Review

ROS 2 in a Nutshell: A Survey

Abdulrahman S. Al-Batati * , Anis Koubaa , Mohamed Abdelkader , Khaled Gabr , Hamad Aloqaily

Posted Date: 8 May 2025

doi: 10.20944/preprints202410.1204.v3

Keywords: ROS; ROS 2; UAV; modularity; real-time; security; multi-robot systems; robotics middleware; embedded systems; distributed robotics; robot software ecosystems; ROS QoS; ROS frameworks; ROS Packages; ROS simulators; ROS database; open-source robotics; DDS; simulation; autonomous systems

 doi:10.20944/preprints202410.1204.v3

Review

ROS 2 in a Nutshell: A Survey

Abdulrahman S. Al-Batati [1,*], Anis Koubaa [2], Mohamed Abdelkader [1], Khaled Gabr [1] and Hamad Aloqaily [1]

[1] Robotics & Internet of Things Lab, Prince Sultan University, Riyadh, 12435, Saudi Arabia
[2] College of Computer and Information Sciences, Alfaisal University, Riyadh, Saudi Arabia
* Correspondence: aalbatati@psu.edu.sa

Abstract: This study presents a comprehensive review that addresses the critical transition from ROS 1 to ROS 2, spotlighting the significant enhancements and the pressing need for a detailed exploration of ROS 2 within the robotics community. Despite the extensive deployment and adaptations of ROS in varied robotics applications, literature lacks a cohesive synthesis that delineates the advancements, limitations, and broader impacts of ROS 2 compared to its predecessor, ROS 1. Our contribution bridges this gap by assembling the largest database of ROS-related research, encompassing 7,371 articles, with a focused analysis in this survey on 435 ROS2-specific publications. We categorize these into *i.)* articles that discuss and analyze core ROS 2 concepts, *ii.)* articles that propose frameworks or tools for ROS 2, and *iii.)* articles utilizing ROS 2. Furthermore, we summarize literature findings of ROS 2 challenges, advancements, and future direction in the fields of *a.)* security, *b.)* real-time, *c.)* middleware, *d.)* embedded and distributed systems, *e.)* communication reliability and QoS, and *f.)* multi-robot systems. The methodology involved meticulous data collection and categorization from multiple databases, facilitating an in-depth online accessible resource. Results underscore ROS2's enhancements in modularity, real-time capabilities, and security, extending its applicability across various robotic platforms and industries. However, challenges in scalability and reliability persist, signaling avenues for future enhancements. This review not only deepens the understanding of ROS2's contributions but also charts a path for ongoing improvements in robotic systems design. The original data presented in the study are openly available in https://www.ros.riotu-lab.org/.

Keywords: ROS; ROS 2; ROS modularity; real-time systems; security; multi-robot systems; robotics middleware; embedded systems; distributed robotics; robot software ecosystems; ROS QoS; ROS frameworks; ROS Packages; ROS simulators; ROS database; open-source robotics; DDS; simulation; autonomous systems

1. Introduction

The Robot Operating System (ROS, aka ROS 1) emerged in 2009 as the de facto standard ecosystem for developing robotics applications, including mobile robots, robotic arms, unmanned aerial systems, self-driving cars, quadruped robots, and space robots [1–5]. ROS provides access to several open-source libraries, facilitating faster development of robotic systems and applications. Examples include gmapping from OpenSLAM [4], Google Cartographer for SLAM [6], OpenCV for image processing [7], and the Point Cloud Library for 3D perception [8]. Additionally, ROS abstracts low-level hardware interactions, allowing developers to focus on high-level robotics applications rather than dealing with complex hardware-specific implementations.

Despite its success, ROS 1 exhibits several limitations, including: (a) a single point of failure with the ROS Master Node, limiting scalability; (b) lack of built-in multi-robot support, as ROS was initially designed for single-robot applications; and (c) absence of real-time guarantees or Quality of Service (QoS) support, raising concerns about its reliability in safety-critical applications [1]. In response, the Open Source Robotics Foundation (OSRF) initiated the development of ROS 2 to address these issues.

 doi:10.20944/preprints202410.1204.v3

The first official ROS 2 release (Ardent Apalone) was launched in December 2017, with incremental improvements leading to feature parity with ROS 1 by 2023 as indicated in Figure 1. However, despite these advancements, ROS 2 adoption has been slower than expected due to factors such as migration complexity, incomplete documentation, and industry hesitation in transitioning from ROS 1. While ROS 2 is expected to fully replace ROS 1 after the end-of-life (EOL) of ROS Noetic (Figure 1 shows the road map to ROS 2 until the Jazzy distribution release.), the transition raises critical questions about its adoption, technical maturity, and real-world impact across different industries.

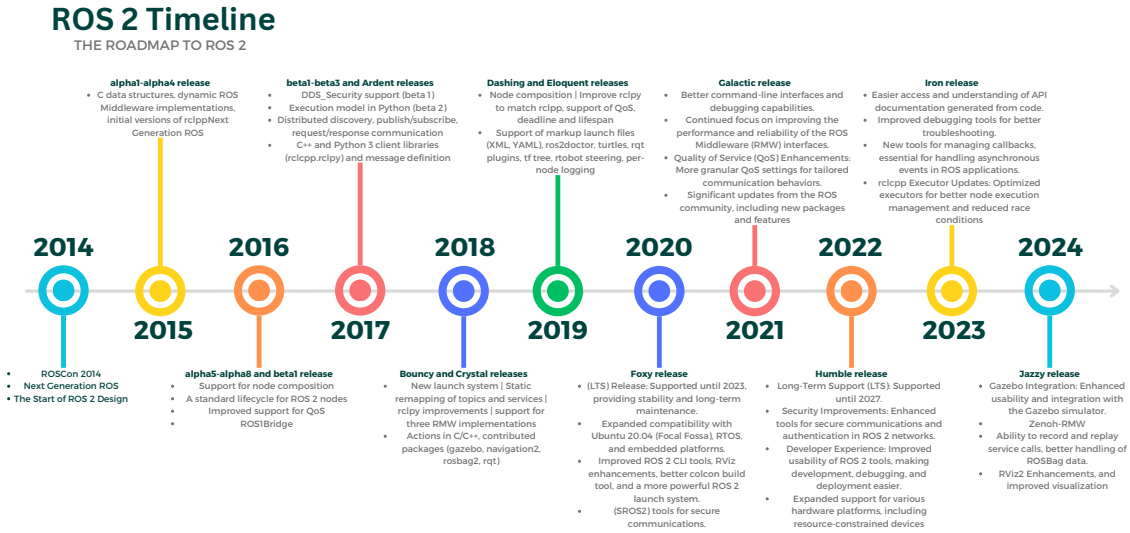

Figure 1. Timeline of ROS 2 until 2024.

1.1. Research Questions and Objectives

While multiple reviews have explored specific subsystems of ROS 2—such as middleware, security, and real-time scheduling—there is currently no comprehensive survey that critically examines both the technical advancements and the practical adoption barriers of ROS 2. Additionally, research on the real-world use cases of ROS 2 across industries remains fragmented. To bridge this gap, this survey systematically investigates ROS 2's technical foundations, adoption trends, and existing challenges by addressing the following key research questions:

1. *How does ROS 2 improve upon ROS 1 in terms of architecture, middleware efficiency, real-time performance, and security, and what are its current limitations?*: This question critically examines DDS-based communication, real-time execution, security features, and middleware efficiency, identifying both strengths and bottlenecks, and is addressed in the part of this survey that presents a background coverage on ROS1 and ROS2.
2. *What are the key trends, challenges, and solutions in adopting ROS 2 across various academic research and industries, including autonomous vehicles, healthcare, agriculture, logistics, and industrial automation?*: This question explores how ROS 2 is deployed in real-world applications, analyzing success stories, industry adoption barriers, and interoperability concerns.
3. *What are the key technical, industrial, and organizational challenges hindering a faster widespread adoption of ROS 2, and how can they be addressed?*: This question investigates barriers such as middleware complexity, migration difficulties from ROS 1, lack of standardized industry support, and real-time performance concerns, while identifying potential solutions to accelerate adoption.

1.2. Methodology

To systematically examine the advancements and challenges of ROS 2, this survey follows a structured methodology grounded in systematic review principles. The research is guided by three key questions: (1) How does ROS 2 improve upon ROS 1 in architecture, middleware efficiency, real-time performance, and security, and what are its limitations? (2) What are the key trends, challenges, and

 doi:10.20944/preprints202410.1204.v3

solutions in adopting ROS 2 across industries such as autonomous vehicles, healthcare, and logistics? (3) What technical, industrial, and organizational factors hinder its widespread adoption, and how can they be addressed?

To systematically investigate these questions, we designed a structured review methodology consisting of data collection, filtering, classification, and analysis steps (see Figure 2.

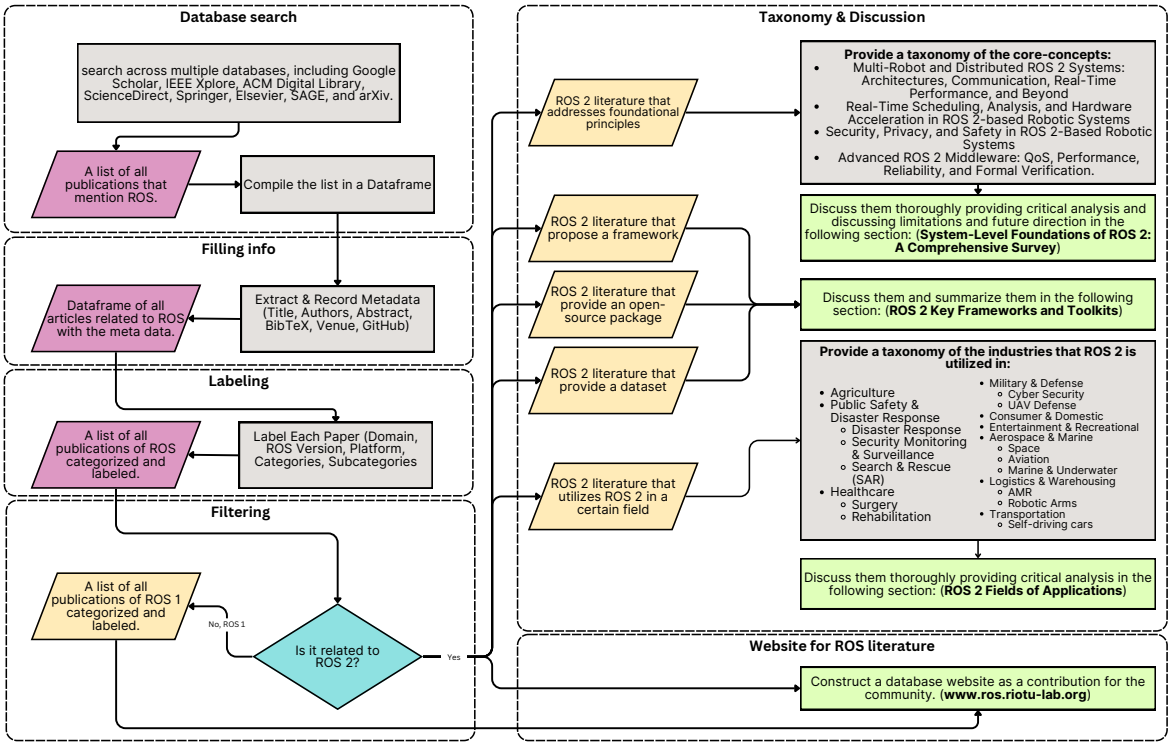

Figure 2. Overall methodology for collecting, labeling, and categorizing **~8000** ROS-related publications, culminating in deeper analyses for ROS 2-specific literature. We show how we (1) aggregated papers from multiple databases, (2) extracted and validated metadata, (3) further labeled the data, (4) identified **435** articles focusing on ROS 2, and (5) categorized them by article type and research domain. The figure also indicates how each subset is discussed in Section 4, Section 6, and Section 5.

1.2.1. Data Collection and Sources

An extensive literature search was conducted across multiple academic databases, including *Google Scholar*, *IEEE Xplore*, *ACM Digital Library*, *ScienceDirect*, *Springer*, *Elsevier*, *SAGE*, and *arXiv*. The search terms included "ROS 2," "Robot Operating System 2," "ROS middleware," "ROS security," "ROS real-time," and other relevant keywords. No restrictions were placed on publication year to ensure coverage of both early-stage and recent research. Additionally, reference lists of selected papers were examined to capture relevant studies that may not have appeared in initial search results.

1.2.2. Inclusion and Exclusion Criteria

Figure 3 illustrates the rigorous filtering process that was applied to ensure the relevance and quality of the selected publications. The inclusion criteria were carefully designed to maintain high academic standards. First, studies that provided detailed discussions of ROS 2's architecture, middleware, security, real-time performance, or adoption were included as they offer direct insights into the system's capabilities. Comparative analyses between ROS 1 and ROS 2 were also incorporated to understand the evolutionary improvements and potential migration challenges. The review focused on peer-reviewed journal articles, conference papers, and preprints from reputable sources to ensure scientific rigor. Additionally, research proposing new frameworks, tools, or open-source packages based on ROS 2 was included to capture innovative developments in the field. Empirical research

 doi:10.20944/preprints202410.1204.v3

evaluating ROS 2 performance in real-world applications was particularly valuable for understanding practical implementation challenges.

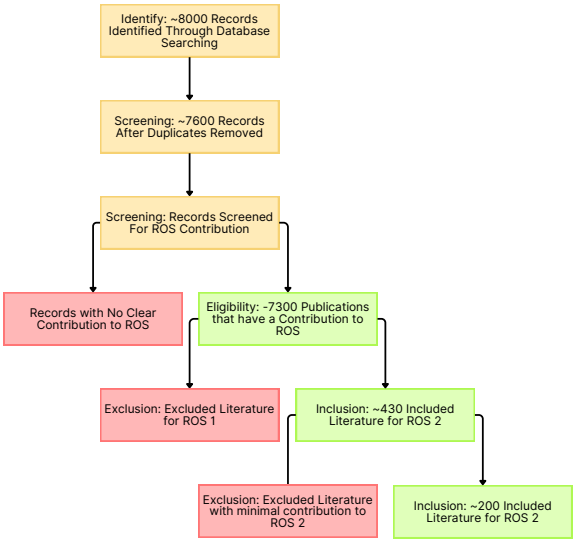

Figure 3. PRISMA flow chart illustrating inclusion and exclusion criteria.

The exclusion criteria were equally stringent. Articles that merely mentioned ROS 2 without substantive discussion were excluded as they provided limited analytical value. Studies focusing exclusively on ROS 1 without addressing the transition to ROS 2 were omitted to maintain relevance to the current technological landscape. Non-peer-reviewed materials, such as blog posts, opinion pieces, and editorial notes, were excluded to ensure academic quality. Furthermore, duplicate records and papers lacking sufficient methodological details for reproducibility were removed to maintain the integrity of the analysis.

After applying these criteria, 7,371 ROS-related publications were retained, of which 435 were specifically focused on ROS 2.

1.2.3. Data Processing and Classification

Each of the 435 ROS 2 publications was systematically categorized to facilitate structured analysis. The classification framework encompassed multiple dimensions to ensure comprehensive coverage. The research domain classification examined various technical aspects, including multi-robot systems, security implementations, real-time scheduling mechanisms, middleware solutions, communication reliability protocols, and distributed systems architectures.

The robotic platform categorization covered a wide spectrum of applications, from aerial vehicles (UAVs) and ground vehicles (UGVs) to humanoid robots, industrial robots, and marine robotics systems. This classification helped identify platform-specific challenges and solutions. The application area analysis focused on key sectors such as healthcare, agriculture, autonomous navigation, logistics, and industrial automation, providing insights into domain-specific requirements and adaptations.

The contribution type classification that is shown in Figure 4 provided another important analytical dimension. Papers discussing fundamental aspects of ROS 2, such as architecture, security, and middleware improvements, were categorized as core component analyses. Research focused on developing new libraries, simulators, or infrastructure enhancements was classified under framework development. Studies contributing publicly available software tools or frameworks were categorized as open-source packages. Application-focused research utilizing ROS 2 for specific problem domains, such as autonomous vehicles or industrial robotics, formed a distinct category. Publications presenting sensor datasets or benchmarking data collected using ROS 2 were classified as dataset contributions.

To ensure accuracy and consistency, categorization was performed independently by two reviewers, with discrepancies resolved through discussion and consensus.

 doi:10.20944/preprints202410.1204.v3

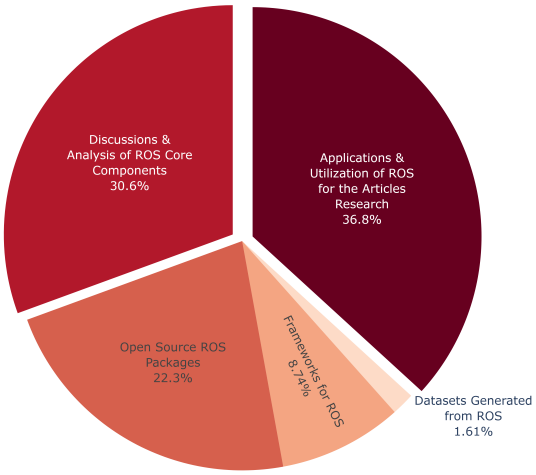

Figure 4. Taxonomy of ROS 2 articles types.

1.2.4. Analysis and Synthesis

Following the classification, the literature was systematically mapped to specific sections of the survey. The technical advancements section (Section 3) provides a comprehensive analysis of improvements in middleware implementations, real-time processing capabilities, and security features. This is followed by an examination of industry-specific applications (Section 4), which explores ROS 2 adoption across different fields while identifying both success stories and implementation challenges. The challenges and adoption barriers section (Section 5) presents a detailed analysis of critical factors such as scalability limitations, migration difficulties, and real-time performance constraints.

Additionally, a thematic analysis was conducted to identify emerging trends in ROS 2 research, highlighting areas requiring further exploration. This analysis revealed patterns in development approaches, common challenges, and potential future directions for the ROS 2 ecosystem.

1.2.5. Open-Source Database for Reproducibility

To enhance transparency and accessibility, all processed data have been compiled into a publicly available ROS 2 literature database at /https://www.ros.riotu-lab.org. This database allows filtering by research domain, ROS version, and robotic platform, ensuring continued research contributions. Metadata such as citation counts, publication venues, and GitHub links are included to facilitate further analysis. The database serves as a valuable resource for researchers and practitioners working with ROS 2, providing a comprehensive overview of current research and development efforts in the field.

1.3. Related Surveys

The body of literature on ROS 2 has grown significantly, with various surveys addressing specific aspects of this robotic operating system. While these studies contribute valuable insights, their focus has often been narrow and fragmented, leaving critical gaps in the overall understanding of ROS 2's ecosystem. This subsection critically examines prior surveys, their methodologies, and the limitations that our work seeks to address. A summarized comparison is presented in Table 1.

Several noteworthy surveys have explored ROS 2 from different perspectives. Audonnet et al. [9] conducted a systematic comparison of simulation frameworks for robotic arm manipulation but did not address broader middleware or real-time concerns. Zhang et al. [10] reviewed distributed robotic systems within the edge-cloud continuum, emphasizing ROS 2's role in decentralized architectures. However, this study was technology-focused and did not evaluate adoption barriers, security risks, or performance trade-offs. Similarly, Macenski et al. [11] provided a detailed review of mobile robotics algorithms within ROS 2, focusing on navigation strategies while neglecting middleware advancements and security challenges.

 doi:10.20944/preprints202410.1204.v3

Other studies have examined specific technical challenges within ROS 2. Choi et al. [12] explored priority-driven real-time scheduling, addressing deterministic execution but without considering its integration into multi-robot systems. Macenski et al. [13] provided an overview of ROS 2's design, architecture, and applications, though the discussion lacked empirical validation of performance metrics. DiLuoffo et al. [14] emphasized security concerns, advocating for improved cryptographic measures but failing to assess real-world security implementations. Alhanahnah [15] investigated software quality in ROS through static analysis but did not extend the discussion to ROS 2's modular improvements.

