dobrzanski (2025) identifies a significant deficiency in current systems: although robots efficiently transport materials, there is a lack of analytical tools for assessing long-term sustainability and information flow. This thesis addresses this

issue by implementing a persistent data logging system that extends beyond temporary real-time monitoring. By creating a structured record of mission durations and success rates, this research establishes a robust data foundation to support the objectives of sustainable fleet management and traceability, as outlined by contemporary researchers.

The scientific challenge of designing effective HMIs for robot fleet supervision requires balancing visual clarity with functional usability to minimize operator cognitive load. In complex industrial environments, UI design properties have a measurable impact on user experience and task performance. Research demonstrates that moderate border radii optimize the balance between visual appeal and cognitive ease, while the inclusion of defined object borders significantly improves user performance by providing clear spatial orientation and reducing the cognitive effort required to interpret interface elements (Baluyot et al., 2026, pp. 6-7). This thesis addresses these design challenges by employing an established Material Design library for the scheduling applications interface. Utilizing a library that incorporates these validated design principles ensures a high level of usability and reduces cognitive barriers for operators. This strategy illustrates how adopting established, modern design systems facilitates the practical and efficient deployment of complex robotic systems in industrial contexts.

1.4 Scope and limitations

The focus of this thesis is on the design and implementation of an external performance monitoring and scheduling application for MiR autonomous mobile robot systems. The application is limited to extending the existing fleet management capabilities using the API for MiR robots. The navigation algorithms, firmware of the MiR robots, or internal control logic are not modified.

The primary scope of the work consists of developing a time-based mission scheduling application that automates mission triggering and collects structured mission execution data. The application functions as a distinct software layer alongside the existing fleet management system. The purpose of the work is not to redesign fleet management

architectures, develop new path-planning algorithms, or introduce advanced optimization methods for multi-robot coordination.

The assessment of the developed application is based on functional tests and analysis of data obtained from tasks performed via the API. This thesis does not aim to make a comparative study between different AMR brands or fleet management platforms. Performance improvements are also evaluated within the implemented solution and available operational data, without conducting large-scale industrial trials across multiple customer environments. While the developed tool may be adaptable to other AMR systems that provide similar API access, the technical implementation and case study are limited to MiR platforms.

1.5 Structure of the thesis

The thesis is structured as follows. Chapter 2 introduces autonomous mobile robots and discusses scheduling, mission planning, and fleet management in AMR systems. Chapter 3 outlines the relevant technical background, including the MiR REST API and data storage technologies used in the implementation.

Chapter 4 describes the system design, including functional requirements, architectural decisions, scheduling logic, and approaches to data collection and performance monitoring. Chapter 5 details the implementation of the developed application, which includes the software architecture, API integration, mission logging mechanisms, and user interface design.

Chapter 6 covers the testing and evaluation of the system. This includes the testing environment, the implemented task scenarios, the selected key performance indicators, and the analysis of the collected data. Chapter 7 summarizes the results of the thesis, addresses limitations, and offers recommendations for further development.

2 Autonomous mobile robots

AMR systems execute tasks within dynamic and shared environments. Efficient task completion depends on the structured organization of both the timing and the order of operations. In this context, scheduling determines the timing and sequencing of task execution.

2.1 Overview of AMR technology

Autonomous robots are systems that perform tasks by continuously interacting with their environment through sensing, computation, and actuation. Unlike purely computational systems, these robots possess both internal and external sensing mechanisms, enabling them to monitor their internal state and respond to external conditions. (Kagan et al., 2019, pp. 25-26) The integration of sensing and actuation facilitates adaptive and goal-oriented behaviour.

AMRs are characterized by a high degree of operational independence, as they can make decisions and adapt to changes in their operational environment without continuous external control (Fragapane et al., 2021, pp. 405-406). Internal sensing and control mechanisms maintain system stability, whereas external sensing enables the robot to respond to environmental changes (Kagan et al., 2019, pp. 25-26). Consequently, robot behaviour results from ongoing interaction with the surrounding environment rather than being strictly predefined.

Mobile robots constitute a subclass of autonomous systems capable of movement within their operating environment. In contrast to fixed-base industrial robots, which generally operate in structured, predictable settings, mobile robots are engineered to operate in dynamic, partially unknown environments (Kagan et al., 2019, pp. 26-27). This necessitates ongoing perception, decision-making, and adaptation throughout operation (Fragapane et al., 2021, pp. 405-406).

Advancements in sensing, computation, and control technologies have significantly influenced the development of autonomous mobile robots. Contemporary AMRs are equipped with sensors such as LiDAR, cameras, and inertial measurement units, enabling real-time environmental perception. These technologies underpin essential functionalities, including localization, mapping, and obstacle avoidance, which are critical for autonomous navigation. (Fragapane et al., 2021, pp. 408-409). Robust obstacle avoidance forms the basis of intelligent traffic coordination, facilitating accident prevention and resolving potential deadlocks in factory environments. The complexity of shared workspaces necessitates capabilities beyond basic object detection; systems must accurately differentiate between human operators and stationary industrial machinery. Unlike fixed equipment, humans exhibit arbitrary movement, resulting in dynamic, unpredictable trajectories that increase the risk of collision, particularly at factory intersections (Oyekanlu et al., 2020, p. 202325). Consequently, the capacity of AMRs to identify and distinguish humans from other obstacles is essential for the safe and effective integration of autonomous fleets into contemporary manufacturing settings.

Motion planning is a fundamental capability of AMRs, involving the determination of feasible, collision-free paths between locations (Fragapane et al., 2021, p. 409). In familiar environments, motion planning uses predefined maps, whereas in dynamic or unknown settings, it relies on real-time sensor data. This process requires real-time sensor data and adaptation to changes, such as moving obstacles or congestion. (Kagan et al., 2019, pp. 33-35). The integration of real-time adaptations with long-term environmental awareness frequently relies on SLAM methods. These approaches enable AMRs to construct and continuously update environmental maps while tracking their position, which is essential for navigation in unknown or dynamic environments. Despite advancements, reviews of the current research indicate that high precision in SLAM and maintaining odometry accuracy remain significant challenges. These issues are often addressed through sensor fusion techniques and filtering methods that enhance data

robustness, thereby ensuring the reliability of environmental maps for motion planning in dynamic industrial settings (Soares & Alves, 2025, p. 25).

Although individual motion planning enables AMRs to navigate without physical infrastructure, task complexity increases substantially when multiple robots operate within the same environment. Due to the computational intensity of identifying a globally optimal solution for a multi-robot fleet, motion planning is frequently separated from robot coordination (Kozjek et al., 2021, pp. 1-2). Coordination strategies typically address this challenge by resolving conflicts online during execution or by preventing them offline through predefined movement constraints and rules. In industrial settings, implementing constraints, such as designated one-way systems in narrow corridors, is crucial to prevent deadlocks and ensure that the enhanced flexibility of AMRs does not result in systemic inefficiencies (Kozjek et al., 2021, p. 1).

In contemporary industrial applications, AMRs are extensively deployed in intralogistics settings, including warehouses, manufacturing systems, and hospitals. Compared to traditional automated guided vehicles, AMRs offer substantially greater flexibility by navigating freely, adapting to environmental changes, and making decentralized decisions. These capabilities enable more dynamic and scalable material-handling systems. (Fragapane et al., 2021, pp. 405-406).

2.2 Task scheduling in AMR systems

The operation of AMR systems necessitates structured planning and control across multiple decision levels. These systems comprise strategic, tactical, and operational layers, each responsible for real-time decisions regarding task execution, routing, and coordination. Within this framework, task scheduling is central to ensuring the efficient assignment and execution of missions in dynamic, evolving environments. (Fragapane et al., 2021).

2.2.1 Role of scheduling in AMR systems

In AMR-based intralogistics systems, scheduling is a central planning and control activity that links higher-level operational objectives to the execution of transport and service tasks. Decision-making in AMR systems occurs at multiple levels, with scheduling constituting a key area alongside resource management, dispatching, and path planning (Fragapane et al., 2021, pp. 405-406). The function of scheduling extends beyond the assignment of individual robot tasks to include the coordination of task execution within the broader operating system. Consequently, scheduling is essential to ensure that AMR operations support the required material flow, service responsiveness, and overall system performance in dynamic intralogistics environments (Fragapane et al., 2021, pp. 414-415).

The significance of scheduling is further highlighted when robots are integrated into complex production and logistics processes. In these settings, robots not only transport materials but also interact with machines, buffers, and other resources, whose availability directly influences operational timing. Scheduling must therefore coordinate task execution, resource utilization, and timing dependencies to minimize unnecessary waiting, blocking, and downtime. As robotic systems become more capable and production environments become more flexible, scheduling complexity increases, reinforcing its critical role in maintaining seamless operations. (Shakeri et al., 2025, pp. 1-2).

Operationally, scheduling directly affects efficiency and throughput. Effective scheduling supports better use of available resources, reduces delays between process stages, and maintains a steady flow of materials and tasks throughout the system. In Industry 4.0 environments, the need to adapt schedules in response to real-time system conditions further underscores the importance of scheduling (Shakeri et al., 2025, pp. 1-2). Therefore, scheduling should be regarded not only as a technical optimization problem

but also as a practical control mechanism that enables AMR systems to achieve greater operational flexibility and more consistent performance.

The efficiency of scheduling is increasingly dependent on the joint optimization of production sequencing and robotic logistics. Minimizing the total makespan of a production batch requires precise synchronization between job-to-machine assignments and the routing of AMRs that service these machines. Since these processes are highly interdependent, treating transport as a reactive service, separate from machine-level job sequencing, often results in a systemic idle time and reduced throughput. Therefore, modern scheduling frameworks must address the complex synchronization of routing, job-to-AMR assignments, and machine sequencing as a unified optimization problem to achieve the performance standards necessary for fully automated factories (Pang & Zhen, 2024).