Despite these contributions, existing surveys exhibit several methodological gaps. Many are domain-specific, lacking a holistic analysis that spans multiple research areas. Some fail to apply systematic literature review principles, relying on selective paper collections without clear inclusion-exclusion criteria. Moreover, prior studies rarely integrate empirical benchmarks, making it difficult to assess the actual performance impact of proposed solutions. Additionally, while ROS 2's evolution over time is crucial for understanding its maturity, few surveys offer longitudinal analyses of adoption trends.

This survey aims to address these gaps by providing a comprehensive, systematic review of ROS 2 research across multiple dimensions: middleware advancements, real-time capabilities, security, modularity, and industry adoption. Unlike prior studies, we introduce a curated, publicly accessible database of ROS and ROS 2 literature, categorizing publications by research domain, robotic platform, industry focus, and contribution type. By synthesizing insights across different fields, we present a broader and more interconnected understanding of ROS 2, highlighting key challenges and emerging research directions that have not been adequately covered in previous surveys. Furthermore, by systematically analyzing these dimensions, this survey addresses our **research questions** regarding how ROS 2 improves upon ROS 1, what factors influence its adoption across industries, and what barriers hinder its widespread deployment. Through this structured investigation, we aim to provide a clearer roadmap for researchers and practitioners seeking to enhance ROS 2's capabilities and accelerate its integration into real-world robotic systems.

Table 1. Comparison of ROS 2 Surveys.

Survey	Focus Area	Gaps	Our Contribution
Audonnet et al. [9]	Simulation software for robotic arms	Limited to simulation software	Comprehensive coverage of ROS 2 applications
Zhang et al. [10]	Edge-cloud integration	Does not cover security, real-time performance	Wide range of research topics including security and modularity
Macenski et al. [11]	Mobile robotics navigation	Focuses only on navigation systems	Holistic view including multi-robot systems and real-time performance
Choi et al. [12]	Real-time scheduling	Limited to real-time performance	Inclusion of various research domains
Bonci et al. [16]	Industrial autonomy	Focused on industrial applications	Broader application fields
Macenski et al. [13]	ROS 2 architecture and uses	Lack of systematic research analysis	Systematic review and database of literature
DiLuoffo et al. [14]	Security in ROS 2	Focused solely on security	Inclusion of multiple research areas
Alhanahnah [15]	Software quality assessment	Centered on ROS 1	Extensive insights into ROS 2

1.4. Contributions

This survey systematically examines ROS 2, addressing three key research questions: (1) How does ROS 2 improve upon ROS 1 in terms of architecture, middleware efficiency, real-time performance,

 doi:10.20944/preprints202410.1204.v3

and security? (2) What are the key trends, challenges, and solutions in adopting ROS 2 across various industries? (3) What barriers hinder its widespread deployment, and how can they be addressed? Our main contributions are as follows:

1. **A Systematic Review of ROS 2 Research** – To answer RQ1, we conduct a structured analysis of middleware advancements, real-time scheduling, security, modularity, and multi-robot systems. By comparing ROS 2's improvements over ROS 1, we highlight enhancements in communication (DDS), real-time constraints, and security mechanisms, while identifying unresolved technical challenges.
2. **Empirical Assessment of Adoption Trends and Challenges** – Addressing RQ2, we evaluate the adoption of ROS 2 in industries such as autonomous vehicles, healthcare, industrial automation, and logistics. We identify key drivers (e.g., modularity, improved middleware) and major barriers (e.g., migration complexity, deterministic execution issues, and security risks) that impact real-world scalability.
3. **Mapping the ROS 2 Ecosystem** – Supporting RQ2 and RQ3, we analyze open-source frameworks, libraries, and tools, assessing their adoption, technical maturity, and gaps in the ecosystem. By investigating influential GitHub repositories and industrial contributions, we highlight areas requiring further development.
4. **A Publicly Accessible ROS 2 Literature Database** – To address RQ3, we introduce a curated, open-access database categorizing ROS 2 research by domain, robotic platform, industry, and contribution type, enhancing reproducibility and guiding future advancements.

By bridging theoretical insights with real-world applications, this survey provides a structured roadmap to accelerate ROS 2 adoption and development.

1.5. Paper Structure

This paper is organized as follows: Section 2 provides an overview of ROS and ROS 2, outlining their development history, core functionalities, and key differences. Section 3 presents a comparative analysis of ROS 1 and ROS 2, highlighting architectural improvements in middleware, real-time performance, and security.

Section 4 explores critical research areas within ROS 2, including security, real-time capabilities, multi-robot systems, and communication frameworks, offering insights into ongoing advancements and challenges. Section 5 reviews the diverse applications of ROS 2 across industries such as autonomous systems, healthcare, and logistics.

Section 6 discusses ROS 2 frameworks, toolkits, and notable open-source contributions, examining their role in system development and adoption. Section 7 introduces the curated ROS 2 literature database, analyzing key trends and research directions.

Finally, Section 8 summarizes the survey's findings, and highlights the survey's answers to the research questions.

2. ROS & ROS 2 Overview

This section directly addresses Research Question 1 by examining how ROS 2 improves upon ROS 1 in terms of architecture, middleware efficiency, real-time performance, security, and scalability. We begin by outlining the core features of ROS, followed by an in-depth discussion of the design objectives and motivations behind ROS 2's development. Finally, we provide a comparative analysis between the two versions, highlighting key advancements and limitations to offer a comprehensive understanding of ROS 2's evolution and its impact on modern robotics applications.

2.1. ROS

2.1.1. ROS Design Goals

ROS was fundamentally designed to serve as a unified platform for building single-robot applications. It offers a standard collection of tools, libraries, and conventions that aid in developing a

 doi:10.20944/preprints202410.1204.v3

broad spectrum of robotic applications, spanning from basic prototypes to intricate robotic systems. One of the salient design objectives of ROS was to promote reusability and modularity, enabling developers to readily share and consolidate software components, thereby facilitating the creation of sophisticated robotic applications. Notwithstanding the diverse use cases it covers, ROS's primary focus remains on single-robot systems. While there have been initiatives to extend ROS's capabilities towards multi-robot systems [17], these efforts were neither seamlessly integrated nor widely adopted.

The next section will provide an overview of ROS Architecture.

2.1.2. ROS Architecture

ROS architecture is depicted in Figure 5 and summarized as follows.

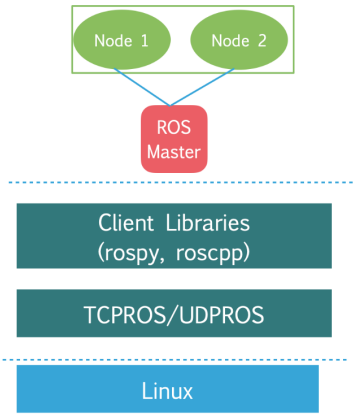

Figure 5. ROS 1 Architecture.

Natively Centralized

ROS primarily serves as a middleware framework, promoting communication between various processes, called *nodes*, within a computational machine. Conceptually, ROS can be envisaged as a *network of nodes (processes)*, exchanging messages to execute a specific task. This network pivots around a *central node*, termed the *ROS master node* (ROSCORE), which serves as a liaison between all nodes within the ROS network. The functionality of the ROS network is critically dependent on this master node. It is worth noting that the dependence on a single master node presents limitations in terms of scalability and introduces a single point of failure.

A ROS node can be a simple C++ program or a Python script, which reads data from other nodes, processes this data, and subsequently publishes it for other nodes. For instance, a camera node may acquire images via its drivers and publish this image data through a specified channel or topic (e.g., `/camera/rgb/`). Another node may receive this image data, process it using computer vision algorithms, and send the results to another node for appropriate action.

A Modular and Scalable Architecture

The structure of ROS, characterized by its distributed nodes, closely reflects the concept of a microservices architecture. In such an architecture, a monolithic application is partitioned into smaller, standalone services, each performing a specific function. This design is a strategic approach to managing complexity in large applications, facilitating modularity, and promoting scalability, as services can be developed, deployed, and scaled independently. Similarly, ROS strongly emphasizes modularity, allowing its applications to be executed and operated over multiple nodes. Each node in a ROS application performs a discrete task and can operate independently, analogous to a service in a microservices architecture. This arrangement allows for simplified debugging, testing, and understanding of individual components while permitting complex behaviors when the components interact. Thus, ROS's modular and distributed nature echoes the microservices architecture principles, enhancing maintainability and scalability in robotic applications.

 doi:10.20944/preprints202410.1204.v3

Communication Patterns

ROS employs several communication mechanisms that enable a seamless exchange of information between nodes.

- ***Publisher-Subscriber (Topics):*** Fundamental to ROS's communication infrastructure is the Publisher-Subscriber model, implemented through *Topics*. In this model, a node that generates data serves as the *Publisher*, broadcasting information over a channel known as a Topic. Other nodes that need this information can *subscribe* to the respective Topic, thereby consuming the published data. This decoupled communication model enables a many-to-many data flow, promoting the distribution of data to all interested nodes. Topics carry ROS messages, which encapsulate the information to be transmitted. The Publisher-Subscriber paradigm enhances flexibility, as nodes can dynamically publish or subscribe to Topics, and robustness, as the failure of one node does not directly impact others.
- ***Synchronous Client/Server: (Services)*** Services denote the synchronous client/server model within ROS. In this paradigm, a client node sends a request to a server node and waits for a response. The client is unable to perform other tasks until this response is received, indicating a *synchronous* operation. This model is applied when the server can complete the task swiftly and the client requires an immediate response. If the task duration is prolonged, the Services model becomes unsuitable, and the ROSLib concept is employed instead.
- ***Asynchronous Client/Server: (ActionLib)*** The ActionLib model signifies the *asynchronous client/server model* in ROS. This model contrasts with ROS Services in that the ActionLib client is not blocked while waiting for the server's response. The client can concurrently execute other tasks while the server processes its mission. In addition, the ActionLib server can send intermediate results to the client, updating it on the task progress. This model is particularly useful for tasks such as robot navigation, where the server processes a navigation request over an extended period, and the client monitors progress while performing other tasks.
- ***Messages and Services:*** In ROS, two distinct types of entities are used to facilitate communication: *Messages* for Topics and *Services* for synchronous client-server communication. Messages (`rosmsg`) are simple data structures comprising typed fields. They are used when nodes want to transmit data to one or multiple recipients. On the other hand, Services (`rossrv`) are defined by a pair of messages, one for the request and one for the response. They are used for synchronous RPC-like communication. This division, while serving its purpose, introduced complexity to the development process, as developers had to work with different types of entities depending on the communication model employed.

Client Libraries and Communication Middleware

While the fundamental concepts of ROS are retained in ROS 2, substantial differences lie in the client libraries, the communication middleware, and their underlying implementations.

In ROS, two primary client libraries are used: rospy for Python and roscpp for C++. These libraries provide the means for creating ROS nodes and handling the communication between them. Rospy primarily supports Python 2 across all ROS versions, except for the final ROS version, ROS Noetic, which introduces Python 3 support.

Regarding communication middleware, ROS employs a custom-made middleware: ROSTCP and ROSUDP, based on the TCP and UDP protocols, respectively. ROSTCP ensures reliable, connection-oriented communication, while ROSUDP enables connectionless communication with less overhead but no delivery guarantees. However, these custom protocols present limitations, such as the lack of real-time capabilities and difficulty in configuring Quality of Service (QoS) settings.

In contrast, ROS 2 introduces new client libraries, namely rclcpp and rclpy, with significant improvements over their ROS counterparts. More notably, ROS 2 transitions from custom protocols to the standardized Data Distribution Service (DDS) middleware for its communication infrastructure.

 doi:10.20944/preprints202410.1204.v3

This shift addresses the limitations of ROS middleware and will be explored in detail in the following section.

2.1.3. Limitations of ROS and the Motivation for ROS 2

Despite the significant contributions of ROS to the robotics community, certain limitations have prompted the development of a more advanced version. The deliberation to address these limitations began at the ROSCon 2014 conference, where the Open Source Robotics Foundation (OSRF) and the ROS community collectively identified areas of improvement.

- ***Single-Robot Focus*:** The primary limitation of ROS is its exclusive design for single-robot applications. The absence of inherent support for multi-robot systems presented a challenge for applications involving swarms or collaborative robots. This is primarily attributed to ROS's design requirement of a central ROS Master node, creating an isolated network per robot. In multi-robot applications, this necessitates additional packages to enable inter-robot cooperation. ROS 2 has addressed this limitation by removing the requirement for a ROS Master node, which we will discuss later.
- ***Lack of Real-Time Guarantees and Quality of Service*:** Another significant drawback of ROS is the absence of real-time guarantees and Quality of Service (QoS) profiles for prioritizing critical messages. ROS's reliance on TCP and UDP protocols translates to a best-effort service for message delivery without any assurance of successful transmission. The absence of message prioritization implies that critical and regular messages are treated on par, making it unsuitable for industrial-grade applications with stringent real-time and safety requirements, such as autonomous vehicles or unmanned aerial systems.
- ***Reliability and Scalability Concerns*:** Lastly, the reliability and scalability of ROS are challenged by its dependency on the central ROS Master node. The entire network fails if the ROS Master node crashes, establishing a single point of failure. This design also constrains the scalability of ROS.

In the following section, we delve into how ROS 2 has been designed to address and overcome these limitations.

2.2. ROS 2

2.2.1. ROS 2 Design Goals

The design of ROS 2 was primarily driven by the need to overcome the limitations of ROS, as discussed in the previous section.

Focus on Swarm Robotics

In contrast to ROS, which was primarily designed for standalone robots, ROS 2 has been engineered emphasizing swarm robotic applications. The inherent complexity of swarm robotics necessitates a fully distributed middleware that enables ad-hoc communication between multiple devices. This distributed communication structure allows for the efficient coordination and collective behavior that characterize a robot swarm. This shift towards a more distributed architecture is a key factor differentiating ROS 2 from its predecessor.

Real-Time and QoS Guarantees

ROS's lack of support for real-time guarantees and Quality of Service (QoS) were significant limitations, especially for applications with stringent timing and reliability requirements. To address this, real-time support and QoS guarantees were positioned at the forefront of ROS 2's design considerations. The middleware used in ROS 2 is expected to provide sophisticated QoS policies, ensuring the prioritized and timely delivery of messages based on their importance and urgency. These features are crucial for applications that require high reliability and deterministic execution, such as autonomous vehicles and industrial automation systems.

 doi:10.20944/preprints202410.1204.v3

Fast Prototyping and Cross-Platform Compatibility

Another important aspect of ROS 2's design is its emphasis on enabling a seamless transition from fast prototyping to production, coupled with cross-platform compatibility. This focus is designed to facilitate the rapid development and deployment of ROS 2 applications across a variety of platforms and into production environments. This compatibility and ease of deployment are expected to expedite the development cycle, a significant improvement over ROS, which presented challenges in these areas.

Middleware Selection

The above design goals significantly influenced the selection of the communication middleware for ROS 2. The need for a robust, efficient, and reliable communication infrastructure that could meet the advanced requirements of real-time processing, Quality of Service, and distributed system support guided the choice of middleware. After extensive deliberations in 2014, DDS was chosen for its inherent ability to meet these requirements. DDS, a standard for high-performance, scalable, and interoperable publish-subscribe communication, was deemed perfectly suited for the task. The choice of DDS as the middleware plays a pivotal role in ensuring that ROS 2 successfully addresses the limitations of ROS and meets its design objectives.

2.2.2. ROS 2 Architecture

ROS 2's architecture, as presented in Figure 6, retains similarities with ROS in certain aspects but also presents a significant evolution to address ROS's limitations.

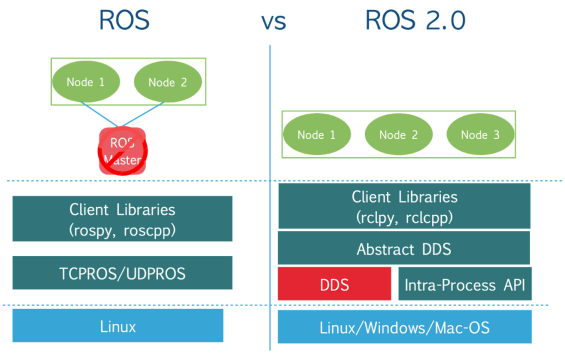

Figure 6. ROS vs. ROS 2 Architecture.

Distributed by Design

In contrast to ROS's centralized approach, ROS 2 is designed to be inherently distributed. It eliminates the need for a central ROS master node, thereby addressing the single point of failure and scalability issues present in ROS. Instead, ROS 2 nodes communicate directly with each other, facilitated by the underlying DDS middleware. This decentralized design is more fitting for multi-robot systems and enables more efficient and reliable communication in large-scale, distributed robotic applications.

Maintained Modularity and Enhanced Scalability

Like ROS, ROS 2 upholds the principle of modularity, where each node operates independently and performs a specific task. However, ROS 2 extends this concept to incorporate the notion of 'Managed Nodes' that can be governed by a lifecycle manager, thereby providing more control over the system's state and behavior. Furthermore, the distributed nature of ROS 2 enhances scalability, enabling the system to handle a more considerable number of nodes efficiently.

 doi:10.20944/preprints202410.1204.v3

Extended Communication Patterns

ROS 2 inherits the primary communication patterns of ROS, with some notable improvements and extensions.

- ***Publisher-Subscriber (Topics):*** The Publisher-Subscriber model, implemented via Topics, remains a vital part of ROS 2. However, ROS 2 introduces the concept of Quality of Service (QoS) policies for Topics, allowing more precise control over message delivery.
- ***Synchronous Client/Server (Services):*** The synchronous client/server model, facilitated by Services, is also retained in ROS 2. Yet, similar to Topics, Services in ROS 2 support QoS settings, enabling reliable and timely communication based on specific requirements.
- ***Asynchronous Client/Server (Actions):*** The asynchronous client/server model, formerly denoted by ActionLib in ROS, has been incorporated into the core of ROS 2 as Actions. Actions in ROS 2 provide a more streamlined interface and support QoS policies, offering improved reliability and flexibility.
- ***Interfaces:*** In ROS 2, the concept of *Interfaces* is introduced, which encapsulates the functionality of Topics, Services, and Action messages. Interfaces replace the separate message types used in ROS, namely `rosmsg` and `rossrv`. This consolidation is an attempt to simplify the message generation process and to facilitate compatibility between different communication models. Interfaces enhance the consistency and maintainability of the ROS 2 communication model, as developers only need to familiarize themselves with a single message type, regardless of the communication pattern they employ. This simplification also enhances code portability between different ROS 2 applications, leading to a more robust and flexible system design.

Enhanced Client Libraries and Advanced Communication Middleware

ROS 2 introduces updated client libraries, rclcpp for C++ and rclpy for Python, which are more robust, maintainable, and feature-rich compared to their ROS counterparts. The most significant shift in ROS 2 is adopting DDS as its communication middleware. DDS is an industry-standard middleware designed for high-performance, real-time, scalable, and interoperable publish-subscribe communication. The use of DDS in ROS 2 addresses the limitations of the customs protocols used in ROS. It provides native support for real-time communication, configurable QoS policies, and robust security mechanisms. Furthermore, the standardized nature of DDS facilitates interoperability with other systems and technologies, broadening the scope and applicability of ROS 2.

In this section, we have explored the evolution of ROS into ROS 2, detailing its architectural advancements in middleware efficiency, real-time performance, security, and scalability. Through a systematic analysis, we have demonstrated how ROS 2 addresses the limitations of ROS 1 by adopting DDS as a standardized communication middleware, introducing enhanced client libraries, and transitioning to a fully distributed architecture. These improvements directly respond to **Research Question 1**, highlighting the ways in which ROS 2 refines the core infrastructure of robotic systems to meet modern requirements.

While these architectural enhancements form the foundation of ROS 2's evolution, its impact extends beyond infrastructure alone. In the next section, we examine non-architectural differences between ROS and ROS 2, such as programming language support, system design paradigms, and usability considerations, further contextualizing the practical implications of these advancements.

3. ROS vs. ROS 2

While Section 2 explored the architectural advancements of ROS 2, this section focuses on the non-architectural differences that impact its usability, flexibility, and adoption. These differences span programming language support, execution models, transformation handling, navigation frameworks, security mechanisms, and platform compatibility.

This section directly contributes to Research Question 1 by further analyzing how ROS 2 improves upon ROS 1 beyond its core architecture. Understanding these enhancements provides insights into

 doi:10.20944/preprints202410.1204.v3

developer experience, system interoperability, and practical deployment considerations, all of which influence ROS 2's suitability for real-world robotic applications. Through this analysis, we highlight how these refinements collectively enhance performance, security, scalability, and cross-platform support, further strengthening ROS 2's position as the next-generation standard for robotic software development.

3.1. Programming Languages (C++ and Python Differences)

In ROS 2, both C++ and Python programming languages are embraced, but their utilization differs from ROS.

C++ in ROS 2: ROS 2 offers enhanced support for modern C++ features, including move semantics and lambda functions. This improved support leads to better performance and code readability in ROS 2 applications. Developers can leverage new language features and libraries that were previously unavailable in the older C++03 version. Notably, certain core libraries in ROS 2 have been upgraded to support C++14, expanding the range of modern C++ capabilities that can be utilized. These advancements empower developers to take full advantage of the latest features and functionalities offered by the C++ programming language.

Python in ROS 2: In contrast to ROS, which predominantly utilized Python 2, ROS 2 adopts Python 3 as its primary version. Consequently, ROS 2 packages are primarily written in Python 3. This transition allows developers to leverage recent Python enhancements, such as type hints and improved asyncio support. By aligning with the broader Python language community, ROS 2 enables seamless integration with other Python-based tools and libraries. However, it's important to note that compatibility updates are required for any existing Python 2 code intended for ROS 2 usage.

Language-Agnostic Communication Interfaces: One of the notable advancements in ROS 2 is the introduction of the Abstract DDS interface, a language-agnostic communication protocol. This protocol enables seamless interaction and communication between nodes written in different programming languages. With DDS, nodes can exchange messages and information efficiently, regardless of whether they are implemented in C++, Python, or any other supported language. This language-agnostic approach facilitates interoperability and collaboration among diverse robotic systems.

These disparities in language support between ROS and ROS 2 emphasize the significance of staying abreast of the latest language features and standards. By accommodating modern C++ and Python versions, ROS 2 empowers developers to fully capitalize on the latest advancements in these languages, enhancing the quality and efficiency of their robotic applications. Ultimately, the choice between C++ and Python hinges on the specific requirements of the target robotics system, and ROS 2's language flexibility caters to diverse needs.

3.2. Executors

In ROS 2, executors play a pivotal role in managing the execution of tasks within a node's lifecycle. They facilitate communication between nodes and ensure the execution of callbacks for incoming messages, services, and actions, significantly influencing the overall ROS 2 architecture.

ROS 2 encompasses three executor types each offering distinct characteristics and benefits:

- `SingleThreadedExecutor`: This executor executes all callbacks within a single thread, employing a round-robin scheduling approach to ensure fairness between tasks. However, computationally intensive tasks may experience slower execution due to the shared thread among all tasks.
- `MultiThreadedExecutor`: Designed to handle high workloads and computationally intensive tasks, this executor employs multiple threads to execute callbacks concurrently. While it provides improved performance, careful synchronization is necessary to avoid potential race conditions or deadlocks.
- `StaticSingleThreadedExecutor`: This executor is similar to `SingleThreadedExecutor` but is specifically designed for nodes with a fixed set of entities (such as sensors, actuators, or other components) known during the compilation process. It optimizes runtime costs by scanning the

 doi:10.20944/preprints202410.1204.v3

structure of the node, including subscriptions, timers, service servers, and action servers, only once during node addition. Unlike other executors, it does not regularly scan for changes. Therefore, the StaticSingleThreadedExecutor is most suitable for nodes that create all their subscriptions, timers, and other entities during initialization and do not dynamically add or remove them during runtime. By eliminating the need for continuous scanning, it improves performance and efficiency in such static systems.

In contrast, ROS employs a " spinner " mechanism to manage callback execution within a node's lifecycle. Two primary spinner types are available:

- **ros::spin():** This single-threaded spinner sequentially processes callbacks within a single thread until the node is shut down. It represents the simplest and most commonly used spinner in ROS.
- **ros::AsyncSpinner:** This multi-threaded spinner concurrently processes callbacks using multiple threads, suitable for scenarios involving computationally intensive tasks or varying callback execution times.

While spinners in ROS serve a similar purpose to executors in ROS 2, the latter offers advanced features and greater flexibility in task execution, including the StaticSingleThreadedExecutor for static systems. Moreover, introducing the DDS middleware in ROS 2 necessitates a refined and adaptable approach to task execution, effectively addressed by the concept of executors.

Using executors in ROS 2 provides efficient task management and optimal utilization.

3.3. Transformations: tf vs tf2

The original library for managing transforms in RO), known as tf [18], utilizes a static transform tree to outline the relationships between coordinate frames. However, tf exhibits certain limitations, including susceptibility to errors during concurrent multi-thread access to the transform tree and inefficiency when handling large trees.

In contrast, tf2, an advanced version, employs a more efficient data structure for the transform tree representation, enhancing both speed and robustness. It is designed to be thread-safe, thereby eliminating the risk of errors during simultaneous access by multiple threads.

A novel feature introduced in tf2 is the Buffer, a sophisticated data structure that oversees the transform tree. It offers APIs for querying transforms between coordinate frames, thereby increasing the versatility and usability of the system.

A significant distinction between tf and tf2 lies in their support for coordinate representations. While tf is limited to the XYZ Euler angle representation for rotations, tf2 supports multiple representations. These include quaternions, rotation matrices, and angle-axis representations, thereby offering a more comprehensive and flexible system for users.

Moreover, tf2 introduces several new features, including support for non-rigid transforms and the capability to interpolate between transforms. It also enhances debugging and visualization support, thereby simplifying the process of diagnosing issues with the transform tree.

To summarize, due to its superior performance, thread safety, and additional features, tf2 is the recommended library for managing transforms in ROS 2. Its design and functionality improvements over tf make it a more robust and efficient tool for managing coordinate frame relationships.

3.4. ROS 2 Navigation Stack: Main Features and Comparison with ROS 1

The ROS 2 Navigation Stack, also known as Navigation2, is a significant upgrade from the ROS 1 navigation stack, with several key features that enhance its functionality and usability. A comprehensive description of the ROS 2 Navigation Stack can be found in the following reference: [19]. In what follows, we present a systematic and methodological comparison of the ROS 1 and ROS 2 navigation stacks:

- **Task Orchestration using Behavior Trees** introduces the use of a *behavior tree* for task orchestration, a feature absent in ROS 1. This tree orchestrates planning, control, and recovery tasks, with

 doi:10.20944/preprints202410.1204.v3

each node invoking a remote server to compute one of these tasks using various algorithm implementations.

- **Modularity and Configurability:** Navigation2 is designed to be highly modular and configurable, a marked improvement over ROS 1. It employs a behavior tree navigator and task-specific asynchronous servers, each of which is a ROS 2 node hosting algorithm plugin. These plugins are libraries dynamically loaded at runtime, allowing for unique navigation behaviors to be created by modifying a behavior tree.
- **Managed Nodes:** ROS 2 introduces the concept of Managed Nodes, servers whose life-cycle state can be controlled. Navigation2 exploits this feature to create deterministic behavior for each server in the system, a feature not present in ROS 1.
- **Feature Extensions:** Navigation2 supports commercial feature extensions, allowing users with complex missions to use Navigation2 as a subtree of their mission. This is a unique feature not found in ROS 1.
- **Multi-core Processor Utilization:** Unlike ROS 1, Navigation2 architecture leverages multi-core processors and the real-time, low-latency capabilities of ROS 2. This allows for more efficient processing and faster response times.
- **Algorithmic Refreshes:** Navigation2 focuses on modularity and smooth operation in dynamic environments. It includes the Spatio-Temporal Voxel Layer (STVL), layered costmaps, the Timed Elastic Band (TEB) controller, and a multi-sensor fusion framework for state estimation, Robot Localization. Each of these supports holonomic and non-holonomic robot types, a feature not as developed in ROS 1.
- **State Estimation:** Navigation2 follows ROS transformation tree standards for state estimation, making use of modern tools available from the community. This includes Robot Localization, a general sensor fusion solution using Extended or Unscented Kalman Filters. This is a more advanced approach compared to ROS 1.
- **Quality Assurance:** Navigation2 includes tools for testing and operations, such as the Lifecycle Manager, which coordinates the program lifecycle of the navigator and various servers. This manager steps each server through the managed node lifecycle: inactive, active, and finalized. This systematic approach to quality assurance is a significant upgrade from ROS 1.

Navigation2 builds on the successful legacy of ROS Navigation but with substantial structural and algorithmic refreshes. It is more suitable for dynamic environments and a wider variety of modern sensors, making it a more advanced and versatile navigation stack than its predecessor, ROS 1.

3.5. ROS 2 Security

The key difference in security between ROS and ROS 2 is that ROS 2 has been designed with security in mind from the ground up, whereas security was not a primary consideration in the initial development of ROS.

In ROS, security mechanisms such as authentication and encryption were not implemented by default, leaving systems vulnerable to potential security threats. However, ROS users could implement security measures manually by using third-party libraries and plugins.

ROS 2, on the other hand, has a built-in security framework that provides authentication, encryption, and access control by default. This framework enables ROS 2 to support secure communication between nodes and across networks, even when communicating with nodes that may not support security natively.

SROS2 (Secure ROS2) [20] is a security extension for ROS 2 that provides additional security features beyond those provided by the ROS 2 security framework. SROS is designed to address some of the limitations of the ROS 2 security framework and to provide enhanced security for critical robotic applications. SROS provides a more flexible and configurable access control system than the default ROS 2 permission system, allowing administrators to define fine-grained access policies for topics,

 doi:10.20944/preprints202410.1204.v3

services, and nodes. This can help to ensure that sensitive data and resources are protected from unauthorized access.

Overall, by implementing SROS in ROS 2, users can benefit from additional security features that are not available in ROS. This can help to ensure the security and integrity of critical robotic applications while also providing a more flexible and configurable security solution that can be tailored to the specific needs of each application.

3.6. Comparison of Platform Support in ROS and ROS 2

ROS and its successor, ROS 2, are widely used frameworks for developing robotic systems. This section explores the level of support and compatibility with different operating systems in both frameworks. Specifically, we will examine the support for Windows, macOS, and Linux platforms, highlighting the advancements made in ROS 2.

- **Windows Support:** When it comes to Windows support, ROS has experimental compatibility, while ROS 2 boasts more comprehensive and reliable support for Windows 10. The enhanced support in ROS 2 allows developers to leverage the framework more effectively on Windows machines, ensuring a stable and efficient environment for building robust robotic applications.
- **macOS Support:** Both ROS and ROS 2 offer official support for macOS. However, ROS 2 surpasses its predecessor in terms of compatibility with the latest macOS versions. By capitalizing on the latest macOS features, such as the Metal graphics API, ROS 2 maximizes performance in specific applications. This advanced compatibility empowers developers to fully exploit the potential of macOS when constructing sophisticated robotics systems.
- **Linux Support:** Both ROS and ROS 2 offer comprehensive support for Linux, with official support for various popular distributions. However, ROS 2 takes a more modular and flexible approach, making porting the framework to different Linux distributions and architectures easier. This flexibility is particularly advantageous in heterogeneous computing environments, allowing developers to deploy ROS 2 on various Linux systems.

The utilization of the DDS, which inherently possesses cross-platform capabilities, significantly facilitated the seamless compatibility of ROS 2 with Windows, macOS, and Linux. In contrast, ROS 1 relied on the specific TCPROS and UDPROS protocols, which were more tightly coupled with the Linux environment, limiting its cross-platform compatibility.

In this section, we have explored non-architectural advancements in ROS 2, focusing on improvements in programming language support, execution models, transformation handling, navigation frameworks, security mechanisms, and platform compatibility. These enhancements directly address **Research Question 1**, demonstrating how ROS 2 refines developer experience, flexibility, and cross-platform usability in ways that go beyond core architectural improvements. By adopting modern C++ and Python standards, improving execution models through advanced executors, strengthening security, and leveraging DDS for seamless cross-platform support, ROS 2 removes many limitations previously hindering its adoption in real-world applications.

However, while these enhancements improve usability and system design, meeting the stringent requirements of modern robotics—such as real-time execution, determinism, safety, and fault tolerance—requires deeper exploration at the system level. The next section delves into these advanced technical aspects by examining middleware optimizations, real-time scheduling, security, and multi-robot coordination. These foundational elements play a crucial role in ensuring that ROS 2 can support high-performance, safety-critical, and large-scale robotic deployments, ultimately shaping its future adoption in both research and industry.

4. Advanced ROS 2: Technical Innovations, Deployments, and Adoption Challenges

This section offers a thorough analysis of key technical and practical facets in ROS 2, presenting four in-depth areas that collectively address all three research questions (**RQ1, RQ2, and RQ3**). First,

 doi:10.20944/preprints202410.1204.v3

we examine **middleware innovations** such as DDS-based QoS, zero-copy techniques, and formal verification (**RQ1**), highlighting their growing presence in industrial sectors (**RQ2**) and adoption barriers due to complexity and scalability concerns (**RQ3**). Next, we focus on **real-time scheduling and hardware acceleration**, demonstrating how advanced executors, FPGAs, and GPUs can enable near-deterministic performance (**RQ1**) for mission-critical deployments (**RQ2**), despite organizational and cost-related hurdles (**RQ3**). We then explore **security, privacy, and safety** in ROS 2, underscoring technical solutions and industry adoption patterns (RQ 1–2) while noting overhead and integration challenges (**RQ3**). Finally, we review **multi-robot and distributed systems**—from bridging methods to edge–cloud offloading—showcasing progress and the need for further standardization. Figure 7 shows sunburst chart of literature taxonomy for those technical facets of ROS 2

Figure 7. ROS 2 Core-Concepts Taxonomy.

4.1. Advanced ROS 2 Middleware

4.1.1. RQ1 – Technical Foundations and Innovations

DDS Backbone and QoS Strategies: ROS 2 builds on a DDS-based communication layer, enabling flexible *quality-of-service (QoS)* configurations to address latency, reliability, and scheduling demands. As summarized in Table 2, most works focus on leveraging or extending these QoS policies—such as `RELIABLE` vs. `BEST_EFFORT`, `KEEP_LAST` vs. `KEEP_ALL`, and deadline constraints—to match application needs, including real-time data fusion, sensor streaming, and distributed control [22,24,25].

 doi:10.20944/preprints202410.1204.v3

Table 2. Taxonomy of ROS 2 Middleware Research.

Category	Representative Papers
QoS and Performance Approaches	[21–25]
Modeling and Formal Verification	[26–29]
Alternative or Enhanced Middlewares	[30–33]
Middleware Comparisons	[34–36]

Because of DDS's pluggable architecture, *alternative or enhanced middlewares* have also emerged (e.g., Zenoh, Kafka/MQTT bridges) [30,32]. Researchers propose these solutions to optimize network usage in edge or wireless scenarios [33].

Zero-Copy and Shared Memory Transport: In ROS 2, zero-copy and shared-memory transport mechanisms offer a significant performance boost by eliminating the need for serialization, thereby reducing overhead for large data payloads such as camera feeds or point clouds [22,23]. This approach allows nodes to access shared data directly, which can drastically reduce latency and improve throughput in high-bandwidth applications. However, leveraging these benefits comes with its own set of challenges: managing concurrency becomes more complex when multiple nodes access shared buffers, often requiring intricate synchronization strategies to prevent race conditions or data corruption [21,24]. Consequently, while zero-copy and shared-memory transports are promising, the ROS 2 community continues to explore robust solutions that balance high performance with safe, reliable data sharing in multi-node environments.

Modeling, Formal Verification, and Comparisons: In ROS 2, modeling, formal verification, and comparative performance studies play a crucial role in ensuring system robustness and efficiency. Researchers have increasingly adopted formal verification techniques—such as timed automata and model-checking frameworks—to rigorously establish properties like deadlock freedom, timely deadline adherence, and reliable end-to-end message delivery [26,28,29]. These methods not only provide strong guarantees about system behavior but also help uncover subtle flaws that might be overlooked during traditional testing, albeit at the expense of increased modeling complexity and computational resources. Simultaneously, empirical studies that compare latency, throughput, and CPU usage under varied QoS settings or network conditions offer valuable insights into how different middleware configurations perform in real-world scenarios [34–36], thereby guiding developers in optimizing ROS 2 deployments for diverse application needs.