2.2.2 Task allocation vs scheduling

Although task allocation and scheduling are closely related within AMR systems, they represent distinct decision problems. In multi-robot systems, task allocation determines which robot, or group of robots, should be assigned to a specific task. Task allocation functions as a supervisory-level process that distributes work among available robots based on task requirements, robot capabilities, and system constraints (Chakraa et al., 2023, pp. 2-3). This distinction is also evident in broader multi-robot control frameworks, where task allocation is considered a separate decision stage alongside task planning and motion planning (Valero et al., 2023, p. 2).

In contrast, scheduling addresses the timing and sequencing of task execution rather than assigning tasks to specific robots. This distinction becomes particularly significant when planning horizons, temporal constraints, or task dependencies must be considered. Chakraa et al. (2023) differentiate between instantaneous assignment and time-extended assignment, where a sequence of tasks is assigned over a defined planning horizon (Chakraa et al., 2023, pp.6-7). As decision-making begins to incorporate elements such as time windows, precedence relations, and synchronization

requirements, task allocation problems extend beyond simple assignment and increasingly resemble conventional scheduling.

In practice, task allocation and scheduling are strongly interconnected. A robot cannot execute a task without first being assigned to it, but effective allocation alone does not ensure optimal system performance if task sequencing, timing, and resource conflicts are not addressed. Consequently, research often considers these decision problems in a broader system context, particularly in complex multi-robot environments where coordination between machines, transport resources, and other system elements must be managed concurrently (Fragapane et al., 2021, pp. 414-415). Nonetheless, maintaining a conceptual distinction remains valuable: task allocation governs the distribution of work among robots, while scheduling organizes the temporal execution of these tasks.

2.2.3 Scheduling approaches in AMR systems

Scheduling in AMR systems is not an isolated decision problem but forms part of a comprehensive planning and control framework. The autonomy of AMRs necessitates continuous decision-making across interconnected domains such as task allocation, resource management, and path planning, all of which must function cohesively in dynamic environments. These decisions occur at strategic, tactical, and operational levels and are increasingly distributed due to the decentralized structure of AMR systems (Fragapane et al., 2021). Consequently, scheduling is intrinsically connected to other control processes, and its effectiveness relies on seamless integration within the broader decision-making system.

Effective task execution in multi-robot environments relies on coordination among robots operating within shared spaces. Simultaneous operation of multiple AMRs can result in task interference or deadlock in confined areas, thereby decreasing system efficiency. To mitigate these issues, traffic management mechanisms are implemented

to regulate robot behaviour. These mechanisms include restructuring access to specific areas, enforcing directional movement rules, and controlling entry to shared regions to ensure exclusive occupancy. Such strategies facilitate coordinated operation and prevent conflicts during task execution (Jeong et al., 2022).

System-level coordination is often facilitated by centralized fleet management platforms that operate in a supervisory capacity. These platforms provide a comprehensive overview of robot status, location, and task progress, enabling fleet-wide monitoring and coordination. In these architectures, low-level operations such as path planning, SLAM, and obstacle avoidance are managed internally by each robot, while higher-level supervision is conducted externally. This centralized approach is particularly critical in heterogeneous environments, where robots from various manufacturers employ different data formats and interfaces, necessitating standardized integration and centralized oversight (Lopes et al., 2025).

2.2.4 Practical considerations and limitations

Implementing AMR systems in warehouses presents challenges beyond technical capabilities. Although AMRs offer significant benefits, organizations often face difficulties with system selection, integration, and performance evaluation. Additional challenges include organizational readiness, infrastructure limitations, safety and facility requirements, and reliance on external technology providers (Grover et al., 2024, pp. 1168-1169). Therefore, the effectiveness of AMR systems and their scheduling approaches depends on both optimization performance and broader organizational and operational factors.

Optimization-based scheduling and task allocation methods also face important technical limitations in multi-robot systems. As the number of robots, tasks, and environmental constraints increases, the complexity of the problem escalates, rendering exact optimization impractical for large-scale or time-sensitive applications. Although

many approximate methods enhance computational feasibility, these approaches may compromise optimality or prove inadequate for highly dynamic environments (Chakraa et al., 2023, pp. 18-20).

Operational constraints significantly limit the effectiveness of scheduling approaches in real-world AMR systems. System performance depends not only on optimization strategies but also on organizational factors, including workforce configuration and robot-to-operator ratios. Variability in task execution times and unforeseen delays can propagate throughout the system, causing ripple effects that degrade overall performance and are only partially addressed by real-time rescheduling. Physical constraints, such as congestion in shared spaces, further complicate coordination by restricting robot movement and increasing the likelihood of interference. Operational design choices, such as zoning, may reduce system flexibility by introducing additional constraints, although their impact on overall system performance is generally limited, thereby highlighting a trade-off between organizational structure and scheduling efficiency (Löffler et al., 2023, pp. 996-997).

2.3 Fleet management and MiR platform capabilities

The operational effectiveness of mobile robotics is largely dependent on RFM, a framework that enables the autonomous coordination of multiple robotic units through advanced algorithmic oversight. Instead of individualized configuration or manual human intervention, RFM systems employ sophisticated software architectures to manage collective tasks within shared industrial environments. These systems generally employ centralized, decentralized, or hybrid control models to synchronize essential functions such as resource management, task allocation, and real-time communication across both homogeneous and heterogeneous fleets (Morais et al., 2025, p. 118976)

Fleet management within AMR systems is increasingly recognized as a critical factor in optimizing processes within industrial and logistical environments. Existing approaches

focus on improving task distribution, movement coordination, and inter-robot communication to improve overall system efficiency and adaptability. In multi-robot systems, optimization and learning-based techniques facilitate dynamic task allocation and collision avoidance, enabling more effective responses to changing operational conditions. Decentralized communication strategies further contribute to scalability and resilience in fleet coordination. Integrating mobile robots with smart infrastructure, such as sensor-equipped environments, enhances environmental awareness, routing efficiency, and congestion mitigation. Collectively, these advancements underscore the significance of fleet management systems in enabling efficient and adaptive AMR operations (Lopes et al., 2025, p. 5).

In addition to internal coordination, fleet management systems must facilitate interaction with external systems and software components. Contemporary robotic systems are increasingly integrated into broader digital environments, exchanging information with other systems, devices, and users (Vermesan et al., 2020). This requires a mechanism for communication and data exchange that enables robotic systems to interface with external applications and operational infrastructure. Standardized interfaces are typically employed to provide access to system state, task execution, and control functions. These interfaces enable external systems to monitor, coordinate, and influence robot operations, laying the foundation for advanced control and comprehensive system integration that extends beyond core fleet management functionalities.

Commercial AMR platforms, such as MiR, offer fleet management capabilities through dedicated software interfaces, with the MiR Fleet Manager being a prominent example. The MiR Fleet Manager provides a web-based user interface that mirrors the functionality available on individual robots, enabling users to configure missions, monitor robot statuses, and manage task execution across multiple units. At the fleet level, a built-in task allocator coordinates task execution by distributing missions among available robots according to current system conditions. Although this approach

supports efficient real-time operation and resource utilization, mission execution is primarily triggered either manually or in response to external events. The system lacks native support for time-based or time-driven task scheduling. Furthermore, the user interface functionality is accessible via REST API, which allows external systems to interact with and control fleet operations. This extensibility enables the integration of higher-level supervisory controls, such as time-based scheduling, which are not supported by default.

3 Technical background

The developed application utilizes technologies that enable communication with the robot system, implement scheduling logic, and support user interaction through a dedicated interface. Collectively, these components constitute the technical foundation of the system.

3.1 MiR REST API overview

The MiR platform provides a REST-based API that enables external systems to interact with robot functionality over a network. This interface exposes the robot's capabilities as structured resources that can be accessed and controlled via standard HTTP requests. The API facilitates structured data exchange and allows external applications to retrieve system information and issue control commands.

A key feature of the interface is its support for mission execution through a queue-based mechanism. External applications can submit missions to the robot, which are then executed according to the internal queue logic. Beyond mission initiation, the interface grants access to the current mission queue, enabling external systems to monitor execution status, identify completed or failed tasks, and manage queued operations as needed. Consequently, the API serves as both a control interface and a source of execution data for supervisory functions.

The API exposes a range of system-level information necessary for integration with external applications. This includes access to robot status, operational parameters, and register values, as well as communication with other systems such as PLCs. The interface also provides access to configuration elements, including missions, positions, and maps, enabling external applications to reference and utilize existing system definitions when implementing control logic.

3.2 Application development with Python and KivyMD

The application developed in this work was implemented in Python, utilizing the KivyMD framework for the user interface. Python was chosen for its simplicity, readability, and extensive library ecosystem, which enables efficient development of network-based applications. In this context, Python provides essential tools for managing HTTP communication with the MiR REST API, processing scheduling logic, and handling local data storage.

The user interface was developed using KivyMD, a Python-based framework that extends the Kivy library with Material Design components. KivyMD enables the creation of cross-platform applications with a consistent visual structure, making it suitable for deployment on a tablet device used as a dedicated control interface. The framework provides prebuilt UI elements, such as navigation bars, dialogs, and lists, that support the development of an operator-facing application with clear, well-organized interaction patterns.

The integration of Python and KivyMD enables the application to combine user interaction, scheduling logic, and communication with external systems within a single environment. The user interface layer is directly linked to the core application logic, allowing user actions, such as scheduling missions or initiating tasks, to be translated into API requests directed to the robot fleet. This integration supports an architecture in which the application functions as a supervisory layer on top of the existing fleet management system, extending its capabilities without altering the underlying robot software.