Relevant Equations, Algorithms, and Results: Equations 1–3 illustrate typical metrics, such as:

$$L_{\text{total}} = L_{\text{comm}} + L_{\text{proc}}, \tag{1}$$

$$P_{\text{success}} = (1 - p_{\text{loss}})^n, \tag{2}$$

$$L_{\text{sec}} = L_{\text{base}} + \kappa \cdot \text{MsgSize}, \tag{3}$$

while pseudo-code algorithms (Algorithms 1–6) demonstrate core middleware functionalities, ranging from real-time dataflow scheduling to safe zero-copy transfers (Sec. 4.1). Reported testbed results show up to 100× lower latency in zero-copy scenarios [22], though encryption overhead can inflate communication delays up to 2× or 3× [25].

 doi:10.20944/preprints202410.1204.v3

Algorithm 1 Real-Time Dataflow Scheduling

```
Initialize dataflow_graph, deadlines
for each pipeline_stage in dataflow_graph do
    pipeline_stage.priority ← assign_priority(deadlines)
end for
while system_active do
    for each stage in sorted_by_priority(dataflow_graph) do
        if stage.ready_to_run() then
            stage.execute()
        end if
    end for
    wait_for_next_cycle()
end while
```

Algorithm 2 Parallel Node Launch with Shared Memory

```
shared_mem_id ← allocate_shared_pool(size=SM_SIZE)
for each node_name in node_list do
    process ← spawn_node(node_name, shared_mem_id)
    set_affinity(process, core_map[node_name])
    add_to_supervisor(process)
end for
start_supervisor()
```

Algorithm 3 Safe Zero-Copy Transfer with Locking

```
procedure PUBLISH_DATA(buffer)
    lock(mutex_buffer)
    write_payload(buffer, new_data)
    set_ready_flag(buffer)
    unlock(mutex_buffer)
    notify_subscribers()
end procedure

procedure SUBSCRIBE_DATA(buffer)
    if wait_for_ready(buffer) then
        lock(mutex_buffer)
        local_copy ← read_payload(buffer)
        clear_ready_flag(buffer)
        unlock(mutex_buffer)
        return local_copy
    end if
end procedure
```

Algorithm 4 Mode Change for Real-Time Streams

```
procedure SWITCH_MODE(current_mode, new_mode)
    broadcast_pause(current_mode)
    wait_all_stages_stop(current_mode)
    reconfigure_params(new_mode)
    set_active_mode(new_mode)
    broadcast_resume(new_mode)
end procedure
```

 doi:10.20944/preprints202410.1204.v3

Algorithm 5 Fault-Tolerant Node Heartbeat

```
Initialize node_status_map
while true do                                   ▷ Runs every HEARTBEAT_INTERVAL ms
    for each node in node_list do
        send_heartbeat_request(node)
    end for
    collect_responses()
    for each node in node_list do
        if no_recent_response(node) then
            attempt_restart(node)
            log_fault_event(node)
        end if
    end for
end while
```

Algorithm 6 Distributed Service Discovery

```
procedure DISCOVER_SERVICES
    local_cache ← local_discovery()
    peers ← get_peer_list()
    for each peer in peers do
        remote_list ← request_discovery(peer)
        merge(local_cache, remote_list)
    end for
    return local_cache
end procedure

procedure PUBLISH_SERVICES(my_services)
    peers ← get_peer_list()                     ▷ Make sure we have the peer list
    for each peer in peers do
        send_service_list(peer, my_services)
    end for
end procedure
```

4.1.2. RQ2 – Industry Trends and Real-World Deployments

Although these middleware advances are grounded in academic research, many industrial domains have begun integrating specialized QoS and DDS features in real-world systems. For instance, large-scale autonomous vehicle platforms rely on robust QoS parameters to handle high-bandwidth LiDAR and camera data with minimal packet loss [23]. In industrial automation, real-time data exchange and consistent reliability are critical for coordinating robotic arms or conveyor subsystems; this has prompted interest in formal verification approaches to ensure safe concurrency when nodes share memory [27,28].

Adoption of Alternative RMWs and Legacy Interoperability: Some companies testing multi-robot fleets leverage *Zenoh-based RMW* (see Figure 8 for Zenoh architecture visualization) to reduce overhead in wireless or mobile deployments [30]. By rethinking the traditional DDS discovery process, Zenoh advertises only resource interests—rather than the full set of publisher and subscriber details—thereby compressing discovery data and significantly reducing network traffic. Experimental results indicate that this approach can lower discovery overhead by up to 97% in worst-case scenarios, and even achieve reductions as high as 99.97% when combining resource generalization with warm start configurations. Moreover, Zenoh supports Internet-scale routing and peer-to-peer communication, integrating transparently into existing ROS 2 systems via the zenoh-plugin-dds. Meanwhile, bridging strategies (ROS 1–ROS 2 or DDS–MQTT) continue to emerge, as industry partners may still rely on legacy ROS 1 components or prefer to route select messages through an event-driven pipeline [34]. These transitions reveal an ongoing trend of partial adoption, where advanced QoS or zero-copy

 doi:10.20944/preprints202410.1204.v3

features are applied selectively to high-throughput data channels while simpler or legacy topics remain unchanged.

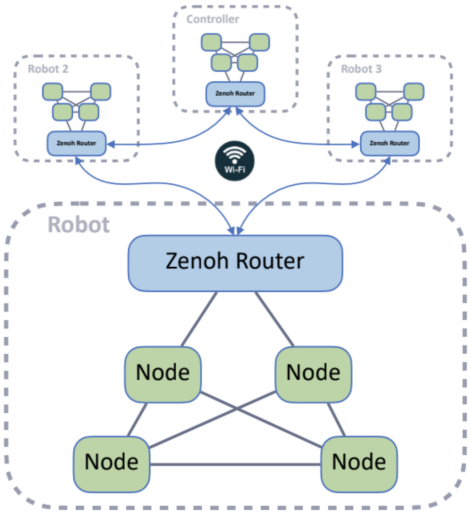

Figure 8. Default Configuration for Zenoh Sessions in ROS 2 Jazzy.

Use Cases in Healthcare and Collaborative Robotics: Healthcare robotics highlights the need for *formal verification*, especially in surgical or assistive systems with strict safety and reliability requirements [29]. Collaborative robots in manufacturing also benefit from deterministic communication guaranteed by robust middleware setups, enabling safe human–robot coexistence under real-time constraints [37].

4.1.3. RQ3 – Adoption Challenges and Future Solutions

Despite the technical promise, several hurdles limit broader industry uptake:

Technical Barriers: Middleware Complexity. DDS QoS parameters, concurrency control for zero-copy, and formal verification toolchains can be daunting. As Table 3 indicates, a unified interface for memory-safe zero-copy under crash-fault conditions remains elusive [24].

Table 3. Challenges and Future Directions in Advanced ROS 2 Middleware.

Challenges / Gaps	Future Directions
Hybrid HPC + reliability demands	Unified interfaces for memory-safe zero-copy in crash-fault conditions
Dynamic adaptivity	Advanced run-time partial reconfiguration beyond basic mode changes
Large-scale scalability	Efficient discovery / orchestration for hundreds of distributed nodes
Formal standardization	Merge advanced HPC & fault-tolerant features into mainstream ROS 2 or DDS

Organizational and Skill Gaps: Some organizations lack the in-house expertise to configure QoS or verify concurrency properties. In many real-world settings, default middleware settings remain unchanged—leading to suboptimal performance or, in the worst case, unintended reliability issues [35].

Potential Solutions and Ongoing Efforts: Potential solutions and ongoing efforts in ROS 2 are taking a multi-pronged approach to address current challenges. One key direction is the move toward standardized QoS profiles and APIs, as initiatives within the ROS 2 community, such as those from the ROS 2 Working Groups, work to establish common presets for real-time, high-throughput, or secure applications—thereby reducing the guesswork involved in configuration. Additionally, there is

 doi:10.20944/preprints202410.1204.v3

growing interest in integrating formal verification tools directly into the build pipeline; embedding techniques like model-checking or timed automata analyses could streamline adoption and enhance system robustness, although scaling these methods remains a challenge [28]. Moreover, efforts are underway to bring cutting-edge research into mainstream ROS 2 distributions. For instance, features like shared-memory transport have already begun to appear in newer releases, and future versions are expected to incorporate even more advanced HPC or fault-tolerant protocols [32].

A broader consensus is that bridging these technical breakthroughs into official ROS 2 and DDS releases is essential for large-scale adoption. In particular, robust default configurations, improved documentation, and user-friendly tooling around zero-copy, QoS tuning, and formal verification would considerably lower adoption barriers for both academic labs and industrial teams.

4.2. Real-Time Scheduling and Hardware Acceleration in ROS 2

4.2.1. RQ1 – Technical Foundations and Innovations

Brief Overview and Taxonomy: Modern robotic applications often demand both deterministic behavior and high computational throughput. As shown in Table 4, solutions in this domain can be categorized into *real-time scheduling & analysis*, *hardware acceleration*, *micro-ROS for constrained platforms*, and *application-level case studies*. A critical component is the choice and configuration of the *ROS 2 executor*, which schedules callbacks under different threading or priority schemes [38,43], while hardware acceleration techniques (FPGA/GPU) target resource-intensive tasks [60,65].

Table 4. Taxonomy of Real-Time and High-Performance Approaches in ROS 2-based Systems.

Category / Subcategory	Relevant References
1. Real-Time Scheduling & Analysis	
1.1 Single-Threaded Executor Analysis	[37–39]
1.2 Multi-Threaded Executor Analysis	[40,41]
1.3 Priority Assignment & Chain-Aware Scheduling	[42–45]
1.4 End-to-End Latency & Response-Time Analysis	[46–57]
1.5 Containerization & Microservices	[58]
1.6 Edge/Cloud Offloading	[59]
2. Hardware Acceleration & Communication	
2.1 FPGA-based Acceleration	[60–64]
2.2 GPU-based Resource Management	[65–67]
3. Micro-ROS & Resource-Constrained Platforms	
	[68,69]
4. Applications & Case Studies	
4.1 Industrial & CNC	[70]
4.2 Multi-Agent & UAV	[71]

Key Real-Time Scheduling Concepts: Compared to ROS 1's single-threaded or round-robin executor, ROS 2 introduces multi-threaded executors and chain-aware priority assignment [42,45]. These approaches address end-to-end latencies through explicit priority mappings, specialized buffer management, and concurrency control. Empirical data show that naive round-robin scheduling can result in hundreds of milliseconds in worst-case latency; chain-based or priority-aware solutions reduce those latencies to tens or single digits of milliseconds [46,47].

Hardware Acceleration and Micro-ROS: Hardware acceleration strategies push heavy computations (e.g., computer vision, SLAM) to GPUs or FPGAs, reporting speedups between 2× and 20× [61,64]. Meanwhile, micro-ROS extends ROS 2 to microcontrollers [68,69], enabling real-time control loops on resource-constrained boards. Achieving deterministic performance in such systems typically

 doi:10.20944/preprints202410.1204.v3

hinges on specialized QoS setups, shared-memory optimizations, and tight integration of scheduling with hardware resources.

Equations and Algorithms Illustrating Real-Time Analysis: Numerous equations model end-to-end latency, scheduling interference, and buffer requirements [43,46]. A typical end-to-end latency formula might sum the callback execution (C_i) and interference (I_i) across a chain of nodes [53] as shown in Eq. 4:

Likewise, hardware-acceleration speedups are often computed as $T_{\text{CPU}}/T_{\text{FPGA}}$ [60], underscoring the gains from offloading. Algorithmic snippets 7–9 (e.g., multi-thread executors, aggregator nodes) frequently appear in studies proposing next-generation real-time frameworks [40,41].

$$L_{\text{end2end}} = \sum_{i \in \text{chain}} (C_i + I_i), \tag{4}$$

where C_i is a callback's execution time and I_i the scheduling interference.

Algorithm 7 Initial budget estimate

Input: set of executors $\mathcal{E}$
for all executor $e \in \mathcal{E}$ without a budget **do**
  needed $\leftarrow \sum_{c \text{ served by } e} \text{rbf}_c(\text{horizon})$
  $\text{bw}(e) \leftarrow \text{needed}/\text{horizon}$
end for
for all callback c **with** unbounded response times **do**
  $e \leftarrow$ executor serving c
  needed $\leftarrow \text{rbf}_c(\text{horizon}) - \text{sbf}_e(\text{horizon})$
  if $\text{bw}(e) = 1$ **then**
    DegradeChain(c) ▷ example function call
  else
    $\text{bw}(e) \leftarrow \min\left(\text{bw}(e) + \frac{\text{needed}}{\text{horizon}}, 1\right)$
  end if
end for
if no feasible partitioning for budgets found: degrade some chain

Algorithm 8 Assignment improvement heuristic

Input: set of executors res affecting response times
while (chain latency $>$ goal) **do**
  for all $e \in$ res **do**
    $d(e) \leftarrow \sum_{c \text{ served by } e}(\text{RT}(c) - \text{RT}_{100\%}(c))$
  end for
  Sort res by decreasing d(e)
  for all e in res (in sorted order) **do**
    if $\text{bw}(e) = 1$ **then**
      remove e from res; continue
    end if
    oldBW $\leftarrow \text{bw}(e)$
    $\text{bw}(e) \leftarrow \min(\text{bw}(e) + 0.05, 1.0)$
    if (new partitioning is feasible) **then**
      candidate found; **break** inner loop
    else
      $\text{bw}(e) \leftarrow$ oldBW ▷ restore old value
    end if
  end for
  if no candidate found: degrade chain
end while

 doi:10.20944/preprints202410.1204.v3

Algorithm 9 Waiting queue pool updating algorithm

Input: queue pool Φ, old mapping (Θ, M), new mapping (Θ', M')
if $M' > M$ **then**
  for $i \in [M,\ M' - 1]$ **do**
    Add Q_i to Φ
  end for
end if
for all mapping $(id, wq) \in \Theta'$ **do**
  find wq' such that $(id, wq') \in \Theta$ and $wq' \neq wq$
  if $wq' \neq wq$ **then**
    for all request in $Q_{wq'}$ **do**
      if request.$id = id$ **then**
        Remove request from $Q_{wq'}$
        Insert request into Q_{wq} (by timestamp ordering)
      end if
    end for
  end if
end for
if $M' < M$ **then**
  for $j \in [M',\ M - 1]$ **do**
    Delete Q_j from Φ
  end for
end if

Empirical Results and Performance Gains: Across the literature, specialized scheduling (priority- or chain-based) can yield up to **80**$\times$ reductions in worst-case latency [43], while GPU/FPGA acceleration brings 5–20$\times$ speedups in vision or SLAM tasks [65,66]. Micro-ROS solutions can maintain $\sim$100 Hz control loops on low-power boards, provided that QoS settings limit packet loss [69].

4.2.2. RQ2 – Industry Trends and Real-World Deployments

Industrial Automation and CNC: In industrial and CNC (computer numerical control) applications, real-time monitoring and rapid response are critical. Studies show that by combining priority-based executors and containerization, latencies below 1 ms can be achieved on production lines [70]. Deployed systems often rely on *hardware accelerators* to handle large sensor streams or complex inverse-kinematics calculations in real time.

Multi-Agent and UAV Deployments: For multi-agent robotics and UAV swarms, specialized scheduling ensures consistency in sensor fusion and flight control across distributed nodes [71]. Edge or cloud offloading can reduce on-board CPU usage by up to 40% [59], albeit at the cost of additional network latency. Container-based orchestration simplifies deployment of real-time containers, while micro-ROS extends basic flight-control loops onto small embedded hardware for distributed sensing.

GPUs, FPGAs, and Resource Mapping in Practice: Industrial integrators are also exploring hybrid CPU–FPGA solutions, especially in advanced manufacturing lines with machine vision or safety-critical controls [63]. GPU-based pipelines appear in autonomous driving startups, where dense LiDAR or camera data must be processed at high frame rates [66]. On the smaller scale, micro-ROS boards with real-time OS patches tackle simpler tasks (e.g., tactile sensors, low-level motor control).

4.2.3. RQ3 – Adoption Challenges and Future Solutions

Technical Barriers: Despite significant performance gains, multiple technical challenges hinder widespread real-time adoption. One major barrier is the complexity of executor concurrency; multi-threaded executors, while beneficial for performance, can incur substantial overhead if locks are not managed carefully [40]. Additionally, the tuning of Quality of Service (QoS) parameters remains fragile. The reliance on vendor-specific DDS implementations, coupled with incomplete standardization, often forces trial-and-error configurations that undermine consistent real-time guarantees [35]. Furthermore,

 doi:10.20944/preprints202410.1204.v3

the imperative for robust security introduces a conflict between performance and protection. Real-time constraints can be at odds with the overhead introduced by encryption or intrusion-detection measures, complicating the balance between securing data paths and maintaining optimal system performance [57].

Industrial and Organizational Challenges: Industrial and organizational challenges are multifaceted. One major hurdle is the cost associated with specialized hardware; for instance, FPGAs, high-end GPUs, or real-time patched operating systems can be prohibitively expensive and often lead to vendor lock-in, thereby hindering broader adoption [61]. Additionally, there is a noticeable lack of unified design practices across teams. This gap in expertise makes it difficult to seamlessly integrate real-time scheduling frameworks with high-performance computing acceleration, frequently resulting in implementations that are only partial or suboptimal.

Potential Solutions and Roadmap: Table 5 highlights several promising future directions. One potential solution is to encourage the ROS 2 community to adopt standard real-time QoS profiles. By using universal QoS presets for HPC or low-latency tasks, the configuration process can be simplified, reducing the guesswork that often hinders effective implementation. In addition, dynamic reconfiguration offers a compelling pathway forward. Techniques such as the partial reprogramming of FPGAs or automated container orchestration can enable adaptive real-time pipelines, as explored in [63]. Enhancing security is another crucial aspect; integrating selective encryption or hardware-based enclaves can help protect critical data paths while incurring minimal performance overhead. Finally, easing the transition for industrial teams by consolidating tooling—such as using wizards and best-practice containers—can streamline the deployment of micro-ROS and edge offloading solutions, thereby accelerating the adoption of microcontroller-based or edge–cloud systems.

Table 5. Challenges and Future Directions in Real-Time and Hardware-Accelerated ROS 2.

Challenges / Gaps	Future Directions
Executor concurrency overhead	Multi-thread or multi-executor scheduling with formal concurrency bounds; minimal lock designs
Fragile or vendor-specific DDS QoS	Standard real-time QoS profiles; aggregator or bridging for large data streams
Multi-robot scalability	Domain-partitioning, hierarchical scheduling; integrated GPU/FPGA for fleet-level HPC
Integration of security with real-time	Selective encryption or hardware-enforced memory isolation to keep latencies low
Dynamic reconfiguration / partial reprogramming	Online scheduling for FPGA tasks, orchestrating container placement, real-time RTOS on SoC

In conclusion, real-time scheduling and hardware acceleration in ROS 2 represents a powerful suite of solutions, yielding up to *2–80*$\times$ improvements in end-to-end latency or throughput. Widespread adoption, however, remains tied to simplifying concurrency control, standardizing QoS, and addressing organizational or cost-related hurdles. By integrating these efforts under a cohesive roadmap (priority-based scheduling, hardware acceleration, micro-ROS, security integration), ROS 2 stands poised to meet the demanding performance needs of next-generation robotic platforms.

4.3. Security, Privacy, and Safety in ROS 2

4.3.1. RQ1 – Technical Foundations and Innovations

DDS-Based Security and Vulnerability Assessments: ROS 2 incorporates a DDS-based security model (SROS2) providing authentication, encryption, and access control at the transport layer. Yet, many deployments leave security partially configured, exposing nodes to replay attacks, unauthorized subscriptions, and malicious bridging [14,72–74]. Efforts to systematically analyze these vulnerabilities propose automated *attack surface* assessments, revealing how default ROS 2 settings can be insufficient for mission-critical applications [75].

 doi:10.20944/preprints202410.1204.v3

Intrusion Detection and Prevention (IDP): Intrusion detection and prevention (IDP) in ROS 2 has garnered significant attention, with research increasingly leveraging domain-specific anomaly detection techniques and ensemble machine-learning classifiers to identify malicious or erratic behaviors in near-real-time [76,77]. These advanced methods allow the system to proactively halt suspicious actuator commands or block malicious topics, thereby reinforcing the overall integrity of the robotic network. Despite detection accuracies often exceeding 99%, a notable challenge lies in the high CPU and network overhead associated with these solutions, particularly as system scale increases [76]. Ongoing research is focused on optimizing these approaches to ensure that the robust security provided does not come at the cost of significant performance degradation in large-scale ROS 2 deployments.

Blockchain and Advanced Access Control: In ROS 2, blockchain-based systems and attribute-based access control (ABAC) are emerging as promising solutions for secure multi-robot collaboration. These approaches enable the logging of command transactions and the enforcement of fine-grained, role-specific permissions, which not only enhances security but also provides a transparent audit trail for system actions [78–80]. Event-driven blockchain architectures, in particular, offer a way to reduce latency overhead by triggering consensus only when necessary, rather than relying on an always-on consensus mechanism. However, while these innovations bring robust security features to ROS 2 deployments, scaling them to large, real-time robotic networks remains challenging due to inherent performance constraints that must be addressed to maintain operational efficiency.