3.3 Data storage with SQLite

Local data storage within the application is implemented using SQLite, an embedded relational database engine optimized for lightweight and serverless operation. SQLite was chosen for its simplicity and suitability for standalone applications, as it eliminates the need for a separate database server and uses file-based storage. These characteristics make SQLite particularly suitable for deployment on tablet devices, where minimal system dependencies and straightforward setup are essential.

In the developed application, SQLite stores configuration data, scheduled missions, and historical execution records. This includes information such as selected robots, mission parameters, scheduled execution times, and logs of completed or failed missions. Local data storage enables the application to preserve its state across restarts and maintain functionality without dependence on external infrastructure.

Employing a relational database structure facilitates efficient management of scheduling and execution data within the application. This approach is essential for organizing upcoming missions, maintaining a record of completed tasks, and calculating performance metrics. Due to the relatively low data volume, SQLite delivers adequate performance while preserving a simple system architecture. This supports the design objective of creating a lightweight, self-contained supervisory application that can operate independently alongside the existing fleet management system.

4 System design

The development of the scheduling and performance monitoring application required translating defined functional requirements into a structured and modular system design. The primary design goal was to extend the temporal scheduling capabilities and structured performance monitoring without modifying the robot's internal control logic.

The design process followed a constructive development method, in which identified functional constraints were converted into functional requirements and then into architectural components. The design approach emphasizes separation of concerns between scheduling logic, communication handling, data persistence, and user interaction. This separation ensures that time-based decision-making remains independent of the robot's internal navigation and fleet coordination mechanisms, while remaining fully compatible with them through standardized API communication.

Particular attention was paid to ensuring predictable performance under varying operating conditions. The system must be able to handle delayed communication, temporary connection interruptions, and mission queue interactions without disrupting the robot's operation. At the same time, the architecture must support the permanent storage of configuration data and execution history to enable performance evaluation.

4.1 Functional requirements

The functional requirements of the developed application are derived from observations during practical deployment and the objectives defined in Chapter 1. The system was designed to extend supervisory-level control over AMR operations without modifying the internal logic of the robot platform. The requirements can be grouped into mission triggering functionality, scheduling, supervisory control, data storage, and performance monitoring.

4.1.1 Mission triggering and scheduling

The application was required to support two distinct methods for mission triggering. First, operators must be able to manually trigger missions using a “Run Now” functionality, requiring the operator to select only the robot and the mission to be executed. This feature allows users to run missions directly from the application without scheduling.

Second, the system must support time-based mission scheduling. Users can define scheduled executions by specifying the target robot, mission, execution date and time, and recurrence type. Recurrence options include one-time execution, daily repetition, and weekly repetition. In addition to time-based scheduling, the system must support idle-triggered automation, where a predefined mission is automatically initiated if a robot remains idle for a configurable duration between 5 and 30 minutes.

Scheduling precision was defined at a minute-level accuracy. Because the system relies on periodic polling of robot status and schedule conditions, the polling interval must be configurable, with a default value of 30 seconds. This design ensures acceptable timing accuracy while maintaining predictable system load.

The application must also handle execution contingencies. If communication with a robot fails at the scheduled execution time, the application shall retry mission submission up to a configurable retry limit. If the retry limit is exceeded, the scheduled mission is marked as failed. If the robot is already executing another mission at the scheduled time, the new mission must be submitted to the robot’s mission queue to ensure compatibility with the robot platform’s internal queue-handling mechanism.

Scheduling is done on a robot-by-robot basis, meaning each scheduled mission is linked to a single robot, even though multiple robots can be configured in the application.

4.1.2 Supervisory control requirements and performance monitoring

The application must provide visibility and control at the supervisory level in addition to scheduling functions. Users must be able to view upcoming scheduled missions, check the current mission queue, and view recently completed missions.

The application must also enable basic operational control functions, such as resetting errors on the robot, clearing the mission queue, and pausing or resuming robot execution. These features enable the application to serve as a lightweight supervisory interface in environments without full-scale automation infrastructure.

In addition, the scheduling logic must support configurable scheduling constraints. These constraints include an optional minimum battery level requirement and restrictions based on selected working days. These constraints allow the scheduling logic to better reflect operational conditions and safety considerations.

The application must also compute and present key performance indicators derived from the data obtained from mission execution monitoring. These include mission success rate, average mission duration, total mission duration per robot, average number of missions per day, and comparison of robot utilization rates. The KPI layer extends the system beyond simple mission automation and supports data-driven performance evaluation.

4.1.3 Mission logging and data collection

A key functional requirement of the application is the structured collection of mission data. For each mission triggered through the application, the application must record the robot ID, mission ID, start time, end time, and execution status.

Mission status and execution progression are monitored through periodic polling of the robot via the API. To manage local storage usage, the system must retain the most recent 100 mission records and automatically remove older entries. All configuration, scheduling data, and mission history are stored locally within the application environment, enabling operation without external cloud infrastructure.

4.1.4 Non-functional requirements

The application is intended for use on a separate tablet device operating on the same local network as the robot. It is designed for local use without authentication mechanisms. The application must retain scheduling and configuration data after the application is restarted and restore the scheduled mission upon restart.

4.2 System architecture

The developed application follows a layered architecture that separates user interaction, scheduling logic, communication with the robot platform, and data storage. This structure allows the scheduling function to operate independently of the robot platform’s internal control mechanisms.

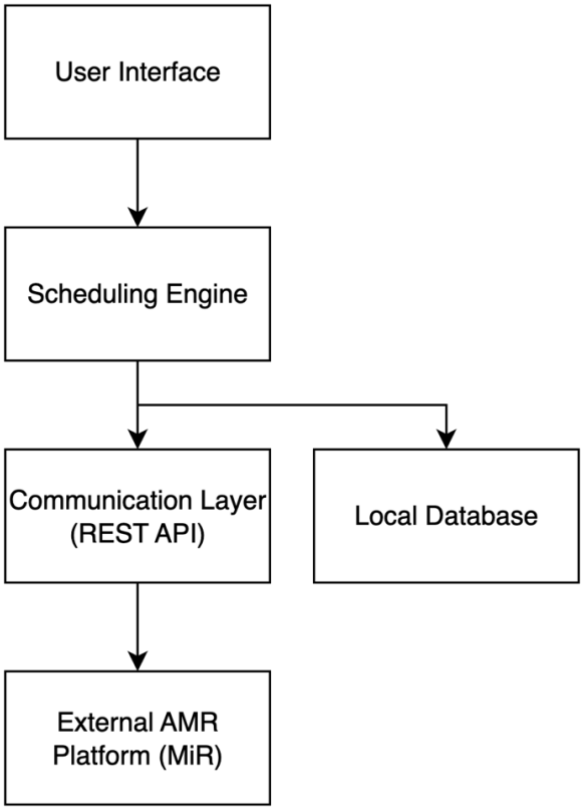

Figure 1. System architecture of the developed AMR scheduling application

The application consists of four conceptual components: a user interface, scheduling logic, a communication layer, and a local data storage layer. The general architectural structure of these components is shown in Figure 1.

The user interface allows the operator to configure robots, define missions, create schedulers, and observe system performance indicators. The scheduling logic periodically evaluates stored schedule definitions using a configurable polling interval and determines whether the execution conditions are met. When a scheduled mission is ready to be executed, it is forwarded to the communication layer.

Communication with the AMR platform is handled via REST API requests. The communication layer sends missions to the robot's mission queue and retrieves the

robot’s status information and mission execution status. Configuration data, scheduling information, and mission execution history are stored locally in a SQLite database.

4.3 Scheduling logic and time-based mission execution

The scheduling mechanism enables the robots to execute missions automatically based on predefined temporal conditions. Unlike event-based automation systems, where external controllers trigger missions, the developed scheduler introduces time-based decision-making directly within the supervisory application.

Scheduled missions are stored in a local database along with their execution parameters, including the target robot, mission ID, execution time, and recurrence type. The scheduler periodically evaluates these variables. During each evaluation cycle, the current system time is compared to the defined scheduling conditions to determine whether the mission should be started.

When the scheduled execution time is reached, the scheduler checks for additional constraints before starting the mission. These constraints include a minimum battery percentage on the robot, and weekdays, which are determined as working days by the user. This validation step ensures that scheduled missions are executed only when operational conditions permit safe and effective robot operation.

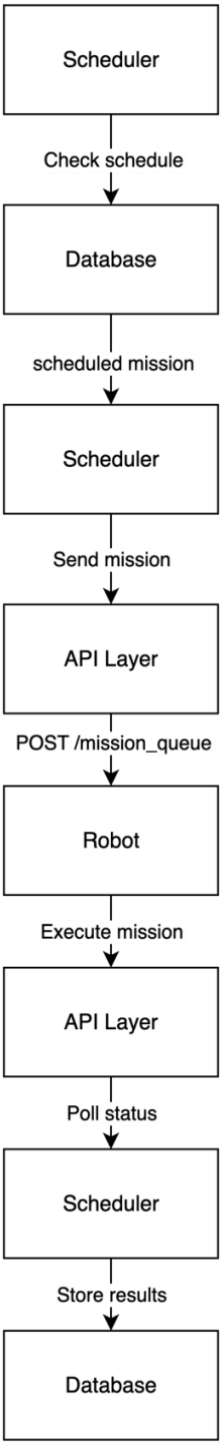

Figure 2. Sequence diagram of scheduled mission execution.

If the required conditions are met, the scheduler forwards the mission request to the communication layer. The mission ID is then sent to the robot via REST API, which adds the mission to the robot's internal mission queue. After sending the mission, the

scheduler continues to monitor the execution status with regular API queries. The mission start time, end time, and execution status are stored in a local database for historical tracking and performance monitoring. The interaction between different components of the application is illustrated in Figure 2.

4.4 Data collection and KPI mapping

In addition to supporting automated mission scheduling, the developed application collects operational data from executed missions to facilitate performance monitoring and system evaluation. Mission execution data is obtained from the robot platform through periodic API polling and stored locally for subsequent analysis. This process allows the application to maintain a historical record of robot activity, forming the foundation for calculating operational performance indicators.

For each executed mission, the system records the robot identifier, mission identifier, mission start time, mission completion time, and final execution status. These records enable the reconstruction of the execution history of scheduled missions and the evaluation of mission completion outcomes. The data is stored locally within the application environment and maintained as a rolling history to optimize storage management.

Several key performance indicators are derived from this execution history to deliver insights into robot operation. These indicators include the mission success rate, which represents the proportion of successfully completed missions relative to failed executions, the average mission duration, the total mission execution time per robot, and the average number of missions completed each day. Comparing robot usage also helps spot differences in how work is shared among the robots.

Converting raw mission execution data into structured performance indicators allows the application to provide operators with a clear overview of system performance. These

indicators assist supervisors in monitoring robot activity, identifying operation inefficiencies, and evaluating the effectiveness of the scheduling system utilizing available robotic resources.

5 Implementation

The scheduling application was implemented as a standalone supervisory tool that operates alongside the existing AMR system. The implementation follows the architectural structure outlined in Chapter 4 and translates the specified system components into a functional software application.

Developed in Python, the application integrates with the robot platform via the MiR REST API. It offers a graphical user interface for robot configuration, mission scheduling, and operational performance monitoring. Internally, the application comprises a scheduling mechanism that evaluates time-based execution conditions, a communication layer for platform interaction, and a local data storage layer for persisting configuration data and mission execution history.

5.1 Application structure and code overview

The application is implemented as a Python program structured into several modules responsible for user interaction, mission scheduling, robot communication, and data storage. The source code is organized into distinct packages that group related functionality, such as user interface screens, service logic, and database operations.

The application entry point is defined in the main module, which initializes the local database, loads the graphical user interface, and launches background processes responsible for robot status monitoring and scheduled task execution. The graphical user interface, developed with KivyMD, is organized into multiple screens that support mission scheduling, operational control, system configuration, and access to execution logs.

Mission scheduling and robot interaction are handled by service modules that evaluate stored schedules and communicate with the robot platform via API requests.

Configuration data, mission schedules, and execution history are stored locally to ensure persistent operation and enable performance monitoring.

5.1.1 Project structure and module organization

The application’s source code is organized into several Python packages, each grouping related functionality. The main project structure is shown in Figure 3. User interface layouts are defined separately using Kivy language files located in the Kivy directory. These files specify the visual layout and widget hierarchy for each application screen.

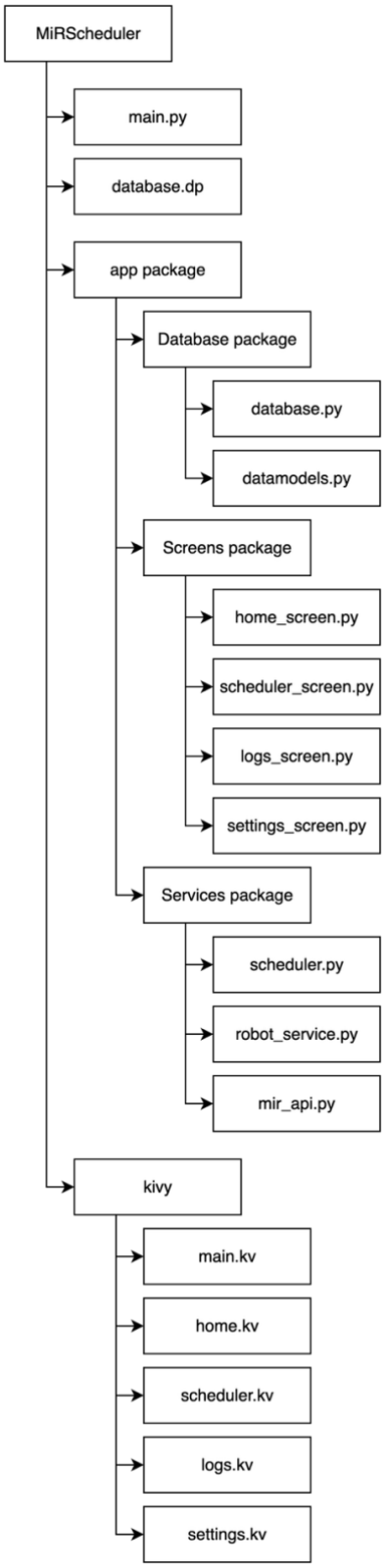

Figure 3. Project structure of the MiR scheduling application.

The core application logic resides in the app package, which includes modules for database access, user interface screens, and service-level logic. The database package manages interactions with the local SQLite database, including storage of robot configuration, scheduled missions, and execution history. The screens package implements controller logic for the graphical user interface, with distinct modules for the home screen, scheduler interface, system settings, and execution logs.

Operational functionality is implemented in the services package, which contains modules for mission scheduling, robot communication, and interaction with the robot platform API. The scheduling component evaluates stored schedules and determines when the mission should be executed. Robot service modules manage communication with the robot system.

5.1.2 Scheduler implementation

Mission scheduling utilizes a periodic polling mechanism to evaluate stored scheduling conditions and determine mission execution eligibility. The scheduler runs a background process, polling at a configurable interval. In each cycle, the application triggers the scheduling mechanism and launches the scheduler logic in a separate background thread. The simplified triggering mechanism of the scheduler is illustrated in Algorithm 1.

```
function start_scheduler_polling():
    poll_interval = read_scheduler_settings()

    schedule_repeating_event(
        callback = scheduler_tick,
        interval = poll_interval
    )

function scheduler_tick():
    if scheduler_running = true:
        return

    scheduler_running = true
    start_background_thread(run_scheduler)

function run_scheduler():
    poll_scheduler()

    schedule_on_ui_thread(reset_scheduler_flag)

function reset_scheduler_flag():
     scheduler_running = false
```

Algorithm 1. Scheduler execution triggering mechanism.

Prior to evaluating scheduled missions, the scheduler assesses idle automation conditions. Idle automation triggers predefined missions when a robot remains in a steady state for a specified duration without executing missions. If the configured idle timeout is reached, a new mission is scheduled for execution.

For each scheduled mission, the scheduler verifies that the execution time has been reached and that any optional execution constraints are satisfied. Such constraints may include working-day restrictions or other system configuration parameters. Once a mission is eligible for execution, the scheduler attempts to submit the task to the robot via the robot service module.

```
function poll_scheduler():
    check_idle_automation()

    schedules = get_upcoming_scheduled_missions()

    for schedule in schedules:
        if current_time ⩾ schedule.execution_time:
            execute_scheduled_mission(schedule)


function execute_scheduled_mission(schedule):
    result = queue_robot_mission(schedule)

    if result = failure:
        increment_retry_count()

        if retry_limit_reached:
            record_failed_mission()
            schedule_next_occurrence()
    else:
        track_mission_execution()
   schedule_next_occurrence_if_needed()
```

Algorithm 2. Scheduler execution logic.

If the mission submission fails, the scheduler increments the retry counter for the scheduled task. The scheduler continues to retry mission submission until a configurable retry limit is reached. If the retry limit is exceeded, the mission is marked as failed and recorded in the execution history. The simplified execution logic is illustrated in Algorithm 2.

For missions with recurring execution patterns, the scheduler automatically calculates the next occurrence of the task. The subsequent execution time is determined based on the defined recurrence interval, such as daily or weekly execution. After scheduling the next occurrence, the completed scheduling entry is removed from the database.

5.1.3 Communication and service layer

A dedicated service layer facilitates communication with the robot platform by separating application logic from direct API interaction. This layer provides functions for submitting missions, retrieving robot status, and monitoring mission execution. Encapsulating robot communication within service modules enables the application to interact with the robot platform through a consistent interface.

The robot service module initiates mission execution by submitting mission requests to the robot's internal mission queue via REST API calls. When the scheduler determines that the mission should be executed, it sends the corresponding mission identifier to the robot through the service layer. The robot then manages mission execution within its internal mission management system.

The service layer is also responsible for retrieving robot state information required by the application. Status data, including connection state, battery level, operational state, and mission queue status, is periodically obtained from the robot platform. This information supports both user interface updates and the evaluation of conditions necessary for scheduling logic.

5.2 API integration with MiR

Interaction with the robot platform is implemented through the REST API provided by the MiR system. The application communicates with the robot over the local network by sending HTTP requests to the robot controller and receiving structured JSON responses. The interface allows external applications to control mission execution and retrieve operational data without requiring modifications to the robot's internal software.

API communication is encapsulated in a dedicated interface module that handles request formatting, authentication, and response processing. This abstraction separates

low-level HTTP communication from application logic and provides a consistent interface for the rest of the system.

Mission execution is initiated by submitting mission identifiers to the robot's internal mission queue via the API. When the scheduler determines that a mission should be executed, the application sends a request to the robot controller with the relevant mission identifier. The robot subsequently places the mission in its internal queue and manages its execution according to its mission-handling logic.

Beyond mission submission, the application periodically retrieves robot status information through API requests. Retrieved data includes connection state, battery level, and mission queue details. This information is used to update the user interface and to evaluate scheduling conditions, including robot availability and battery constraints.

Communication with the robot occurs through authenticated HTTP requests using credentials specified in the application's settings. The service layer processes API responses, converting the returned data into application-level information for use by the scheduler and user interface components.