Formal Verification and Safety: In ROS 2, formal verification and safety are pivotal for ensuring system reliability and compliance. Researchers employ formal methods—such as timed automata and privacy-centric graph analysis—to rigorously verify run-time safety, detect collisions, and guarantee the timely delivery of messages [81,82]. To address privacy concerns, privacy-protecting "sanitizer" nodes are integrated to filter or anonymize sensitive data, aligning ROS 2 systems with stringent compliance frameworks like GDPR. However, as verification efforts scale with system complexity, the resulting state-space growth and extended verification horizons can compromise real-time responsiveness, prompting ongoing research into more efficient, modular verification techniques that balance safety with performance.

Representative Equations and Overhead Considerations: Equations (5)–(8) model aspects like attack probability, encryption overhead, security-induced latency, and detection probability across multiple intrusion detection modules:

$$P_{\text{attack}} = 1 - (1 - p)^{N_{\text{vul}}}, \tag{5}$$

$$L_{\text{enc}} = L_{\text{base}} + \kappa_{\text{enc}} \cdot M, \tag{6}$$

$$L_{\text{sec}} = \max(L_{\text{enc}}, L_{\text{auth}}) + \delta, \tag{7}$$

$$P_{\text{detect}} = 1 - \prod_{j=1}^{m} (1 - \gamma_j). \tag{8}$$

These formulations underscore the trade-offs between strong security measures (encryption, access control, intrusion detection) and real-time performance constraints (latency, CPU usage).

4.3.2. RQ2 – Industry Trends and Real-World Deployments

Real-World Use Cases and Partial Deployments: Industry adoption of security, privacy, and safety features in ROS 2 remains fragmented. Many organizations still rely on default or minimal security settings, as robust encryption or blockchain-based logging can introduce latencies on the order of hundreds of milliseconds or add significant CPU overhead [74]. Nevertheless, certain high-stakes environments (e.g., healthcare robotics, autonomous vehicles) are beginning to adopt advanced cryptographic or intrusion-prevention solutions to protect patient data or ensure collision-free navigation.

Use in Healthcare and Multi-Robot Collaboration: Healthcare robotics systems demand rigorous privacy protections for patient data while ensuring fault tolerance. Formal verification tools—such as those introduced by [81]—are applied to critical tasks (e.g., surgical assistance) to prevent hazardous

 doi:10.20944/preprints202410.1204.v3

collisions or device malfunctions. Multi-robot fleets in industrial or logistics settings (e.g., automated forklifts, warehouse robots) demonstrate early implementations of access control frameworks (blockchain-based or ABAC) to restrict commands to authorized operators [78].

Balancing Security and Real-Time Constraints: While advanced security is crucial for large-scale or safety-critical deployments, real-time system integrators often face overhead barriers. For example, selectively encrypting only mission-critical topics (rather than all topics) can substantially cut latency [75], and advanced hardware-based approaches (TPM or secure enclaves) mitigate CPU overhead. However, such solutions are not yet widespread, indicating a cautious or partial adoption trend, especially among companies with legacy ROS 1 architectures or limited security expertise [73].

4.3.3. RQ3 – Adoption Challenges and Future Solutions

Technical Barriers and Organizational Hurdles: Default ROS 2 configurations do not adequately secure large fleets, and the overhead of advanced security or blockchain frameworks remains high for many real-time workloads [72]. The complexity of configuring role- or attribute-based access control, coupled with dynamic policy updates for multi-robot systems, frequently exceeds the skillsets of typical robotics teams. Meanwhile, formal verification remains computationally heavy once system topologies scale [82].

Potential Solutions and Roadmap: Despite these obstacles, multiple paths forward exist. One promising solution is to implement selective or context-aware security measures, focusing on encrypting only the most sensitive topics and leveraging hardware accelerators such as TPM or HSM to reduce CPU overhead [82]. In addition, large fleets demand robust, user-friendly tools for automated PKI and certificate management, facilitating key distribution, credential rotation, and the maintenance of cryptographic freshness [78]. The development of evolving privacy graphs also offers a dynamic approach to tracking data flows as nodes change, ensuring compliance across fluctuating topologies. Finally, integrating intrusion detection tools—either by embedding IDP modules directly within ROS 2 distributions or via simple plugins—can help lower the barrier to real-time anomaly detection [77].

Looking Ahead: Table 7 and Table 6 summarize the key contributions, results, and open challenges. Future efforts to combine secure-by-default DDS configurations with lightweight intrusion detection, partial encryption, and formal safety checks may foster wider industry adoption. Nonetheless, bridging the gap between robust security guarantees and strict real-time performance remains an active research frontier—one that will likely require closer collaboration between robotics developers, security experts, and hardware vendors to achieve safe and privacy-compliant ROS 2 systems at scale.

Table 6. Challenges and Future Directions.

Challenges / Limitations	Potential Directions
Scalability for Large Fleets	- Hierarchical or partial encryption - Automated PKI/cert management
Maintaining Real-Time Guarantees	- Selective crypto or context-aware verification - Hardware accelerators (TPM, HSM)
Dynamic Policy & Intrusion Handling	- On-the-fly access updates - ML-based anomaly detection with minimal overhead
Privacy Graph Evolution	- Real-time sanitization for changing nodes - External data flow integration

 doi:10.20944/preprints202410.1204.v3

Table 7. Key Contributions and Results.

Contributions	Results
Blockchain-based or ABAC-based Access Control	- Near real-time command logging (latencies ≈ 0.5–1.0 s) - Flexible policy resolution for multi-user multi-robot
Intrusion Detection/Prevention	- ML-based ensembles achieve >99% detection - Real-time safety actions (e.g. halting actuators)
Formal Safety Verification & Privacy Tools	- Ensured collision-free navigation with fallback controllers - Privacy graph blocked data leaks, enabling GDPR compliance
DDS Security / Vulnerability Analysis	- 10–150% latency overhead from encryption - System remains exposed if security is not properly configured

4.4. Multi-Robot and Distributed ROS 2 Systems

Table 8. Proposed Taxonomy for Multi-Robot and Distributed ROS 2 Systems.

Category	Papers
(A) Co-simulation and Hybrid Bridging Architectures	[83–86]
(B) Real-Time and Time-Synchronized Control & Access Control	[57,79,87]
(C) Edge–Cloud Offloading, Resource Aggregation, and IoT Integration	[59,88–92]
(D) Multi-Robot Communication, QoS, and Middleware (RMW) Performance	[93–96]

4.4.1. RQ1 – Technical Foundations and Innovations

Masterless Architecture and Decentralized Discovery: In ROS 2, a significant departure from the ROS 1 master-based architecture is its embrace of a masterless design rooted in DDS, which enables decentralized discovery and flexible QoS configurations. This shift eliminates the single point of failure inherent in ROS 1, thereby enhancing fault tolerance and making the system more resilient in multi-robot or large-scale distributed deployments [95]. Moreover, the masterless model allows nodes to dynamically join or leave the network without disrupting overall operations, which is particularly advantageous in unpredictable or variable network environments. However, while this decentralized approach offers greater scalability and robustness, it also introduces challenges related to managing network resources and maintaining consistent communication performance, especially under high-latency conditions or in lossy networks [96].

Bridging, Co-Simulation, and Real-Time Coordination: Bridging, co-simulation, and real-time coordination are crucial components in facilitating seamless integration and operation across diverse robotic systems in ROS 2. Bridging solutions enable communication between ROS 1 and ROS 2 nodes—or even across different DDS/IoT layers—allowing for incremental migration and effective multi-robot coordination [84,86]. Co-simulation frameworks enhance this integration by combining physical system models with bridging logic, which helps manage challenges like system latencies, packet losses, and real-time QoS requirements [85]. Moreover, many researchers are incorporating time-synchronized or velocity-aware bridging techniques to further reduce end-to-end latency, a critical factor in ensuring smooth and coordinated operation within distributed robotic swarms [87].

Edge–Cloud Offloading and Specialized RMWs: In ROS 2, edge–cloud offloading and specialized RMWs serve as critical strategies for managing the computational and communication demands of distributed robotic systems. Offloading compute-intensive tasks, such as sensor fusion or large map building, to the edge or cloud can reduce a robot's CPU usage by up to 40% [59], thereby freeing local resources for real-time decision-making. However, this efficiency gain comes at the expense of increased network latency, which must be carefully managed to avoid compromising system responsiveness. Meanwhile, specialized RMW implementations like Zenoh are designed to perform

 doi:10.20944/preprints202410.1204.v3

well under challenging network conditions, such as in lossy or mobile environments, significantly enhancing communication reliability and overall performance in dynamic multi-robot settings [93].

Equations and Algorithms: Typical formulas (Eqs. (9)–(13)) model packet loss, overhead from offloading, and QoS balancing objectives. Algorithm 10 illustrates a velocity-aware adaptation that reduces latency and packet drops by dynamically tuning buffering windows.

$$\text{Packet loss probability: } L_p = 1 - \frac{\sum_i D_s^{(i)}}{\sum_i D_p^{(i)}}, \tag{9}$$

$$\text{Window adaptation: } w_a = w + \frac{|v_p - v_s|}{1000}, \tag{10}$$

$$\text{Offloading overhead: } \Lambda_{\text{offload}} = \kappa \cdot \text{DataSize} + \eta \cdot \text{SecurityCost}, \tag{11}$$

$$\text{QoS balancing objective: } \max(R - \alpha \cdot \text{Loss}(D)), \tag{12}$$

$$\text{Throughput: TP} = \frac{\sum_{m=1}^{M} \text{Packets}_m}{\sum_{m=1}^{M} \text{Time}_m}, \tag{13}$$

Algorithm 10 Velocity-Aware Sliding Window in Multi-Robot Co-simulation.

Require: Δt_{sim}, netSimQueue Q, base window w, velocities $\{v_i\}$
Initialize $t \leftarrow 0$
while simulation not finished **do**
  $t \leftarrow t + \Delta t_{\text{sim}}$
  for each pair (i, j) **do**
    $\Delta v \leftarrow |v_i - v_j|$
    $w_a \leftarrow w + \Delta v / 1000$ ▷ Eq. (10)
    Update netSimQueue Q with w_a
    Evaluate L_p, latency, throughput
  end for
end while

4.4.2. RQ2 – Industry Trends and Real-World Deployments

Use in Large-Scale Industrial Fleets: Multiple industries, such as logistics, automotive manufacturing, and warehouse operations, integrate ROS 2 to orchestrate hundreds of robots or vehicles. *Hybrid bridging* (ROS 1–ROS 2) remains common where legacy ROS 1 nodes must co-exist with new DDS-based systems [57]. Edge or fog nodes in production lines aggregate real-time sensor data, offload compute workloads, and apply QoS balancing to maintain predictable cycle times [88,92].

Autonomous Vehicle and UAV Swarms: In autonomous vehicles, robust multi-sensor fusion demands high-throughput communication with minimal packet loss. Zenoh-based RMW solutions can outperform standard DDS in intermittent or wireless networks, such as UAV swarms operating over Wi-Fi or cellular links [94]. Case studies suggest that carefully tuning DDS QoS (`KEEP_LAST`, `RELIABLE`) or leveraging co-simulation for mission planning can yield up to 30% lower latency [85].

Integration with IoT and Cloud Platforms: ROS 2's microcontroller-based integrations and bridging to IoT protocols (e.g., MQTT) facilitate real-time data collection in constrained devices [89,90]. Meanwhile, commercial cloud robotics frameworks (e.g., AWS RoboMaker, Microsoft Azure) use containerization and domain-partitioning approaches to host large multi-robot simulations or orchestrate distributed computational pipelines [59,91].

4.4.3. RQ3 – Adoption Challenges and Future Solutions

Technical Barriers: Despite demonstrable gains in scalability and resilience, several technical barriers impede full-fledged adoption. One significant challenge is discovery overhead; in large fleets of 50 or more robots, decentralized DDS discovery can become slow or inconsistent in high-latency networks

 doi:10.20944/preprints202410.1204.v3

[95]. Additionally, complex QoS tuning poses a substantial hurdle. Advanced configurations required for network scheduling, velocity-aware bridging, and adaptive buffers are not well-documented for industrial contexts, making it difficult to optimize performance. Moreover, the coexistence of heterogeneous hardware—such as microcontrollers, GPUs, and specialized SoCs—introduces bridging overhead and concurrency issues, further complicating network management [96].

Organizational and Standardization Gaps: Large-scale or cross-facility robotics programs often lack domain-specific guidelines for multi-domain discovery, hierarchical scheduling, or bridging best practices. Additionally, new RMW solutions like Zenoh are not yet officially standardized, causing reluctance among risk-averse enterprises [93].

Proposed Solutions and Roadmap: Table 9 outlines key directions for addressing these barriers. One promising approach is the implementation of segmented DDS domains or hierarchical discovery mechanisms to effectively scale operations beyond 50 robots while reducing communication overhead. In parallel, developing adaptive QoS strategies for non-stationary networks—through on-the-fly reconfiguration or machine learning-based link-quality prediction—could further enhance network robustness. Moreover, the formal standardization of alternative RMWs, such as Zenoh, is essential for achieving robust adoption in heterogeneous systems. Finally, improving bridging efficiency for legacy and edge devices by leveraging lightweight bridging protocols or generating code that spans ROS,1 and ROS,2 stacks is critical to ensuring seamless integration across diverse hardware platforms.

Table 9. Challenges and Future Directions in Multi-Robot/Distributed ROS 2 Systems.

Challenges	Potential Future Directions
Scaling beyond 50+ robots	- Hierarchical or segmented DDS domains - More efficient discovery logic
Adaptive QoS for non-stationary networks	- On-the-fly reconfiguration - ML-based network condition prediction
Heterogeneous hardware and bridging overhead	- Lightweight bridging for microcontrollers - Joint code generation for ROS 1 & ROS 2
Formal standardization of new RMWs	- Hybrid DDS + Zenoh integration - Unified APIs for edge, cloud, and IoT

Concluding Remarks: In summary, multi-robot and distributed ROS 2 systems can achieve significant gains in scalability, performance, and fault tolerance when carefully tuned. Edge–cloud offloading, velocity-aware bridging, and advanced QoS strategies each contribute to reducing latency and packet loss. Nonetheless, complexities in discovery, QoS tuning, and standardization remain obstacles for wider industrial deployment. Ongoing research on new RMW features (e.g., Zenoh) and next-generation capabilities (e.g., multi-domain discovery, hierarchical scheduling) is expected to shape the future of robust large-scale multi-robot networks in ROS 2.

4.5. Final Remarks

In summary, our exploration of ROS 2's technical innovations and real-world deployments has addressed the core research questions driving this survey. Regarding **RQ1**, we have demonstrated how ROS 2's enhanced middleware—including DDS-based communication, zero-copy techniques, and formal verification methods—provides significant improvements in performance, real-time execution, and security compared to ROS 1, while also highlighting persistent limitations in scalability and complexity.

For **RQ2**, this section has mapped emerging trends and industrial deployments across diverse domains—from autonomous vehicles and industrial automation to healthcare and multi-robot systems—illustrating both the transformative potential and the incremental nature of ROS 2 adoption.

Finally, concerning **RQ3**, we identified key technical, organizational, and standardization challenges that hinder rapid, widespread uptake, and proposed potential solutions such as standardized QoS profiles, integrated verification tooling, and streamlined bridging strategies.

 doi:10.20944/preprints202410.1204.v3

Collectively, these discussions underscore the multifaceted evolution of ROS 2, setting the stage for further innovations and practical tool development.

5. ROS 2 in Practice: Industry Applications, Trends, and Adoption Challenges

Figure 9. Taxonomy of different fields that utilizes ROS 2.

This section examines the practical deployment of ROS 2 across diverse industrial domains, focusing primarily on two research questions: (RQ2) What are the key trends and successful use cases of ROS 2 in real-world applications? and (RQ3) What technical, industrial, and organizational challenges hinder its broader adoption, and how can they be addressed? We review how ROS 2's advanced features—such as its DDS-based, modular architecture, real-time QoS capabilities, and support for multi-robot systems—enable robust performance in fields like agriculture [97,98], consumer robotics [99], healthcare [100,101], and autonomous vehicles [58,102]. At the same time, we highlight domain-specific challenges, including sim-to-real transfer, stringent safety certifications, and interoperability with legacy systems [35,103]. By synthesizing these insights, the section outlines both the successes and the obstacles that will shape ROS 2's future evolution.

 doi:10.20944/preprints202410.1204.v3

5.1. Agriculture

Industry Trends and Successful Deployments:

In agriculture, ROS 2 leverages DDS-based communication to provide robust, decentralized control under remote or bandwidth-limited conditions [97]. Real-time capabilities are critical for automated tasks such as weed removal, pesticide spraying, and soil analysis. Several projects demonstrate extensive sensor fusion—integrating LiDAR, vision, GNSS, and IMUs—into ROS 2 node-based pipelines [98,104]. Multi-robot fleets have been deployed to enhance redundancy and coverage in large orchards and greenhouses [105].

Adoption Challenges and Future Solutions:

Challenges in agriculture include mapping and perception issues due to occlusions in dense vegetation [103]. Additionally, handling delicate produce requires specialized soft manipulators and advanced end-effectors [106,107]. Bridging the sim-to-real gap remains critical, as current simulations do not fully capture the complexities of real-world fields [108]. Future research must focus on domain-specific SLAM, vegetation-aware costmaps, and fault-tolerant multi-robot management to overcome these obstacles.

5.2. Consumer & Domestic Robotics

Industry Trends and Successful Deployments:

In consumer and domestic robotics, ROS 2 is widely employed for tasks ranging from cleaning to human–robot collaboration. Systems often leverage Nav2 or MoveIt for navigation and manipulation [99,109]. The use of micro-ROS enables cost-effective deployment on home devices [110], while enhanced security features help protect user data [82]. Social and humanoid robots, supported by LLM-based explanations and robust gesture/voice interaction modules, demonstrate advanced human-centric functionalities [111,112].

Adoption Challenges and Future Solutions:

Key challenges include ensuring human safety and building user trust. Transparent robot behaviors and quick responses are crucial, yet achieving these consistently remains difficult. Future directions emphasize improved edge–cloud AI, advanced user interfaces, and LLM-based autonomy that can adapt to dynamic environments, thereby enhancing safety and user acceptance.

5.3. Entertainment & Recreational Robotics

Industry Trends and Successful Deployments:

In entertainment robotics, ROS 2 enables multi-robot choreographies such as “swarmalator” light shows and interactive museum exhibits [113]. High-fidelity datasets support high-frequency autonomous motion in racing drones and robotic competitions [114]. Additionally, ROS 2’s distributed architecture powers museum cicerone robots that navigate crowds and deliver speech-based interactions [115]. Integration with AR/VR further expands the platform’s capabilities, combining advanced sensing (e.g., PointIt [116]) with cloud-based scene understanding.

Adoption Challenges and Future Solutions:

Real-time synchronization and low latency remain significant hurdles for large-scale multi-robot shows. Future improvements in ROS 2 tracing [117] and high-performance computing (HPC) hardware acceleration are expected to enable tighter synchronization and more immersive interactive performances.

5.4. Healthcare

Industry Trends and Successful Deployments:

Healthcare applications of ROS 2 include assistive and rehabilitation robotics, where real-time control is critical. For example, knee rehabilitation robots have demonstrated safe, sub-millisecond

 doi:10.20944/preprints202410.1204.v3

control loops [100]. In exoskeleton research, motion capture systems (MOCAPs) are employed to track movements in real time, although standard support remains limited [118]. Autonomous endovascular systems illustrate the need for integrated, modular testbeds that combine tracking, control, and instrument manipulation [101].