5.3 User interface development

The user interface was developed to offer a practical supervisory environment for robot configuration, mission scheduling, execution monitoring, and performance data review. As the application is intended to function as a continuously available local control tool, the interface supports rapid access to essential functions and presents operational information in a clear, structured format.

The interface consists of four primary views: Home, Scheduler, Settings, and Logs. These views collectively facilitate daily operations and system configuration by integrating real-

time monitoring of robot status, mission control, scheduler management, parameter adjustment, and access to execution history.

5.3.1 Interface architecture

The graphical user interface is developed using the KivyMD framework, enabling cross-platform development with a consistent visual style. The layout is specified in Kivy language files, while the application logic resides in Python-based screen controllers. This separation ensures that the visual structure and interaction logic are maintained independently.

The interface employs a screen-based navigation model managed by a central screen manager that controls the active view and allows users to switch between functional areas. Navigation is facilitated by a bottom navigation bar that provides direct access to the primary screens: Home, Scheduler, Logs, and Settings. This structure enables quick transitions between operational views without the need for hierarchical menus.

A top application bar displays global information and controls that remain accessible on all screens. This bar includes the current system time and an indicator of the selected robot's connection status. Consistent presentation of this information across all views supports continuous monitoring of the system state during operation.

5.3.2 Operational interface screens

The application’s operational functionality is structured around two primary views: the Home screen and the Scheduler screen. These views offer real-time system visibility and facilitate both manual and scheduled mission control. The interface is designed to present essential information alongside relevant control actions within a single view, thereby minimizing navigation requirements during operation.

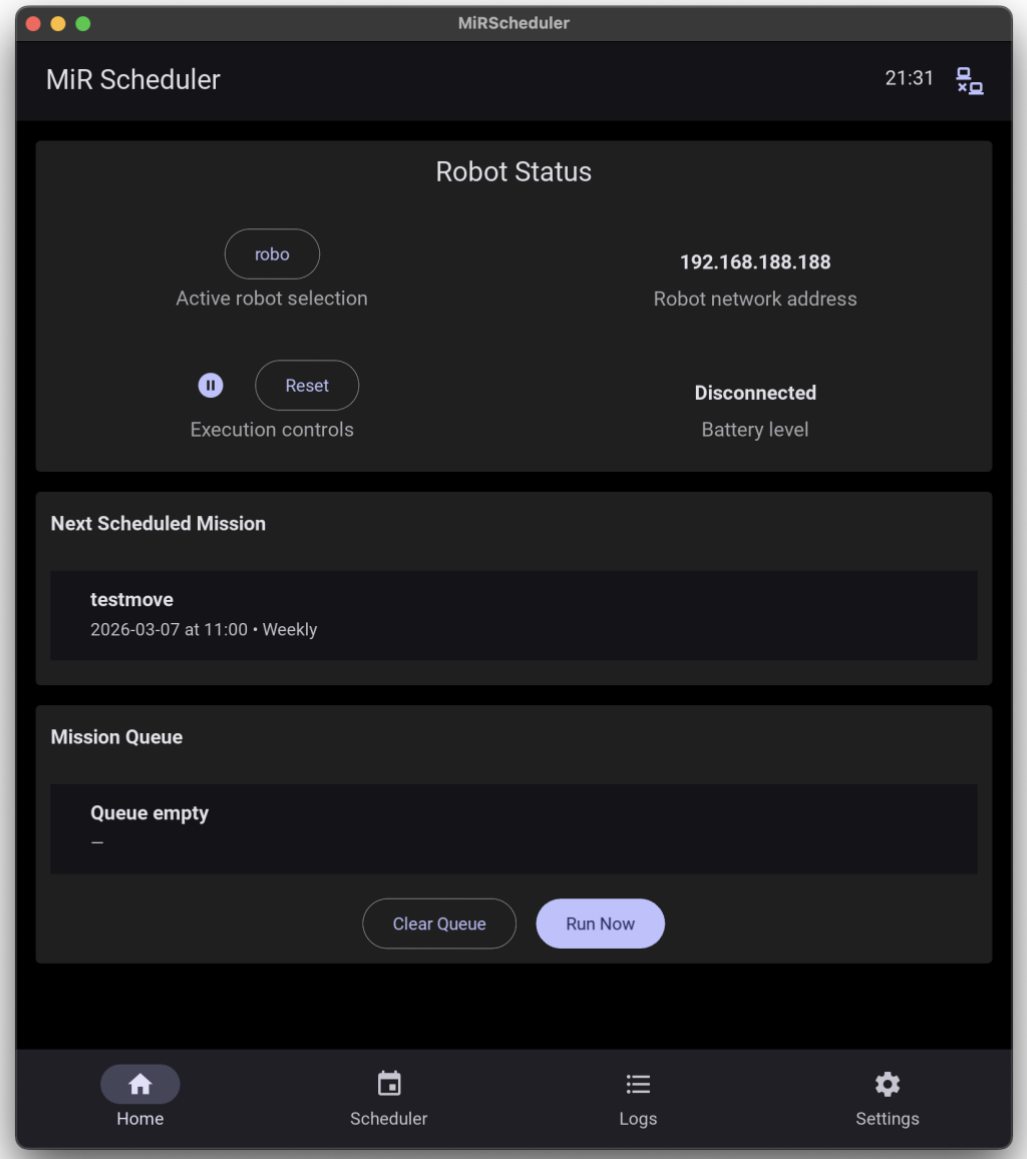

Image 1. Home screen interface

As shown in Image 1, the Home screen displays the selected robot's current operational state and provides direct control functions. The interface incorporates a robot selection component, enabling the operator to switch between configured robots. Key status indicators, including network address, connection state, and battery level, are prominently displayed to facilitate rapid assessment of system readiness.

Execution controls are integrated into the same view, allowing the operator to pause or resume robot operation and reset the system when required. The mission queue appears in a dedicated section, showing all missions in the selected robot's internal queue, whether initiated through the application or directly from the robot interface. A "Run now" function enables immediate execution of selected missions without scheduling. Combining status monitoring, queue visibility, and control functions on a single screen supports efficient operation and reduces interaction complexity.

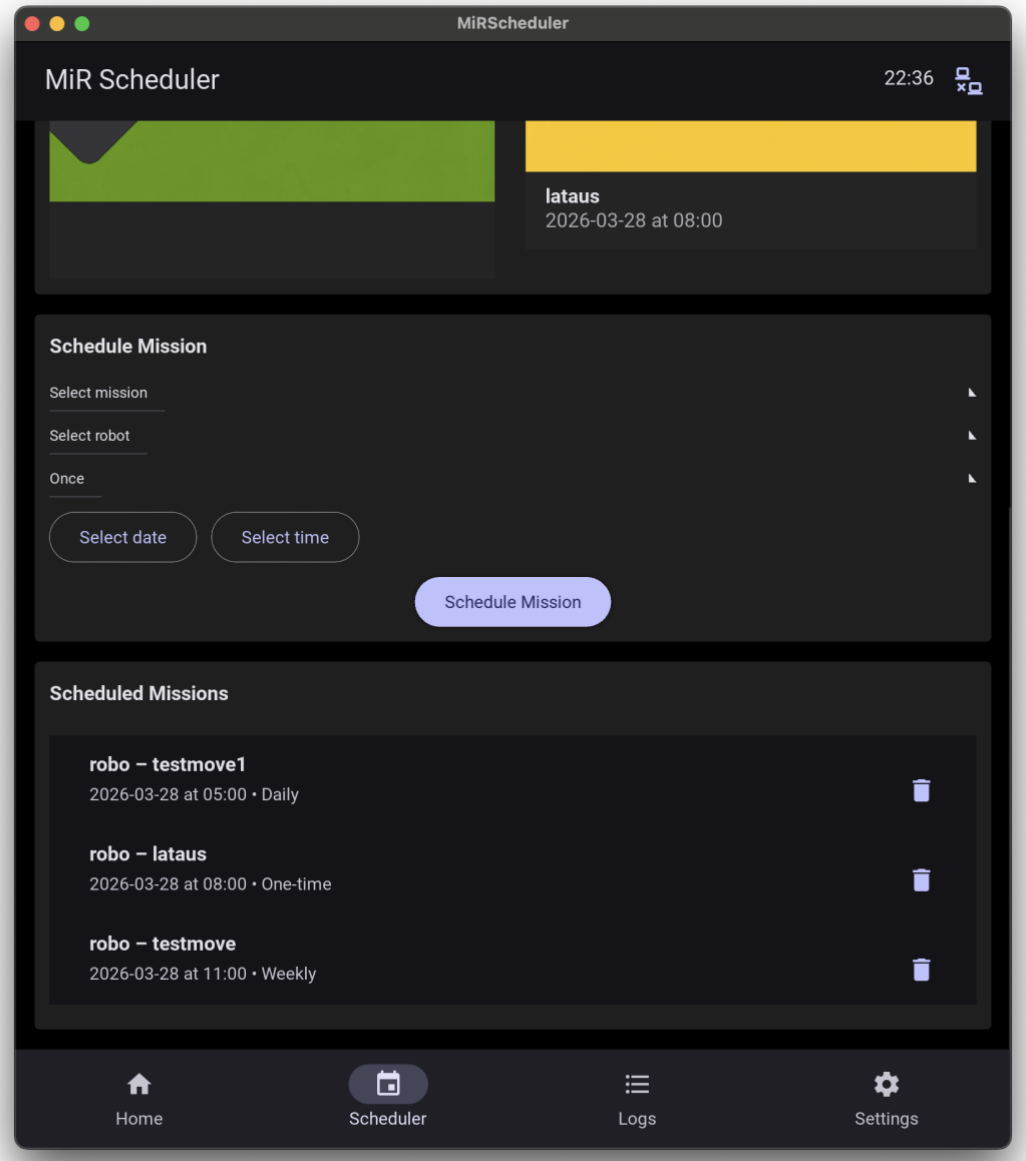

Image 2. **Mission scheduler interface**

The scheduler screen, shown in Image 2, provides functionality for creating and managing time-based mission execution. The interface is divided into three main sections: upcoming missions, schedule definition, and scheduled mission list.

New scheduled missions are created using a structured input form that includes mission selection, robot selection, execution time, and recurrence options. Placing scheduling control adjacent to the list of existing schedules enables immediate verification that newly created tasks have been stored correctly. Scheduled missions are displayed in a list format, allowing them to be reviewed and managed as required. The layout of the Scheduler screen prioritizes clarity by grouping related inputs and outputs within a single view. This approach reduces the likelihood of configuration errors and supports efficient creation and management of scheduled tasks.

5.3.3 System configuration and logs

The application offers dedicated interfaces for system configuration and performance monitoring via the Settings and Logs screens. These views facilitate system setup, parameter adjustment, and evaluation of mission execution.

The settings screen enables management of robots, missions, and scheduler parameters. The interface allows the operator to configure robot connection details and define available missions for the scheduling system. These elements constitute the foundation for all scheduling operations within the application.

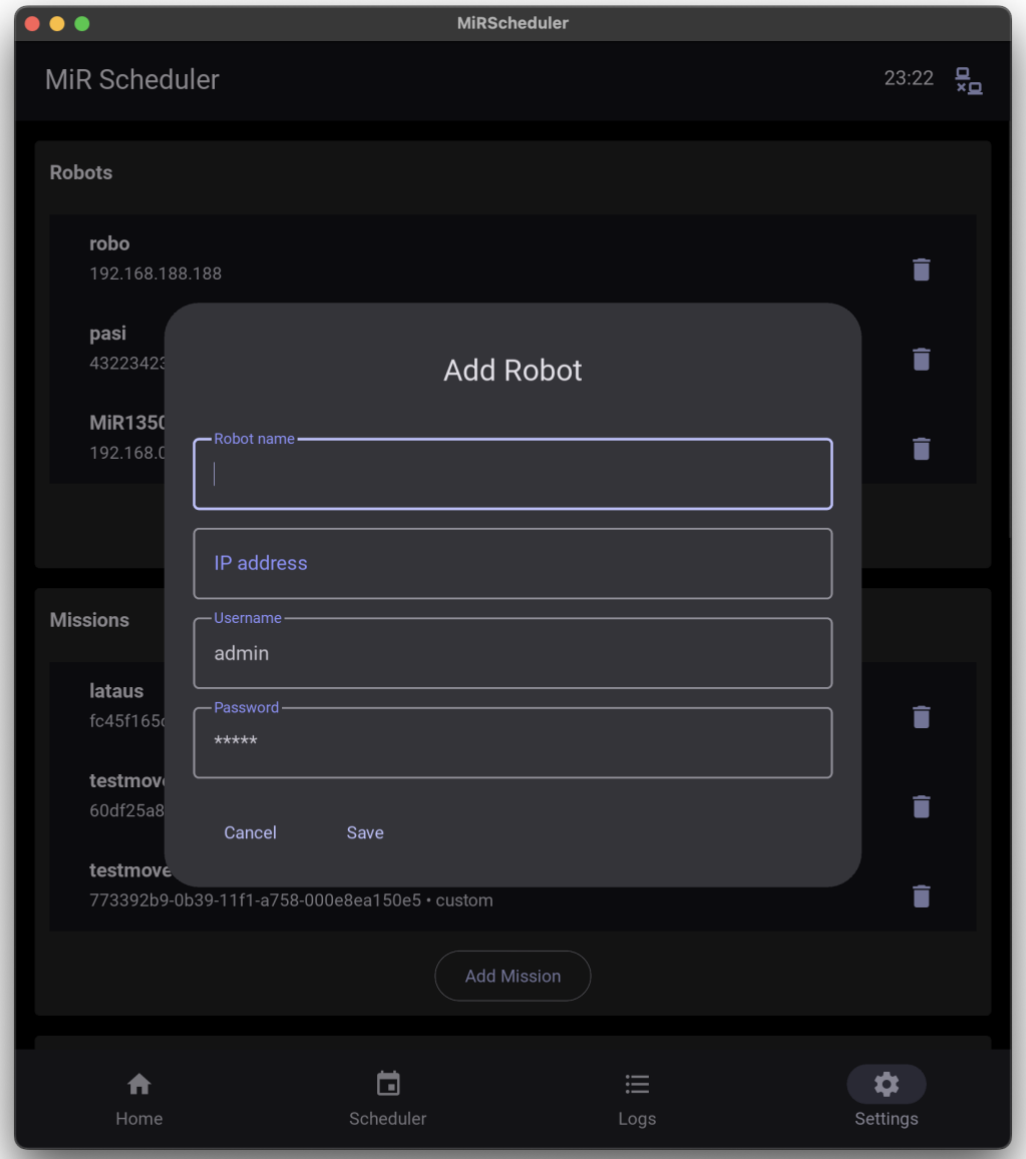

Image 3. **Robot parameter configuration window**

Configured robots and missions are displayed as lists within the interface, with entries that can be added or removed directly. New entries are created through dialog windows that prompt for the required parameters. As shown in image 3, robot configuration is performed through a dialog interface in which a name, network address, and authentication credentials are defined. The robot's name serves as the identifier within

the application, while the network address and credentials enable communication with the robot controller.

Beyond basic configuration, the interface provides controls for defining scheduling constraints, such as minimum battery thresholds, working-day restrictions, and idle-automation settings. Consolidating the parameters within a single view enables operators to adjust system behaviour without altering the underlying implementation. Separating configuration from operational controls mitigates the risk of unintended changes during active use.

Scheduler behaviour can also be adjusted through parameters such as retry limits and polling intervals. These settings directly affect the scheduling system’s response to execution failures and the frequency of scheduling condition evaluations. Access to these parameters facilitates the system's adaptation to various operational environments.

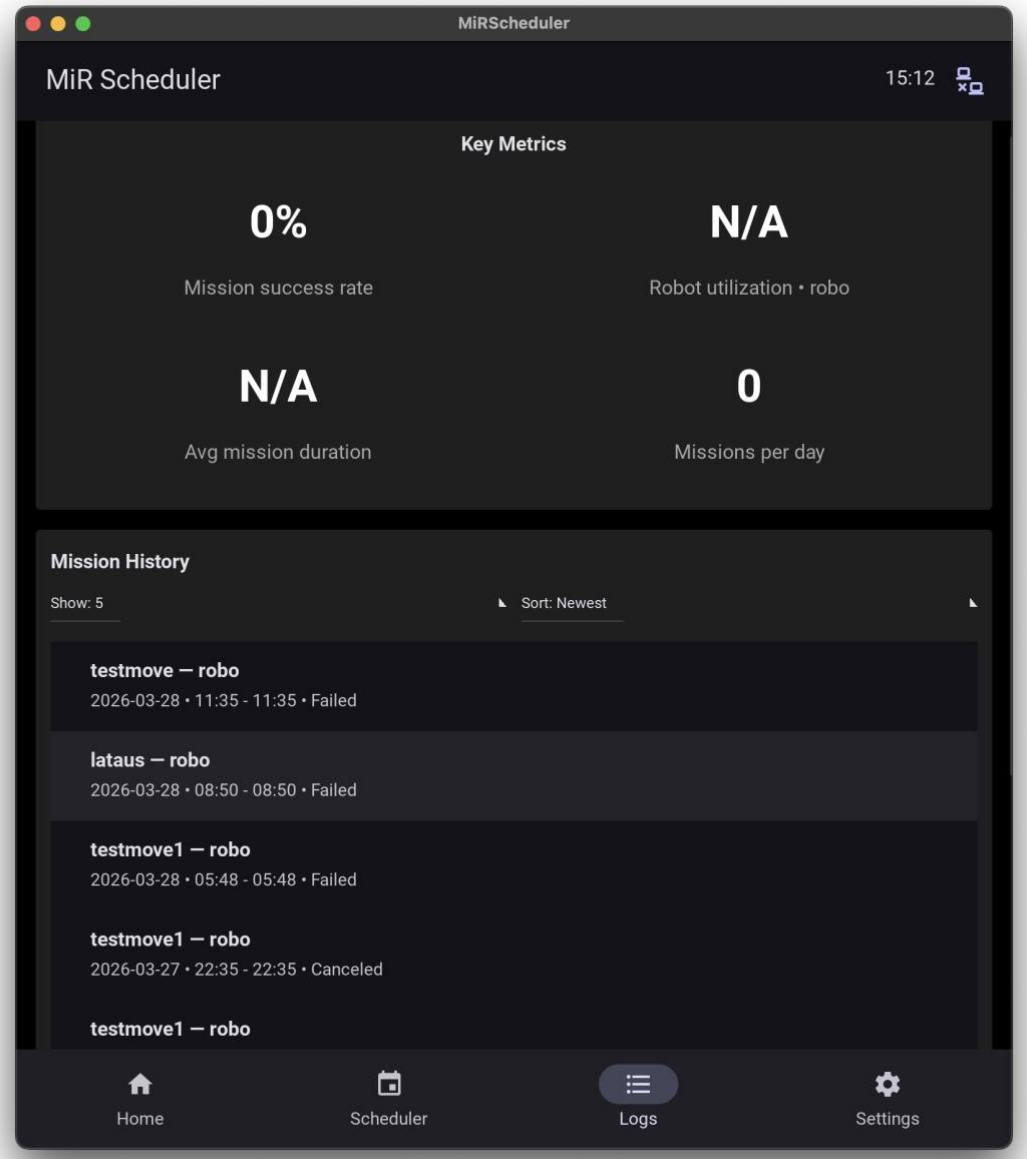

Image 4. **Mission execution log interface**

The Logs screen, shown in Image 4, provides access to mission execution history and performance metrics. The interface displays recorded mission events, including execution status, timestamps, and associated robot information. This data is retrieved from the local database and presented in a structured list format.