Adoption Challenges and Future Solutions:

Challenges in healthcare include rigorous regulatory requirements (e.g., FDA, CE marking) and closing the sim-to-real gap for surgical guidance systems. Future efforts should focus on achieving medical-grade reliability, advanced AR-based surgical guidance, and standardized certification processes to facilitate broader adoption in clinical settings.

5.5. Military & Defense

Industry Trends and Successful Deployments:

ROS 2's inherent security features, such as SROS2, and real-time node scheduling make it suitable for defense applications. Resilient multi-robot networks are employed in contested environments, particularly in UAV swarms, to maintain robust sensor fusion even under partial network failures [26,95]. Blockchain-based event-driven systems have been explored to ensure tamper-proof logging [78].

Adoption Challenges and Future Solutions:

Despite these advances, defense applications face challenges in integrating domain-specific protocols and achieving certification (e.g., DO-178C). Future directions include developing robust run-time monitoring, mission-critical scheduling, and secure multi-agent autonomy tailored to defense requirements.

5.6. Logistics & Warehousing

Industry Trends and Successful Deployments:

In logistics, ROS 2 supports autonomous mobile robots (AMRs) and collaborative robotic arms by leveraging its topic-based architecture and real-time QoS [119]. Large fleets require advanced traffic management systems and conflict resolution strategies, with some teams employing FogROS2 to offload heavy computations [120]. Integration with industrial controllers (PLCs) further enhances multi-sensor data fusion and real-time planning in dynamic warehouse environments [121].

Adoption Challenges and Future Solutions:

Logistics applications must overcome challenges in real-time scheduling for global path planning and decentralized multi-robot orchestration. Future improvements should target enhanced safety in human–robot collaboration and standardized multi-robot communication protocols to streamline integration across diverse industrial systems.

5.7. Aerospace & Marine

Industry Trends and Successful Deployments:

ROS 2 is utilized in space missions and marine robotics, where multi-robot coordination is critical. For instance, ROS 2 supports multi-robot lunar exploration and UAV swarms for space applications [97,122]. In marine robotics, advanced model predictive control is used to manage disturbances from waves and currents [123]. Specialized QoS parameters help maintain data integrity in bandwidth-limited or disconnected networks [96].

Adoption Challenges and Future Solutions:

Key challenges include ensuring robust discoverability in disconnected environments and managing resource limitations on space-qualified hardware. Future work should focus on refining real-time

 doi:10.20944/preprints202410.1204.v3

HPC on resource-constrained boards, securing data channels in extreme conditions, and enhancing SLAM and hazard detection for unstructured planetary terrains.

5.8. Public Safety & Disaster Response

Industry Trends and Successful Deployments:

In disaster response, ROS 2 is used to coordinate multi-robot systems for search and rescue, leveraging advanced sensor fusion and real-time autonomy [71]. Drone forensics and digital twin technologies facilitate post-incident analysis, while integrated sensor networks aid in survivor detection under challenging conditions [124,125]. Security monitoring through blockchain or intrusion detection further supports mission data integrity [76,78].

Adoption Challenges and Future Solutions:

For public safety, real-time HPC on portable hardware and reliable communication under contested conditions remain challenging. Future research should integrate edge AI, robust fault tolerance, and secure communication protocols to improve performance in disaster response scenarios.

5.9. Transportation (Self-driving Cars)

Industry Trends and Successful Deployments:

ROS 2's modular architecture has been adopted in autonomous driving for cooperative sensor sharing and real-time control [102,126]. Containerized deployments improve maintenance and stability, while integration with Adaptive AUTOSAR and automotive-grade HPC platforms supports stringent safety standards [58,127]. Advanced neural networks and simulators (CARLA, LGSVL) are used for scenario-based testing and enhanced perception under challenging conditions [128].

Adoption Challenges and Future Solutions:

However, achieving full ISO 26262 compliance in ROS 2 remains a hurdle, and bridging open-source algorithms with automotive-grade standards is challenging. Future efforts should target enhanced safety certification, optimized real-time processing under automotive conditions, and tighter integration with commercial automotive platforms.

5.10. Overall Reflection and Future Directions

Across all application domains—from agriculture and domestic robotics to aerospace and transportation—ROS 2's DDS-based, modular architecture significantly lowers barriers to deploying large-scale, real-time robotic systems. However, each domain faces unique challenges: mapping complexities in agriculture, user trust in domestic robotics, latency in entertainment, rigorous certification in healthcare and automotive, and secure, resilient communications in defense. Future research must focus on domain-specific QA processes, extended real-time scheduling, adaptive QoS tuning, and seamless interoperability with legacy systems and specialized standards. Continued evolution of ROS 2 packages and industry-tailored solutions will be critical in overcoming these challenges and fully realizing the potential of ROS 2 in diverse real-world applications.

This section comprehensively addresses the real-world deployment of ROS 2, directly engaging with RQ2 by showcasing successful use cases and industry trends across domains such as agriculture, consumer robotics, entertainment, healthcare, military, logistics, aerospace, public safety, and transportation. Simultaneously, the section tackles **RQ3** by identifying and discussing the technical, organizational, and domain-specific challenges that currently hinder broader ROS 2 adoption—such as sim-to-real transfer issues, stringent certification requirements, and integration with legacy systems. By synthesizing these practical insights, the section not only demonstrates ROS 2's transformative impact on diverse industrial applications but also highlights areas for future research and improvement. These discussions seamlessly pave the way for the subsequent exploration of ROS 2's key frameworks,

 doi:10.20944/preprints202410.1204.v3

libraries, and toolkits, which provide the essential software ecosystem for addressing the identified challenges and further enhancing ROS 2's real-world applicability.

6. ROS 2 Key Frameworks, Libraries, and Toolkits

As we transition from an in-depth analysis of ROS 2's technical underpinnings, deployment scenarios, and adoption challenges, the next section shifts focus to the essential frameworks, libraries, and toolkits that empower developers and researchers to harness these innovations. Building on the advanced middleware techniques, real-time scheduling, and security features discussed earlier, this section represents the practical software ecosystem surrounding ROS 2. Here, we examine foundational libraries that facilitate everything from transform management and sensor integration to simulation and visualization. This exploration not only reinforces the technical strengths outlined in **RQ1** and **RQ2** but also addresses the practical concerns raised in **RQ3** by showcasing tools that simplify configuration, enhance interoperability, and lower the barrier to entry for both academic and industrial users.

6.1. ROS 2 Compatible Simulators

Simulation plays a crucial role in robotics by allowing developers to test algorithms and entire software stacks in controlled, reproducible environments. Historically, *Gazebo* was the default simulator for ROS 1; it then evolved through *Ignition* (an experimental rearchitected branch) into what is now called *Gazebo Sim* as shown in Figure 10. The compatibility between ROS and Gazebo varies significantly depending on the versions in use. For example, the recommended combination is ROS 2 **Jazzy** with GZ **Harmonic**, which has been fully supported and tested, while other pairings such as ROS 2 **Humble** with GZ **Ionic** are marked as incompatible. Some configurations, like ROS 2 **Rolling** with GZ **Garden** or ROS 2 **Iron** with GZ **Harmonic**, are possible but require extra effort and careful tuning to achieve reliable performance (Full compatibility table is shown here Table 10). Ultimately, it is advisable to adhere to the supported combinations for a smoother integration and a more stable simulation experience [129].

Table 10. Summary of Compatible ROS and Gazebo Combinations. ✓ Recommended, ✗ Incompatible, † Possible but use with caution.

ROS Version	GZ Citadel (LTS)	GZ Fortress (LTS)	GZ Garden	GZ Harmonic (LTS)	GZ Ionic
ROS 2 Rolling	✗	✗	†	†	✓
ROS 2 Jazzy (LTS)	✗	✗	†	✓	✗
ROS 2 Iron	✗	✓	†	†	✗
ROS 2 Humble (LTS)	✗	✓	†	†	✗
ROS 2 Foxy (LTS)	✓	✗	✗	✗	✗

Figure 10. Gazebo simulator evolution.

Beyond Gazebo-based solutions, multiple other simulators have gained traction within the ROS 2 ecosystem. For example, **Isaac Sim** (NVIDIA) is a high-fidelity simulator leveraging PhysX and advanced rendering; its official ROS 2 bridges make it suitable for testing visually rich or physically complex tasks, although it can be resource-intensive [130]. In addition, **PyBullet** is a lightweight, Python-focused physics engine widely used in reinforcement learning research, and its community-

 doi:10.20944/preprints202410.1204.v3

driven ROS 2 bridges facilitate integration with ROS 2 topics and services, making it a preferred choice for fast prototyping or large-scale parallel simulations [131].

Similarly, **Webots** is known for its ease of use and a broad set of built-in robot models, and it provides a ROS 2 interface for sensor data and actuation commands, catering to both educational and industrial prototyping needs [132]. **CoppeliaSim** (formerly V-REP) offers versatile scene building, multiple physics engines, and built-in "proximity sensor" simulation; however, its bridges to ROS 2 might require additional customization for advanced features [133].

Additionally, **CARLA** is a high-fidelity autonomous driving simulator designed for testing self-driving algorithms, featuring realistic urban environments, vehicle dynamics, and sensor simulation, which makes it a key tool for ROS 2-based autonomous vehicle research [134]. Moreover, the ROS 2 community continues to expand support for alternative or domain-specific simulators, with some projects integrating Unity or Unreal Engine with ROS 2 to enhance perception tasks through photorealistic rendering. Figure 11 illustrates a comparative table of ROS 2 simulators, and a recent systematic review [9] comparing multiple ROS 2-compatible simulators for robotic arm manipulation (including Ignition, Webots, Isaac Sim, PyBullet, and CoppeliaSim) confirms that none uniformly outperforms all others—some excel in stability and realism over long-duration tasks, while others offer lower CPU overhead or better headless performance.

Name	Programming language	API support	Physics Engine	CAD files support	Multithreading enabled?	Family of robots	Supported actuators	ML Support	Headless Support	Open Source	Supported sensors
Gazebo	C++, Python	C++, Python	ODE Bullet Simbody DART	SDF/URDF OBJ STL COLLADA (.dae)	Yes	Supports a wide range of robots, including wheeled robots, legged robots, and drones.	Motors (position and velocity), servos, and force-torque actuators.	External	Full	Yes	Camera Sensor, Depth camera sensor, Distance and Proximity Sensors, Laser, Force Sensors
Webots	C, C++, Python, Java, MATLAB	C++, Python, ROS 2 API	ODE	WBT, VRML, X3D	Yes	Supports a wide range of robots, including humanoids, drones, wheeled robots, and custom designs.	Brake, Connector, Display, Emitter, Linear Motor, Muscle, Pen, Propeller, Rotational Motor, Speaker, Track (Conveyor Belt or tank robots)	External	Partial	Yes	Accelerometer, Camera, Compass, Distance Sensor, GPS, Gyro, Lidar, Position Sensor, Receiver, Touch Sensor
CoppeliaSim	Lua	C, C++, Python, Java, Urbi, Matlab, Octave	ODE, Bullet, Vortex, Newton	OBJ, STL, DXF, 3DS, Collada, URDF	No	Mobile, Humanoid, Industrial	Revolute (and some variants), Prismatic and Spherical Motors	External	Full	No	Force Sensor Proximity Sensor Accelerometers Gyro Lasers ...
Unity	C#	C#	NVidia PhysX, Box2D	FBX, Collada, DXF, OBJ	Yes	Mobile, Humanoid, Industrial	Revolute and Prismatic Joints	External	Full	No	Cameras
Ignition	C++, Python	C++, Python	DART, Bullet, TPE, and ODE.	STL, COLLADA (.dae), and OBJ.	Yes	Supports a wide range of robots, including wheeled robots, humanoids, drones, and more.	Motors (velocity, position, and torque control), servos, and custom actuators.	External	Full	Yes	Cameras, LIDAR, IMUs, GPS, sonar, force-torque sensors, and many other sensors through the use of plugins and sensor models.
Isaac Sim	Python, C++	Python API, C++, and ROS 2 API integration	PhysX	URDF, OBJ, FBX, USD (Universal Scene Description)	Yes	Focuses on industrial robots, mobile robots, manipulators, drones, and custom robot designs.	Motors, servos, position, velocity, and force control actuators.	Integrated	Full	No	Cameras, LIDAR, IMUs, GPS, force-torque sensors, and many others through the extensive sensor model support in Isaac Sim.
PyBullet	Python, C++	Python API (primarily), C++	Bullet	URDF, SDF, STL and OBJ.	No	Supports a variety of robots, especially manipulators and mobile robots, often used for robotics research and reinforcement learning.	Motors (position, velocity, and torque control), force actuators, and custom actuators.	External	Full	Yes	Cameras, LIDAR, IMUs, force sensors, and custom sensor implementations through scripting.
Mujoco	C, Python (via bindings)	C API, Python API (via bindings like mujoco-py)	Mujoco	URDF and MJCF, OBJ and STL	No	Focuses on manipulators, humanoids, legged robots, and custom robots, widely used in research on robotics and control.	Motors (position, velocity, and torque control), muscle models, and force/torque actuators.	External	Full	Yes	IMUs, force-torque sensors, touch sensors, cameras, and various custom sensors through scripting and configuration files.

Figure 11. ROS 2 Simulators.

6.2. ROS 2 Frameworks

ROS 2 builds on the strong legacy of ROS 1 by introducing robust frameworks for motion planning, navigation, and industrial applications. Key meta-libraries such as **MoveIt 2** [135,136], **ROS Industrial** [137], and **Micro-ROS** [138,139] demonstrate the ecosystem's versatility. **MoveIt 2** provides advanced functionalities like inverse kinematics, collision checking, and path optimization [140], and integrates seamlessly with 3D sensors and control interfaces for complex tasks [141] for tasks such as pick-and-place [141]. In contrast, **ROS Industrial** targets the unique demands of industrial robot arms, offering specialized drivers, calibration tools, and protocol bridging. Meanwhile, **Micro-ROS** extends ROS 2 communication to resource-constrained devices, enabling real-time sensor integration and control on microcontrollers [142]. A comprehensive overview of these meta-libraries is provided in Table 11.

Complementing these frameworks are solutions like Navigation2 (**Nav2**) [11], **RViz2** [143], and **SpaceROS** [144], each contributing distinct capabilities to the ROS 2 ecosystem. **Navigation2** serves as the flagship mobile robot navigation stack, combining global and local path planning with innovative strategies such as the Smac Planner [145]. **RViz2** offers extensive 3D visualization support, facilitating

 doi:10.20944/preprints202410.1204.v3

real-time debugging and teleoperation across various frameworks. Additionally, the experimental **SpaceROS** project illustrates ROS 2's expanding reach into aerospace applications, emphasizing reliability and adaptation for radiation-hardened hardware [146,147]. Together, these frameworks and tools form a comprehensive toolkit that supports a wide range of robotics applications, as summarized in Table 11.

Table 11. ROS 2 Meta-Libraries and Their Applications.

ROS 2 Meta-Libraries	Application
MoveIt 2	Motion planning for manipulators and humanoid robots
Navigation2 (Nav2)	Autonomous navigation for wheeled and legged robots
ros_control	Hardware abstraction, low-level control, and hardware interfaces for robots
Micro-ROS	ROS 2 for microcontrollers, used in embedded systems and resource-constrained robots
MAVROS	Communication interface between ROS 2 and MAVLink-based drones (e.g., multi-rotor UAVs)
SROS2 (Secure ROS 2)	Security features like authentication, encryption, and access control for ROS 2 systems
SMACC	State machine framework for robot decision-making in ROS 2
Orocos	Real-time control framework for complex robot control tasks, integrated with ROS 2

6.3. ROS 2 Libraries

ROS 2's functionality extends beyond its extensive "toolkit" packages to include several core libraries that underpin essential capabilities such as 3D perception, transformation handling, and behavior orchestration. For instance, **PCL** (Point Cloud Library) [148] is frequently employed for advanced 3D perception, object recognition, and environment reconstruction, with nodes and plugins seamlessly integrating with ROS 2 `PointCloud2` messages. Similarly, **TF2** (Transform Library) [149] is indispensable for managing 3D coordinate transforms among various robot links and sensors, ensuring consistent referencing of frames crucial for tasks like sensor fusion, navigation, and manipulation.

In addition, BehaviorTree.CPP (**Behavior Trees**) [150] provides a flexible framework for high-level task execution in ROS 2, enabling stacks such as Nav2 and manipulation frameworks to orchestrate complex sequences like path planning, base movement, recovery, and gripper operations. This modular approach enhances readability and reusability by allowing developers to replace or refine sub-trees without rewriting an entire state machine [151]. Table 12 outlines a selection of these ROS 2 core libraries along with their primary applications, demonstrating how they collectively support the robust functionality required in modern robotic systems.

 doi:10.20944/preprints202410.1204.v3

Table 12. ROS 2 Meta-Libraries and Their Applications.

ROS 2 Core-Libraries	Application
Perception (PCL, OpenCV)	3D perception, object recognition, and environment mapping
TF2 (Transform Library)	Coordinate transformations between multiple frames for robots
BehaviorTree.CPP	Task and decision-making frameworks for autonomous robots
ROS2 Bag	Data recording and playback for debugging and analysis
Fast-RTPS	Real-time communication middleware for distributed robotic systems (DDS implementation)
Lifecycle	Lifecycle management for ROS 2 nodes, handling transitions between states
PlotJuggler	Real-time data visualization and plotting tool for ROS 2
DDS-Security	Security plugins for encryption, authentication, and access control in DDS
Rosbridge	JSON API for ROS 2, enabling interaction with web interfaces and mobile devices
RViz2 Plugins	Custom plugins for extended visualization of data in RViz2

6.4. ROS 2 Open Source Packages

Beyond the officially maintained libraries, the ROS 2 community has cultivated a robust ecosystem of specialized packages that extend the framework's capabilities into diverse domains. **MAVROS** [152] serves as a crucial communication bridge between ROS 2 and MAVLink-based drones, enabling the control of multi-rotor UAVs or fixed-wing platforms by translating standard ROS topics and services into MAVLink commands. Similarly, **SMACC** [153] offers a state machine-based approach for complex robot decision-making, complete with introspection tools and code-generation features that support hierarchical concurrency and dynamic, run-time reconfiguration of behaviors.

In addition, open-source autonomous driving frameworks such as **Autoware.Auto** [154] and **CyberRT** [155] provide comprehensive solutions for perception, planning, and vehicle control, incorporating reference architectures for LiDAR-based localization, object detection, and route planning. Complementing these are community-driven extensions for node lifecycle management [156], hardware interface abstraction via **ros_control** [157], and specialized visualization tools in RViz2, with many robotics labs contributing domain-specific nodes spanning underwater, agricultural, and swarm robotics applications [158]. Collectively, these packages underscore the vibrant and expansive nature of the ROS 2 ecosystem, enabling tailored solutions for a wide range of robotic applications.

Table 13 provides a comprehensive overview of the diverse ROS 2 open source packages developed by the community, organized by key application domains. The table categorizes packages into areas such as Multi Robotic Systems, Cooperative Robotics & HRI, Simulators, Computer Vision, Reinforcement Learning, Performance Evaluation, Real-Time, Cyber Security, Software Platforms, State Estimation & Prediction, Planning, Navigation, and Embedded & Distributed Systems. Additionally, it highlights support for various robotic platforms including UAVs, UUVs, self-driving cars, service robots, and product integration. Each category lists representative packages along with their corresponding citations—for instance, Aerostack2 [159] for multi-robot systems, Opendr [160,161] for cooperative robotics, and LGSVL Simulator [162] for simulators—illustrating the extensive scope and active development within the ROS 2 ecosystem. This curated listing underscores the dynamic nature of ROS 2 community contributions and their impact on a broad range of robotics applications.

 doi:10.20944/preprints202410.1204.v3

Table 13. ROS 2 Open Source Packages.