In addition to historical records, the interface presents aggregated performance indicators, including mission success rate, robot utilization, and average mission duration. These metrics offer a high-level overview of system performance and support evaluation of scheduling effectiveness. The integration of detailed execution logs and aggregated performance metrics enables both operational monitoring and post-analysis of system behaviour. This approach supports the broader objective of performance tracking within the developed scheduling system.

5.4 Mission storage and logging mechanism

The application employs a local database to store mission-related data necessary for scheduling and performance tracking. This encompasses both upcoming scheduled missions and historical records of completed missions. Persisting this data ensures reliable operation of the scheduler and supports analysis of system performance over time.

Scheduled missions are stored before execution and continuously evaluated by the scheduler to determine appropriate mission initiation times. Upon execution, a mission is removed from the scheduling queue and, for recurring missions, replaced by a new entry as necessary.

In addition to scheduling data, the application records the execution history for all missions processed by the system. Each execution event is stored with relevant details, including the associated robot, mission identifier, execution timestamps, and final status. Execution outcomes are classified as completed, failed, or cancelled. These records are generated during both standard operation and exceptional circumstances, such as when retry limits are exceeded or missions are manually cancelled.

The logging mechanism is integrated with mission-tracking functionality that monitors mission execution via the robot API. When a mission reaches a terminal state, the

corresponding execution record is stored in the database. This process ensures consistent capture of both successful and unsuccessful executions.

Collected execution data serves as the foundation for performance monitoring within the application. Performance indicators are derived from the stored mission history by aggregating execution outcomes and timestamps. The mission success rate is determined by comparing the number of completed missions to the sum of completed and failed missions, excluding cancelled missions. Average mission duration is calculated solely from completed missions, based on the difference between recorded start and end timestamps. Mission frequency is defined as the average number of completed missions per recorded day.

Robot-specific performance indicators are also derived from the execution history. Specifically, cumulative mission duration is calculated for each selected robot, enabling the interface to present a utilization metric for the currently active robot. These calculations are performed dynamically during data display in the user interface, eliminating the need to store KPI values separately and ensuring that the presented indicators always reflect the most recent mission history.

6 Testing and evaluation

The implemented scheduling system underwent functional testing and analysis of recorded mission execution data. The objective was to verify system behaviours under typical operating conditions and to assess the reliability of the application in mission scheduling, execution tracking, and management of exceptional situations.

The evaluation encompasses both normal operations and scenarios involving failures and manual interventions. The recorded mission data was used to validate that scheduling, execution monitoring, and logging mechanisms function consistently across varying conditions.

6.1 Test environment setup

The system was tested in an industrial workshop environment dedicated to the assembly and validation of automation systems. The testing area was mapped using the robot's native mapping functionality and featured a charging station as well as multiple predefined robot positions. These positions serve as mission targets during both scheduling and execution tests.

Testing utilized a single MiR 1350 autonomous mobile robot. The application interfaced with the robot over a local network, enabling real-time communication for mission scheduling and status monitoring.

The environment was not fully controlled because normal workshop activities continued during testing. These activities included personnel movement and occasional forklift traffic, which introduced variability in robot operation. Consequently, the testing conditions represent a realistic operational setting rather than a strictly controlled laboratory environment.

6.2 Test scenarios and executed missions

The scheduling system was tested using a series of functional scenarios intended to validate all core system behaviours. Instead of prioritizing large-scale data collection, the evaluation focused on verifying correct operation across various usage conditions and identifying potential edge cases during system execution.

Basic functionality was validated by executing individual missions manually and via scheduled execution. This process verified that missions were correctly submitted to the robot, executed as expected, and accurately recorded in the system's execution history.

Time-based scheduling behaviour was assessed using both one-time and recurring missions. Recurring scheduling scenarios involved repeated execution at defined intervals, confirming that new scheduling entries were generated correctly and that previously executed entries were removed or updated as intended.

Exceptional scenarios were evaluated to ensure robust system behaviours. These scenarios included simulated execution failures to verify retry logic and enforcement of retry limits, as well as manual cancellation of scheduled missions. In these cases, the system correctly updated mission status and recorded the corresponding events in the execution history.

Idle automation functionality was tested by allowing the robot to remain in a ready state without assigned tasks. The system detected idle conditions and triggered predefined missions after the configuration timeout period.

Testing was conducted iteratively during development, with identified issues corrected and revalidated. This approach ensured that all major functional components of the system were verified under both normal and exceptional operating conditions.

6.3 Performance metrics and data analysis

During testing, the application recorded mission execution data, which was subsequently used to generate performance metrics within the user interface. Although the evaluation did not include long-term data collection, the available mission history was sufficient to verify that the implemented performance tracking functionality operated as intended.

The system generated key performance indicators based on the recorded mission data, including mission success rate, average mission duration, mission frequency, and robot-specific utilization metrics. The calculated values accurately reflected the executed test scenarios and were updated dynamically as new mission records were added.

The mission success rate metric responded to both successful execution and simulated failure scenarios, confirming that execution outcomes were accurately recorded and classified. Similarly, average mission duration values corresponded to observed execution times, demonstrating correct processing of timestamp data.

Mission frequency metrics represented the number of completed missions relative to the number of recorded days, providing an overview of the system activity during testing. Robot-specific metrics, such as cumulative mission duration, enabled differentiation of workload between robots, although testing was primarily conducted with a single robot.

Overall, the generated metrics were consistent with the executed test scenarios and confirmed that the system can transform raw execution data into meaningful operational indicators. Although the dataset was limited in size, the results indicate that the performance-tracking mechanism functions correctly and provides a foundation for further analysis during extended deployment.

6.4 Discussion of results

Functional testing results demonstrate that the implemented scheduling system operates reliably under the tested conditions. Core functionalities, such as mission scheduling, execution triggering, and status tracking performed consistently during both standard operations and exceptional scenarios. The system correctly handled time-based scheduling, recurring missions, and manual interactions without unexpected behaviour.

The system's response to exceptional conditions, including failed mission execution and manual cancellation, functioned as intended. Retry mechanisms were activated appropriately, and mission outcomes were accurately recorded in the execution history. These results confirm the system's capability to maintain consistent operation despite deviation from standard execution.

The interface supported efficient user interaction by integrating monitoring and control functions within a limited set of views. The combination of real-time status information, scheduling controls, and execution history facilitated effective supervision of robot operations during testing.

The evaluation was conducted within a limited testing environment, utilizing a single robot and a short observation period. Although this approach verifies correct system functionality, extended deployment in larger-scale environments would be required to fully assess long-term performance and scalability.

7 Conclusions

The results of the implementation and evaluation establish a foundation for assessing the developed system within a practical context. Collectively, these findings demonstrate the system's capabilities and identify key considerations for its application and further development.

7.1 Summary of findings

This thesis aimed to develop and evaluate an external scheduling and performance-tracking system for autonomous mobile robots that lack native time-based scheduling functionality. The findings demonstrate that this functionality can be effectively implemented as an external application without requiring modifications to the robot's internal control system.

The developed system enables time-based mission scheduling through a dedicated software layer that communicates with the robot platform using REST API. Scheduled missions were executed reliably under test conditions, with support for recurring tasks, retry mechanisms, and handling of exceptional situations, such as failed or cancelled missions. These results confirm that external scheduling extends the operational flexibility of AMR systems in a practical and scalable manner.

Beyond scheduling, the system provides a structured method for collecting and storing mission-execution data. This facilitates the development of performance indicators that improve visibility into robot operations and system usage. The implementation demonstrates that performance tracking can be integrated with scheduling without necessitating changes to the robot platform.

Overall, the results indicate that external scheduling and performance tracking can be integrated into a unified application that enhances both operational control and system

transparency. This approach offers a practical solution for environments lacking native fleet management tools that support time-based mission automation or performance tracking.

7.2 Evaluation of research questions

The thesis initially examined how operational efficiency in AMR systems can be enhanced through a custom-built external scheduling interface. The results indicate that flexibility increases when high-level task planning is decoupled from the robot's internal logic. By enabling missions to be scheduled hours or days in advance, the system allows the AMR fleet to synchronize with broader production cycles, a capability not inherently supported by the native MiR software.

The analysis of technical requirements and challenges for integrating a REST-based scheduling system identified synchronization between the external Python environment and the robot's internal state as the primary challenge. Developing a robust communication layer required precise management of mission IDs and the implementation of error-handling logic to address network latency, thereby ensuring reliable REST API calls under variable industrial network conditions.

To address how performance tracking and data logging can be implemented, the application utilized an embedded SQLite database. This method established a persistent record of mission-specific timestamps, including creation, start, and completion times. Unlike the robot's internal logs, this structured data enabled calculation of actionable KPIs, such as mission success rates and average execution durations, which are essential for effective fleet management.

The thesis also examined the impact of the transition from reactive mission execution to structured time-based scheduling on fleet efficiency and predictability. The developed application demonstrates that addressing functional gaps in existing fleet management systems, particularly the absence of native time-based task queuing, significantly

enhances operational predictability. Moving beyond a reactive approach offers users a clearer overview of upcoming tasks and generates insights for continuous performance monitoring, thereby making the fleet a more transparent and reliable component of the logistics chain.