Category	Citation
Multi Robotic Systems	Aerostack2[159], CrazyChoir[163], KubeROS[164], ROS2SWARM[165], The Cambridge RoboMaster[166], Toychain[167], TestbedROS2Swarm[168], ChoiRbot[169], ROS2BDI[170],
Cooperative Robotics & HRI	ChoiRbot[169], ROSGPT[171], opendr[160,161], NAO[172], qml_ros2_plugin[173], PointIt[174], ros2-foxy-wearable-biosensors[175]
Simulators	MVSim[176], HuNavSim[177], LGSVL Simulator[162], LunarSim[178], MAES[179], UUV simulator[180]
Computer Vision	HawkDrive[128], ROSGPT_Vision[181], GLIM[182], UAV Volcanic Plume Sampling[183], YOLOX[184], direct_visual_lidar_calibration[185], Bridging 3D Slicer and ROS2[186], Video Encoding and Decoding for High-Definition Datasets[187],
Reinforcement Learning	ros2-forest[188], gym-gazebo2[189], drl_grasping[190], LPAC[191], opendr[160, 161], ros2learn[192], An Educational Kit for Simulated Robot Learning in ROS 2[193,194]
Performance Evaluation	ChoiRbot[169,195], FogROS2[120], DriveEnv-NeRF[196], RobotPerf[197]
Real-Time	CARET[195,198], ros2_tracing[117]
Cyber Security	Bobble-Bot[199], Hyperledger Fabric Blockchain[200], rvd[201], KISS-ICP[202], RCTF[203], SROS2[204],
Software Platforms	Aztarna[205], CFV2[206], SkiROS2[207], SMARTmBOT[208], Space ROS[146], ros2-3gppSA6-mapper[209]
State Estimation & Prediction	MixNet[210], FusionTracking[211], NanoMap[212], wayp[213], lidar_cluster_ros2[127], VECTOR[214], SMARTTRACK[215]
Planning	navigation2[216], PlanSys2[217], SAILOR[218], YASMIN[219]
Navigation	mola[220], depth_nav_tools[221], nav2_accountability_explainability[222], Navigation Approach based on Bird's-Eye View[223], DeRO[224], vox_nav[225, 226], evo[227], FlexMap Fusion[228], Mobile MoCap[229], MOCAP4ROS2[230], Multi-Robot-Graph-SLAM[231], pointcloudset[232], flexible_navigation[233], The Marathon 2[19],
Embedded & Distributed Systems	embeddedRTPS[234], forest[235], ReconROS[236,237], FogROS[238], FogROS2[239–241], ros2-message-flow-analysis[93], PAAM[65], RobotCore[242]
Robotic Platforms Support	
UAV	anafi_ros[243], CrazyChoir[163], UAV Volcanic Plume Sampling[183], HyperDog[244], Aerostack2[159], MPSoC4Drones[245], uav_dataset_generation_ros[246]
UUV	SUAVE[247], Angler[248], UUV simulator[180]
Self-driving Cars	Autoware_Perf[249], DriveEnv-NeRF[196], XTENTH-CAR[250]
Service Robots	MERLIN & MERLIN2[251,252],
Product Integration	libiiwa[253], HRIM[254], kmriiwa[255], LBR-Stack[256], MeROS[257], OtterROS[258], RCLAda[259], RoboFuzz[260], Wrapyfi[194]

 doi:10.20944/preprints202410.1204.v3

6.5. Datasets for/from ROS 2

Datasets are fundamental to data-driven robotics, driving advances in SLAM, perception, manipulation, and autonomy. Recently, several groups have released ROS 2-compatible datasets or used ROS 2 to generate and annotate sensor recordings. For example, [261] provide a dataset designed to benchmark modern visual SLAM approaches in ROS 2 environments, offering curated sensor logs for evaluating localization accuracy. In the automotive domain, the dataset from [126] explores vehicle behavior analytics by detecting autonomous driving patterns from robot motion data. Domain-specific datasets for night-driving perception, integrated with advanced vision-language models, are presented in [128,262].

In simulated settings, [190] introduce an environment for grasping tasks on uneven lunar terrain, capturing 3D octree observations via ROS 2 topics. Agricultural datasets documented in [263] record challenging greenhouse environments using multi-modal sensors, often logged via micro-ROS or embedded ROS 2 pipelines. Datasets emphasizing UAV high-speed flight or collaborative sensor fusion in swarm scenarios are detailed in [114,264]. Additionally, a dataset for more than 5000 UAV trajectories is detailed in [214]. For cybersecurity research, the "ROSPaCe" dataset from [265] captures network traffic and system-level anomalies in ROS 2 setups. Additionally, human activity datasets [266,267] and urban traffic datasets (e.g., CARLA images with semantic segmentation [268]) further illustrate ROS 2's versatility. Many of these datasets are recorded with ROS 2 logging tools (e.g., `rosbag2`), ensuring reproducibility and seamless integration with standard `sensor_msgs` or `nav_msgs` formats.

Summary

this section not only delineates the technical underpinnings that distinguish ROS•2 from ROS•1—addressing the technical enhancements and limitations highlighted in **RQ1**—but also maps out the diverse and practical applications across industry sectors as outlined in **RQ2**. The discussion of simulators, motion-planning frameworks, and core libraries demonstrates how these components facilitate robust, real-world deployments, while the review of community packages underscores ongoing efforts to overcome challenges such as middleware complexity and migration hurdles (**RQ3**). By providing a detailed landscape of both the technical and practical assets within the ROS•2 ecosystem, this section lays a strong foundation for the next part of the survey: the *ROS Database*. This online platform builds on the insights gathered here by offering dynamic visualizations and interactive filtering tools that enable users to explore trends, challenges, and innovative solutions in ROS•2 adoption across various fields.

7. ROS Database

As part of this comprehensive survey on ROS 2, we have developed a dedicated *online database* to centralize and analyze the growing body of ROS-related literature (Figure 12). This platform goes beyond merely listing references—it enables rich **user interaction** by supporting multi-level filtering, browsing, exporting, and community submission of both articles and frameworks. By offering dynamic visualizations and advanced search capabilities, the database allows users to identify key trends, understand industry-specific challenges, and explore innovative solutions in the adoption of ROS 2 across diverse fields such as autonomous vehicles, healthcare, agriculture, logistics, and industrial automation. In this section, we detail not only the data we have collected, but also *how users can interact* with the database's features to address these real-world application challenges outlined in RQ2.

 doi:10.20944/preprints202410.1204.v3

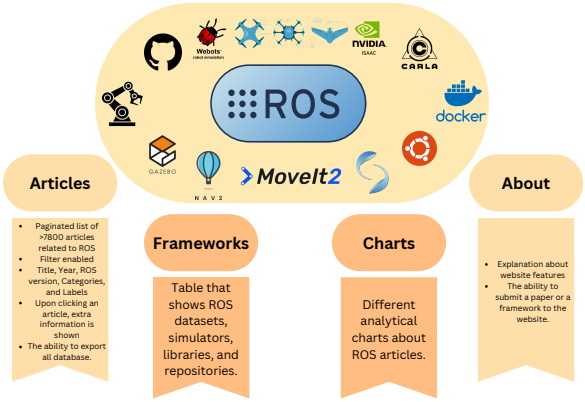

Figure 12. High-level layout of our ROS Database website.

1) **Articles Tab.**

The database currently indexes **7,371** ROS-related articles, covering both ROS 1 and ROS 2. Users are presented with a paginated table where each entry displays the *title*, *year of publication*, and *ROS version*, along with detailed categorization that includes the *category*, *subcategory*, *sub-subcategory*, and various *domain-specific labels* such as *UAV*, *dataset*, *self-driving car*, or *UUV*. Additional metadata—like the *abstract*, *DOI* link, and *GitHub* repository when available—can be accessed on demand. A robust filtering and searching mechanism allows users to sort articles by year, category, platform type (e.g., UAV, UGV), or ROS version, facilitating the quick isolation of specific subsets, such as articles on ROS 2 combined with UAV swarms from 2022. For deeper insights, clicking an entry expands it to reveal key contributions, direct links to the paper, and any recorded notes regarding the experiments. Moreover, the **export** functionality enables users to download either the entire table or a filtered selection in CSV format, making it an excellent tool for meta-analyses or offline reference management.

2) **Frameworks Tab.**

Another core feature is the aggregated Frameworks Tab, which compiles information on popular ROS frameworks, simulators, and libraries, as outlined in the schematic in Figure 12. This table encompasses key ROS libraries—such as `Nav2`, `MoveIt,2`, `Rviz2`, `PCL`, and micro-ROS—as well as widely used simulators like Gazebo, Isaac Sim, Webots, and CARLA. It also features community packages from GitHub and curated ROS datasets, including examples like *GREENBOT* [263], *HawkDrive* [128], and *Swarm-SLAM* [269]. Each entry highlights the framework's name, a brief description, its main domain focus, the corresponding GitHub link, and any relevant publications from our Articles Tab, providing users with a comprehensive overview of the ROS ecosystem.

3) **Charts Tab.**

The Charts Tab emphasizes the importance of visual analytics in understanding ROS research trends. It features a scatter plot that tracks publication counts over time, highlighting the yearly growth and the comparative usage between ROS 1 and ROS 2—as shown in Figure 14. Additionally, a sunburst plot provides a detailed illustration of how ROS research is distributed across various subdomains, such as Real-Time systems, Multi-Robot configurations, and Planning, mirroring the insights in Figure 13. Complementing these, a bar chart outlines the different types of article contributions, offering users a comprehensive visual summary of the field's landscape.

 doi:10.20944/preprints202410.1204.v3

Figure 13. ROS 2 Research Domains.

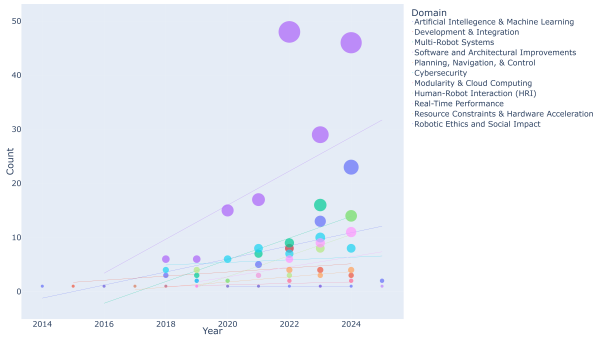

Figure 14. Number of Articles vs Year (ROS1 vs ROS2).

4) **About Tab.**

Finally, the "About" section provides a comprehensive overview of the website's features and additional resources. It offers a short explanation of functionalities such as article pagination, advanced filtering, interpretation of the frameworks table, and the usage of the various charts. Additionally, it details collaboration information regarding the survey's creation, outlining the data-collection pipeline and the team behind the project. A user-friendly **submission portal** is also available, enabling anyone

 doi:10.20944/preprints202410.1204.v3

to propose a new paper or framework by submitting metadata like title, year, relevant domain, and GitHub link, which is then reviewed by the curation team. Finally, the section includes explanatory plots that clarify the labels used in the first tab, as shown in Figure 15.

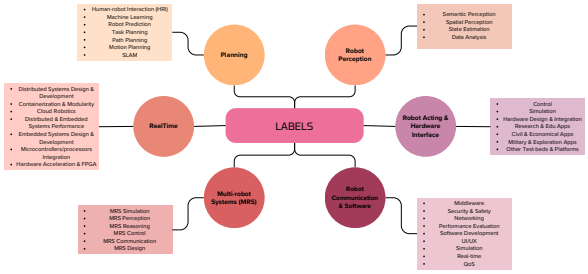

Figure 15. Different Labels that are tagged based on the research domain.

Data Integrity and Growth.

With each new ROS 2 publication, we frequently update the database. Our curation pipeline scans major indexing services (*IEEE Xplore, arXiv, MDPI, SpringerLink, etc.*) for ROS-related papers, cross-checks with user submissions, and applies manual labeling. Thus, the database remains *evolving*—currently surpassing 7,371 entries (including both ROS 1 and ROS 2 works).

In summary, the online ROS database complements our survey by *facilitating direct engagement with the literature*. Figure 12 and the concept figures provided earlier illustrate how articles, frameworks, and analytics are interconnected for a seamless exploration of the ROS ecosystem.

8. Conclusion

This survey set out to answer three key research questions (RQ1–RQ3) regarding the evolution and adoption of ROS 2, and our findings provide several important lessons and insights. With respect to RQ1, our comparative analysis of ROS 1 and ROS 2 demonstrates that ROS 2 overcomes many of its predecessor's limitations through a range of innovations. For instance, by transitioning to a fully distributed architecture with DDS-based communication, ROS 2 eliminates the single point of failure associated with the ROS Master. Modern executors and advanced scheduling techniques enhance middleware efficiency and support near-deterministic real-time performance, while robust security protocols improve protection against vulnerabilities. Despite these advancements, our review highlights ongoing challenges, including the need for consistent determinism and scalability in real-time applications.

Our investigation into RQ2 reveals that ROS 2's modular, DDS-based design has spurred its adoption across a wide array of domains such as autonomous vehicles, healthcare, agriculture, and industrial automation. Emerging trends indicate a growing reliance on ROS 2 in mission-critical and large-scale applications due to its improved interoperability and flexibility. The survey documented successful real-world deployments that showcase ROS 2's capability to manage complex tasks in dynamic environments, even as domain-specific challenges—like certification hurdles in healthcare and automotive sectors and integration issues with legacy systems—persist. Among our contributions, the open-source ROS Literature Database serves as a valuable resource for continued collaboration and targeted research. This database, along with figures such as 13 (ROS 2 Research Domains Taxonomy), 7 (ROS 2 Core Concepts Literature Taxonomy), and 11 (ROS 2 Simulators Comparison), as well as tables 11, 12, and 13 for ROS 2 libraries, datasets, and GitHub repositories, underscores the comprehensive nature of our analysis.

Regarding RQ3, our study identifies several technical, industrial, and organizational barriers that hinder the rapid adoption of ROS 2. Technical challenges include the complexity of migrating from ROS 1, the difficulties in standardizing Quality of Service profiles, and ensuring deterministic real-time performance. At the same time, industrial and organizational challenges such as resistance to change and limited empirical validation compound these issues. In response, we propose actionable solutions

 doi:10.20944/preprints202410.1204.v3

that involve developing integrated verification tools, streamlining bridging strategies between ROS 1 and ROS 2, and establishing standardized testing frameworks. These recommendations provide a clear roadmap for future research and development.

In summary, our survey demonstrates that ROS 2 not only effectively addresses the limitations of ROS 1 through architectural enhancements, improved middleware efficiency, and strengthened security but also maps out transformative trends and persistent challenges across various industries. The insights gained from this comprehensive review—complemented by our extensive literature database and detailed taxonomies—offer a robust foundation for accelerating the adoption of ROS 2 in both academic and industrial settings, ultimately paving the way for more resilient and efficient robotic systems.

Acknowledgments: The authors express their gratitude for the support provided by Prince Sultan University (PSU) in Saudi Arabia.

References

1. M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Ng, Ros: an open-source robot operating system, ICRA Workshop on Open Source Software 3 (01 2009).
2. D. Shim, S.-H. Kim, K. Cho, Open-source robotic platforms in ros for advanced robotics research, International Journal of Control, Automation and Systems 14 (5) (2016) 1205–1215.
3. M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, et al., End to end learning for self-driving cars, arXiv preprint arXiv:1604.07316 (2016).
4. L. Xie, J. Hu, J. Li, H. Zhang, Y. Wang, A novel mechanical design of a quadruped robot with force-position hybrid control, International Journal of Advanced Robotic Systems 15 (6) (2018) 1729881418810173.
5. J. Badger, D. Gooding, K. Ensley, K. Hambuchen, A. Thackston, Ros in space: A case study on robonaut 2 (2016). doi:10.1007/978-3-319-26054-9_13.
URL https://doi.org/10.1007/978-3-319-26054-9_13
6. H. Gao, Z. Wu, J. Liu, Cartographer: a system that provides real-time simultaneous localization and mapping (slam) in 2d and 3d across multiple platforms and sensor configurations, IEEE Robotics and Automation Letters 3 (2) (2018) 750–757.
7. G. Bradski, The opencv library, Dr. Dobb's Journal of Software Tools (2000).
8. R. B. Rusu, S. Cousins, 3d is here: Point cloud library (pcl), in: 2011 IEEE International Conference on Robotics and Automation, 2011, pp. 1–4. doi:10.1109/ICRA.2011.5980567.
9. F. P. Audonnet, A. Hamilton, G. Aragon-Camarasa, A systematic comparison of simulation software for robotic arm manipulation using ros2 (2022). doi:10.48550/ARXIV.2204.06433.
10. J. Zhang, F. Keramat, X. Yu, D. M. Hernández, J. P. Queralta, T. Westerlund, Distributed robotic systems in the edge-cloud continuum with ros 2: a review on novel architectures and technology readiness, in: 2022 Seventh International Conference on Fog and Mobile Edge Computing (FMEC), 2022, pp. 1–8. doi:10.1109/FMEC57183.2022.10062523.
11. S. Macenski, T. Moore, D. V. Lu, A. Merzlyakov, M. Ferguson, From the desks of ros maintainers: A survey of modern and capable mobile robotics algorithms in the robot operating system 2, Robotics and Autonomous Systems 168 (2023) 104493. doi:https://doi.org/10.1016/j.robot.2023.104493.
12. H.-S. Choi, D. Enright, H. Sobhani, Y. Xiang, H. Kim, Priority-driven real-time scheduling in ros 2: Potential and challenges (2022).
13. S. Macenski, T. Foote, B. Gerkey, C. Lalancette, W. Woodall, Robot operating system 2: Design, architecture, and uses in the wild, Science Robotics 7 (66) (2022) eabm6074. arXiv:https://www.science.org/doi/pdf/10.1126/scirobotics.abm6074, doi:10.1126/scirobotics.abm6074.
14. V. DiLuoffo, W. R. Michalson, B. Sunar, Robot operating system 2: The need for a holistic security approach to robotic architectures, International Journal of Advanced Robotic Systems 15 (3) (2018) 1729881418770011. arXiv:https://doi.org/10.1177/1729881418770011, doi:10.1177/1729881418770011.
15. M. Alhanahnah, Software quality assessment for robot operating system (2020). arXiv:2012.07196.
16. A. Bonci, F. Gaudeni, M. C. Giannini, S. Longhi, Robot operating system 2 (ros2)-based frameworks for increasing robot autonomy: A survey, Applied Sciences 13 (23) (2023). doi:10.3390/app132312796.
17. A. Koubâa, M. Sriti, H. Bennaceur, A. Ammar, Y. Javed, M. Alajlan, N. Al-Elaiwi, M. Tounsi, E. M. Shakshuki, COROS: A multi-agent software architecture for cooperative and autonomous service robots, in: A. Koubâa,

 doi:10.20944/preprints202410.1204.v3

J. R. M. de Dios (Eds.), Cooperative Robots and Sensor Networks 2015, Vol. 604 of Studies in Computational Intelligence, Springer, 2015, pp. 3–30. doi:10.1007/978-3-319-18299-5_1.