The thesis also identified architectural and user interface design principles that support practical deployment in industrial environments. A modular, middleware-based architecture was determined to be essential for separating the concerns of the user interface, data management layer, and robot communications. For the interface, the use of KivyMD demonstrated the importance of high-contrast, responsive design and clear visual feedback. Additionally, leveraging the Material Design library contributes to a more intuitive user experience, as most users are familiar with its visual language and interaction patterns from other modern applications. These principles ensure that critical information, such as mission status and fleet connectivity, remains accessible across various hardware platforms, which is essential for successful adoption in diverse industrial settings.

7.3 Limitations and challenges

The implementation and evaluation of the developed system were conducted within a defined scope, introducing specific limitations that must be considered when interpreting the results and assessing the solution’s applicability.

A primary limitation concerns the testing scope. Validation was performed using a single autonomous mobile robot over a short testing period. While this is sufficient to verify correct functionality and system behaviour, it does not allow for evaluation of long-term performance, reliability during continuous operation, or behaviour in larger multi-robot deployments. Further testing in more extensive operational environments is necessary to assess scalability and robustness under increased system load.

The system design is constrained by reliance on the robot platform's REST API. Consequently, the implementation does not modify or extend the robot's internal control logic, and all functionality is achieved through external interaction. Although this approach ensures compatibility and facilitates integration, it limits the degree of control over mission execution and system behaviour.

Functionally, the scheduling mechanism relies on predefined execution times and does not incorporate advanced decision-making or optimization logic. Features such as dynamic task allocation, priority-based scheduling, and automatic distribution of missions among multiple robots are not included. In multi-robot scenarios, mission assignment must be defined manually during scheduling, which restricts the level of automation.

The system operates as a local application without centralized or cloud-based coordination. Although this simplifies deployment and reduces system complexity, it restricts remote access, scalability, and integration with broader enterprise-level systems.

Finally, the performance metrics generated by the system are based on a limited dataset collected during testing. While this dataset is sufficient to verify the correct operation of the performance-tracking mechanism, it does not support a comprehensive analysis of long-term operational efficiency or comparative evaluation across different deployment scenarios.

7.4 Recommendations for further development

Future development of the system should address the identified limitations concerning data availability and scheduling functionality. Extending data collection over longer operational periods would facilitate a more comprehensive analysis of robot performance and system behaviour, thereby supporting a more reliable evaluation of system effectiveness.

Additionally, the collected mission data could be utilized to enhance scheduling capabilities. Implementing data-driven methods, such as adaptive scheduling, would enable the system to respond more effectively to dynamic operational conditions. In multi-robot environments, these methods could be extended to support automatic mission allocation, thereby improving task distribution and reducing the need for manual assignment during scheduling.

The scheduling mechanism could also be improved by incorporating priority-based execution logic and basic conflict handling between scheduled tasks. These enhancements would ensure that time-critical missions are prioritized and would minimize the risk of overlapping or impractical scheduling scenarios.

7.5 Final remarks

The work presented in this thesis demonstrates that external systems can be used to extend the functionality of autonomous mobile robots in a practical and accessible manner. By relying solely on the available API, it is possible to add scheduling and monitoring capabilities without interfering with the robot's internal operation.

From an engineering perspective, the developed solution offers a direct means of enhancing operational flexibility while preserving system simplicity. The implementation illustrates how lightweight tools can address practical limitations encountered in real-world deployments.

The results of this work support the broader perspective that external applications can significantly enhance industrial robotic systems. Even without extensive integration into the underlying control architecture, these solutions offer measurable improvements in usability, flexibility, and system transparency.

References

Baluyot, P. J., Ramos, K. R., Tan, C. A., Valenzuela, R. I., Saflor-Balmes, C., & Bernardo, E. (2026). Effect of design styles of user interface on user experience. Engineering Proceedings, 128(1), 23. https://doi.org/10.3390/engproc2026128023

Chakraa, H., Guérin, F., Leclercq, E., & Lefebvre, D. (2023). Optimization techniques for multi-robot task allocation problems: Review on the state-of-the-art. Robotics and Autonomous Systems, 168, 104492. https://doi.org/10.1016/j.robot.2023.104492

Dobrzanska, M., & Dobrzanski, P. (2025). Simulation model as an element of sustainable autonomous mobile robot fleet management. Energies, 18(8), 1894. https://doi.org/10.3390/en18081894

Eisen, M., Sudhakaran, S., Mageshkumar, V., Baxi, A., & Cavalcanti, D. (2023). Joint resource scheduling for AMR navigation over wireless edge networks. IEEE Open Journal of Vehicular Technology, 4, 1–11. https://doi.org/10.1109/OJVT.2022.3218460

Fragapane, G., de Koster, R., Sgarbossa, F., & Strandhagen, J. O. (2021). Planning and control of autonomous mobile robots for intralogistics: Literature review and research agenda. European Journal of Operational Research, 294(2), 405–426. https://doi.org/10.1016/j.ejor.2021.01.019

Grover, A. K., & Ashraf, M. H. (2024). Leveraging autonomous mobile robots for Industry 4.0 warehouses: A multiple case study analysis. International Journal of Logistics Management, 35(4), 1168–1199. https://doi.org/10.1108/IJLM-09-2022-0362

Jeong, S., Ga, T., Jeong, I., Oh, J., & Choi, J. (2022). Layered-cost-map-based traffic management for multiple AMRs via a DDS. Applied Sciences, 12(16), 8084. https://doi.org/10.3390/app12168084

Kagan, E., Ben-Gal, I., & Shvalb, N. (2019). Autonomous mobile robots and multi-robot systems. John Wiley & Sons.

Kozjek, D., Malus, A., & Vrabič, R. (2021). Reinforcement-learning-based route generation for heavy-traffic autonomous mobile robot systems. Sensors, 21(14), 4809. https://doi.org/10.3390/s21144809

Löffler, M., Boysen, N., & Schneider, M. (2023). Human–robot cooperation: Coordinating autonomous mobile robots and human order pickers. Transportation Science, 57(4), 979–998. https://doi.org/10.1287/trsc.2023.1207

Lopes, D., Pereira, T., Goncalves, A., Cunha, F., Lopes, F., Antunes, J., Santos, V., Coutinho, F., Barreiros, J., Duraes, J., Santos, P., Simoes, F., Ferreira, P., Freitas, E. D. C., Trovao, J. P. F., Ferreira, J. P., & Fonseca Ferreira, N. M. (2025). Integrated fleet management of mobile robots for enhancing industrial efficiency: A case study on interoperability in multi-brand environments within the automotive sector. Applied Sciences, 15(13), 7235. https://doi.org/10.3390/app15137235

Morais, P. H. C., Vivaldini, K. C. T., Kato, E. R. R., & Inoue, R. S. (2025). A review of robot fleet management. IEEE Access, 13, 118975–119003. https://doi.org/10.1109/ACCESS.2025.3586564

Next Move Strategy Consulting. (2025). *Global Autonomous Mobile Robot (AMR) Market Overview, Trends, and Forecasts 2024–2030*. Next Move Strategy Consulting. https://doi.org/10.5281/ZENODO.18043048

Oyekanlu, E. A., Smith, A. C., Thomas, W. P., Mulroy, G., Hitesh, D., Ramsey, M., Kuhn, D. J., Mcghinnis, J. D., Buonavita, S. C., Looper, N. A., Ng, M., Ng'oma, A., Liu, W., Mcbride, P. G., Shultz, M. G., Cerasi, C., & Sun, D. (2020). A review of recent advances in automated guided vehicle technologies: Integration challenges and research areas for 5G-based

smart manufacturing applications. IEEE Access, 8, 202312–202353. https://doi.org/10.1109/ACCESS.2020.3035729

Pang, H., & Zhen, L. (2024). Automated mobile robots routing and job assignment in automated factory. Computers & Industrial Engineering, 195, 110420. https://doi.org/10.1016/j.cie.2024.110420

Papoutsakis, M., Hatzivasilis, G., Michalodimitrakis, E., Ioannidis, S., Michael, M., Savva, A., Nikolaou, P., Stokkou, E., & Bozdemir, G. (2025). SESAME: Automated security assessment of robots and modern multi-robot systems. Electronics, 14(5), 923. https://doi.org/10.3390/electronics14050923

Shakeri, Z., Benfriha, K., Varmazyar, M., Talhi, E., & Quenehen, A. (2025). Production scheduling with multi-robot task allocation in a real Industry 4.0 setting. *Scientific Reports, 15*(1), 1795. https://doi.org/10.1038/s41598-024-84240-3

Soares, F. A. L., & Alves, L. P. B. (2025). Indoor autonomous mobile robot for environment mapping: A systematic mapping of the literature. Abakós, 13(2), e2025130202. https://doi.org/10.5752/P.2316-9451.e2025130202

Valero, O., Antich, J., Tauler-Rosselló, A., Guerrero, J., Miñana, J.-J., & Ortiz, A. (2023). Multi-robot task allocation methods: A fuzzy optimization approach. Information Sciences, 648, 119508. https://doi.org/10.1016/j.ins.2023.119508

Vermesan, O., Bahr, R., Ottella, M., Serrano, M., Karlsen, T., Wahlstrom, T., Sand, H., Ashwathnarayan, M., & Gamba, M. (2020). Internet of robotic things: Intelligent connectivity and platforms. Frontiers in Robotics and AI, 7, 104. https://doi.org/10.3389/frobt.2020.00104

Appendices

Appendix 1. Disclosure of AI-assisted tools

The author used DeepL and Grammarly during the writing process for language editing, translation support, and grammatical correction. These tools were not used to generate research results, analysis, or conclusions. The author remains fully responsible for the final content of the work.