18. T. Foote, tf: The transform library, in: Technologies for Practical Robot Applications (TePRA), 2013 IEEE International Conference on, Open-Source Software workshop, 2013, pp. 1–6. doi:10.1109/TePRA.2013.6556373.
19. S. Macenski, F. Martín, R. White, J. G. Clavero, The marathon 2: A navigation system, in: 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2020, pp. 2718–2725. doi:10.1109/IROS45743.2020.9341207.
20. V. M. Vilches, R. White, G. Caiazza, M. Arguedas, Sros2: Usable cyber security tools for ros 2 (2022). doi:10.48550/ARXIV.2208.02615.
21. A. Jalil, J. Kobayashi, Efficacy of local cache for performance improvement of reliable data transmission in aggregated robot processing architecture, in: 2022 22nd International Conference on Control, Automation and Systems (ICCAS), 2022, pp. 1339–1344. doi:10.23919/ICCAS55662.2022.10003765.
22. S. Parra, S. Schneider, N. Hochgeschwender, Specifying qos requirements and capabilities for component-based robot software, in: 2021 IEEE/ACM 3rd International Workshop on Robotics Software Engineering (RoSE), 2021, pp. 29–36. doi:10.1109/RoSE52553.2021.00012.
23. B. Jaiswal, H. Tyagi, A. Gopalan, V. Sevani, Wiros: A qos software solution for ros2 in a wifi network, in: 2023 15th International Conference on COMmunication Systems & NETworkS (COMSNETS), 2023, pp. 216–218. doi:10.1109/COMSNETS56262.2023.10041412.
24. A. Jalil, J. Kobayashi, T. Saitoh, Qos balancing optimization in aggregated robot processing architecture: Rate and buffer, Journal of Advances in Artificial Life Robotics 3 (4) (2023) 209–213. doi:10.57417/jaalr.3.4_209.
25. J. Fernandez, B. Allen, P. Thulasiraman, B. Bingham, Performance study of the robot operating system 2 with qos and cyber security settings, in: 2020 IEEE International Systems Conference (SysCon), 2020, pp. 1–6. doi:10.1109/SysCon47679.2020.9275872.
26. Y. Liu, Y. Guan, X. Li, R. Wang, J. Zhang, Formal analysis and verification of dds in ros2, in: 2018 16th ACM/IEEE International Conference on Formal Methods and Models for System Design (MEMOCODE), 2018, pp. 1–5. doi:10.1109/MEMCOD.2018.8556970.
27. Q. Lu, X. Li, Y. Guan, R. Wang, Z. Shi, Modeling and analysis of data flow-oriented ros2 data distribution service, International Journal of Software and Informatics 11 (2021) 505–520. doi:10.21655/ijsi.1673-7288.00258.
28. L. Dust, R. Gu, C. Seceleanu, M. Ekström, S. Mubeen, Pattern-based verification of ros 2 nodes using uppaal, in: A. Cimatti, L. Titolo (Eds.), Formal Methods for Industrial Critical Systems, Springer Nature Switzerland, Cham, 2023, pp. 57–75.
29. M. Ghaffari Saadat, A. Ferrando, L. A. Dennis, M. Fisher, Rosmonitoring 2.0: Extending ros runtime verification to services and ordered topics, Electronic Proceedings in Theoretical Computer Science 411 (2024) 38–55. doi:10.4204/eptcs.411.3.
URL http://dx.doi.org/10.4204/EPTCS.411.3
30. H. Kang, C. Lee, W. Choi, S. Hong, Splash on ros 2: A runtime software framework for autonomous machines, in: 2021 18th International Conference on Ubiquitous Robots (UR), 2021, pp. 557–564. doi:10.1109/UR52253.2021.9494646.
31. W. Liu, J. Jin, H. Wu, Y. Gong, Z. Jiang, J. Zhai, Zoro: A robotic middleware combining high performance and high reliability, Journal of Parallel and Distributed Computing 166 (2022) 126–138. doi:https://doi.org/10.1016/j.jpdc.2022.04.010.
32. J. Kwok, S. Li, M. Lohstroh, E. A. Lee, Hprm: High-performance robotic middleware for intelligent autonomous systems (2024). arXiv:2412.01799.
URL https://arxiv.org/abs/2412.01799
33. Z. Zhao, Y. Xiang, Z. Zhou, K. Chong, H. Ma, P. Chen, Srfs: Parallel processing fault-tolerant ros2-based flight software for the space ranger cubesat (2024). arXiv:2412.08164.
URL https://arxiv.org/abs/2412.08164
34. Y. Maruyama, S. Kato, T. Azumi, Exploring the performance of ros2, in: 2016 International Conference on Embedded Software (EMSOFT), 2016, pp. 1–10. doi:10.1145/2968478.2968502.
35. J. Zhang, X. Yu, S. Ha, J. Peña Queralta, T. Westerlund, Comparison of middlewares in edge-to-edge and edge-to-cloud communication for distributed ros 2 systems, Journal of Intelligent & Robotic Systems 110 (4)

 doi:10.20944/preprints202410.1204.v3

(2024) 162. doi:10.1007/s10846-024-02187-z.
URL https://doi.org/10.1007/s10846-024-02187-z

36. W.-Y. Liang, Y. Yuan, H.-J. Lin, A performance study on the throughput and latency of zenoh, mqtt, kafka, and dds (2023). arXiv:2303.09419.
URL https://arxiv.org/abs/2303.09419
37. H. Teper, O. Bell, M. Günzel, C. Gill, J.-J. Chen, Bridging the gap between ros 2 and classical real-time scheduling for periodic tasks (2024). arXiv:2408.03696.
URL https://arxiv.org/abs/2408.03696
38. T. Blaß, D. Casini, S. Bozhko, B. B. Brandenburg, A ros 2 response-time analysis exploiting starvation freedom and execution-time variance, in: 2021 IEEE Real-Time Systems Symposium (RTSS), 2021, pp. 41–53. doi:10.1109/RTSS52674.2021.00016.
39. L. J. Dust, E. Persson, M. Ekström, S. Mubeen, C. Seceleanu, R. Gu, Experimental evaluation of callback behavior in ros 2 executors, in: 2023 IEEE 28th International Conference on Emerging Technologies and Factory Automation (ETFA), 2023, pp. 1–8. doi:10.1109/ETFA54631.2023.10275668.
40. P. Ghiglino, G. Sarabia, The ring-buffer ros2 executor: a novel approach for real-time ros2 space applications, in: 2023 IEEE Space Computing Conference (SCC), 2023, pp. 80–85. doi:10.1109/SCC57168.2023.00021.
41. H. Sobhani, H. Choi, H. Kim, Timing analysis and priority-driven enhancements of ros 2 multi-threaded executors, in: 2023 IEEE 29th Real-Time and Embedded Technology and Applications Symposium (RTAS), 2023, pp. 106–118. doi:10.1109/RTAS58335.2023.00016.
42. H. Choi, Y. Xiang, H. Kim, Picas: New design of priority-driven chain-aware scheduling for ros2, in: 2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS), 2021, pp. 251–263. doi:10.1109/RTAS52030.2021.00028.
43. Y. Tang, Z. Feng, N. Guan, X. Jiang, M. Lv, Q. Deng, W. Yi, Response time analysis and priority assignment of processing chains on ros2 executors, in: 2020 IEEE Real-Time Systems Symposium (RTSS), 2020, pp. 231–243. doi:10.1109/RTSS49844.2020.00030.
44. A. K. Chaaban, A new algorithm for real-time scheduling and resource mapping for robot operating systems (ros), Applied Sciences 13 (3) (2023). doi:10.3390/app13031532.
URL https://www.mdpi.com/2076-3417/13/3/1532
45. Y. Saito, F. Sato, T. Azumi, S. Kato, N. Nishio, Rosch:real-time scheduling framework for ros, in: 2018 IEEE 24th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), 2018, pp. 52–58. doi:10.1109/RTCSA.2018.00015.
46. T. Blass, A. Hamann, R. Lange, D. Ziegenbein, B. B. Brandenburg, Automatic latency management for ros 2: Benefits, challenges, and open problems, in: 2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS), 2021, pp. 264–277. doi:10.1109/RTAS52030.2021.00029.
47. T. Kronauer, J. Pohlmann, M. Matthé, T. Smejkal, G. Fettweis, Latency analysis of ros2 multi-node systems, in: 2021 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI), 2021, pp. 1–7. doi:10.1109/MFI52462.2021.9591166.
48. L. Puck, P. Keller, T. Schnell, C. Plasberg, A. Tanev, G. Heppner, A. Roennau, R. Dillmann, Performance evaluation of real-time ros2 robotic control in a time-synchronized distributed network, in: 2021 IEEE 17th International Conference on Automation Science and Engineering (CASE), 2021, pp. 1670–1676. doi:10.1109/CASE49439.2021.9551447.
49. C. Plasberg, H. Nessau, D. Timmermann, C. Eichmann, A. Roennau, R. Dillmann, Towards distributed real-time capable robotic control using ros2, in: 2022 IEEE 18th International Conference on Automation Science and Engineering (CASE), 2022, pp. 2205–2210. doi:10.1109/CASE49997.2022.9926569.
50. H. Teper, T. Betz, M. Günzel, D. Ebner, G. Von Der Brüggen, J. Betz, J.-J. Chen, End-to-end timing analysis and optimization of multi-executor ros 2 systems, in: 2024 IEEE 30th Real-Time and Embedded Technology and Applications Symposium (RTAS), 2024, pp. 212–224. doi:10.1109/RTAS61025.2024.00025.
51. T. Betz, M. Schmeller, H. Teper, J. Betz, How fast is my software? latency evaluation for a ros 2 autonomous driving software, in: 2023 IEEE Intelligent Vehicles Symposium (IV), 2023, pp. 1–6. doi:10.1109/IV55152.2023.10186585.
52. Y. Tang, N. Guan, X. Jiang, X. Luo, W. Yi, Real-time performance analysis of processing systems on ros 2 executors, in: 2023 IEEE 29th Real-Time and Embedded Technology and Applications Symposium (RTAS), 2023, pp. 80–92. doi:10.1109/RTAS58335.2023.00014.

 doi:10.20944/preprints202410.1204.v3

53. H. Teper, T. Betz, G. Von Der Brüggen, K.-H. Chen, J. Betz, J.-J. Chen, Timing-aware ros 2 architecture and system optimization, in: 2023 IEEE 29th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), 2023, pp. 206–215. doi:10.1109/RTCSA58653.2023.00032.
54. C. S. V. Gutiérrez, L. U. S. Juan, I. Z. Ugarte, V. M. Vilches, Towards a distributed and real-time framework for robots: Evaluation of ros 2.0 communications for real-time robotic applications (2018). arXiv:1809.02595.
55. H. Abaza, D. Roy, S. Fan, S. Saidi, A. Motakis, Trace-enabled timing model synthesis for ros2-based autonomous applications, in: 2024 Design, Automation & Test in Europe Conference & Exhibition (DATE), IEEE, 2024, pp. 1–6. doi:10.23919/DATE58400.2024.10546872.
56. Y. Ye, Z. Nie, X. Liu, F. Xie, Z. Li, P. Li, Ros2 real-time performance optimization and evaluation, Chinese Journal of Mechanical Engineering 36 (1) (2023) 144. doi:10.1186/s10033-023-00976-5.
URL https://doi.org/10.1186/s10033-023-00976-5
57. J. Park, R. Delgado, B. W. Choi, Real-time characteristics of ros 2.0 in multiagent robot systems: An empirical study, IEEE Access 8 (2020) 154637–154651. doi:10.1109/ACCESS.2020.3018122.
58. T. Betz, L. Wen, F. Pan, G. Kaljavesi, A. Zuepke, A. Bastoni, M. Caccamo, A. Knoll, J. Betz, A containerized microservice architecture for a ros 2 autonomous driving software: An end-to-end latency evaluation (2024). arXiv:2404.12683.
59. Y. Yu, S. Lee, Measurements of the benefits of edge computing on autonomous driving, in: 2022 13th International Conference on Information and Communication Technology Convergence (ICTC), 2022, pp. 2155–2159. doi:10.1109/ICTC55196.2022.9952693.
60. V. M. Vilches, G. Corradi, Adaptive computing in robotics, leveraging ros 2 to enable software-defined hardware for fpgas, ArXiv abs/2109.03276 (2021).
61. C. Lienen, S. H. Middeke, M. Platzner, fpgadds: An intra-fpga data distribution service for ros 2 robotics applications (2023). arXiv:2303.00532.
62. C. Lienen, M. Platzner, Reconros executor: Event-driven programming of fpga-accelerated ros 2 applications, ArXiv abs/2201.07454 (2022).
63. C. Lienen, P. A. Nowosad, M. Platzner, Mapping and optimizing communication in ros 2-based applications on configurable system-on-chip platforms, in: Proceedings of the 2023 9th International Conference on Robotics and Artificial Intelligence, ICRAI '23, Association for Computing Machinery, New York, NY, USA, 2024, p. 23–31. doi:10.1145/3637843.3637846.
URL https://doi.org/10.1145/3637843.3637846
64. V. Mayoral-Vilches, J. M. Reina-Muñoz, M. Crespo-Álvarez, D. Mayoral-Vilches, Ros 2 on a chip, achieving brain-like speeds and efficiency in robotic networking (2024). arXiv:2404.18208.
65. D. Enright, Y. Xiang, H. Choi, H. Kim, Paam: A framework for coordinated and priority-driven accelerator management in ros 2 (2024). arXiv:2404.06452.
66. R. Li, T. Hu, X. Jiang, L. Li, W. Xing, Q. Deng, N. Guan, Rosgm: A real-time gpu management framework with plug-in policies for ros 2, in: 2023 IEEE 29th Real-Time and Embedded Technology and Applications Symposium (RTAS), 2023, pp. 93–105. doi:10.1109/RTAS58335.2023.00015.
67. M. De Marchi, F. Lumpp, E. Martini, M. Boldo, S. Aldegheri, N. Bombieri, Efficient ros-compliant cpu-igpu communication on embedded platforms, Journal of Low Power Electronics and Applications 11 (2) (2021). doi:10.3390/jlpea11020024.
URL https://www.mdpi.com/2079-9268/11/2/24
68. A. Mamani-Saico, P. R. Yanyachi, Implementation and performance study of the micro-ros/ros2 framework to algorithm design for attitude determination and control system, IEEE Access 11 (2023) 128451–128460. doi:10.1109/ACCESS.2023.3330441.
69. Z. Wang, S. Liu, D. Ji, W. Yi, Improving real-time performance of micro-ros with priority-driven chain-aware scheduling, Electronics 13 (9) (2024). doi:10.3390/electronics13091658.
URL https://www.mdpi.com/2079-9292/13/9/1658
70. D. Patel, C. Maiti, S. Muthuswamy, Real-time performance monitoring of a cnc milling machine using ros 2 and aws iot towards industry 4.0, in: IEEE EUROCON 2023 - 20th International Conference on Smart Technologies, 2023, pp. 776–781. doi:10.1109/EUROCON56442.2023.10199020.
71. Y. Zhang, B. Luo, A. Mukhopadhyay, D. Stojcsics, D. Elenius, A. Roy, S. Jha, M. Maroti, X. Koutsoukos, G. Karsai, A. Dubey, Shrinking pomcp: A framework for real-time uav search and rescue (2024). arXiv:2411.12967.
URL https://arxiv.org/abs/2411.12967

 doi:10.20944/preprints202410.1204.v3

72. G. Deng, G. Xu, Y. Zhou, T. Zhang, Y. Liu, On the (in)security of secure ros2, in: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS '22, Association for Computing Machinery, New York, NY, USA, 2022, p. 739–753. doi:10.1145/3548606.3560681.
URL https://doi.org/10.1145/3548606.3560681
73. N. Goerke, D. Timmermann, I. Baumgart, Who controls your robot? an evaluation of ros security mechanisms, in: 2021 7th International Conference on Automation, Robotics and Applications (ICARA), 2021, pp. 60–66. doi:10.1109/ICARA51699.2021.9376468.
74. S. Yang, J. Guo, X. Rui, Formal analysis and detection for ros2 communication security vulnerability, Electronics 13 (9) (2024). doi:10.3390/electronics13091762.
75. S. Sandoval, P. Thulasiraman, Cyber security assessment of the robot operating system 2 for aerial networks, in: 2019 IEEE International Systems Conference (SysCon), 2019, pp. 1–8. doi:10.1109/SYSCON.2019.8836824.
76. S. Johnson, A. Borah, A. Paranjothi, J. Thomas, Ros-lighthouse: An intrusion detection system (ids) in ros using ensemble learning (10 2024).
77. E. Soriano-Salvador, F. Martín-Rico, G. G. Múzquiz, Implementing a robot intrusion prevention system (rips) for ros 2 (2024). arXiv:2412.19272.
URL https://arxiv.org/abs/2412.19272
78. L. Fu, S. Salimi, J. P. Queralta, T. Westerlund, Event-driven fabric blockchain - ros 2 interface: Towards secure and auditable teleoperation of mobile robots (2023). arXiv:2304.00781.
79. S. Salimi, F. Keramat, T. Westerlund, J. P. Queralta, A customizable conflict resolution and attribute-based access control framework for multi-robot systems (2023). arXiv:2308.16482.
80. R. White, H. I. Christensen, G. Caiazza, A. Cortesi, Procedurally provisioned access control for robotic systems, in: 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2018, pp. 1–9. doi:10.1109/IROS.2018.8594462.
81. F. Yang, S. S. Zhan, Y. Wang, C. Huang, Q. Zhu, Case study: Runtime safety verification of neural network controlled system (2024). arXiv:2408.08592.
82. T. Zander, J. Wohnig, J. Beyerer, Understanding, describing, and mitigating the flow of personal data in ros 2 systems to comply with the gdpr and beyond.*, in: 2024 IEEE International Conference on Advanced Robotics and Its Social Impacts (ARSO), 2024, pp. 146–152. doi:10.1109/ARSO60199.2024.10557807.
83. E. Dey, M. Walczak, M. S. Anwar, N. Roy, A reliable and low latency synchronizing middleware for co-simulation of a heterogeneous multi-robot systems (2022). doi:10.48550/ARXIV.2211.05359.
84. M. F. Ginting, K. Otsu, J. A. Edlund, J. Gao, A.-A. Agha-Mohammadi, Chord: Distributed data-sharing via hybrid ros 1 and 2 for multi-robot exploration of large-scale complex environments, IEEE Robotics and Automation Letters 6 (3) (2021) 5064–5071. doi:10.1109/LRA.2021.3061393.
85. X. Yu, I. Catalano, P. T. Morón, S. Salimpour, T. Westerlund, J. P. Queralta, Fusing odometry, uwb ranging, and spatial detections for relative multi-robot localization (2023). arXiv:2304.06264.
URL https://arxiv.org/abs/2304.06264
86. Z. Gao, T. Wanyama, I. Singh, A. Gadhrri, R. Schmidt, From industry 4.0 to robotics 4.0 - a conceptual framework for collaborative and intelligent robotic systems, Procedia Manufacturing 46 (2020) 591–599, 13th International Conference Interdisciplinarity in Engineering, INTER-ENG 2019, 3–4 October 2019, Targu Mures, Romania. doi:https://doi.org/10.1016/j.promfg.2020.03.085.
URL https://www.sciencedirect.com/science/article/pii/S235197892030963X
87. L. Puck, P. Keller, T. Schnell, C. Plasberg, A. Tanev, G. Heppner, A. Roennau, R. Dillmann, Distributed and synchronized setup towards real-time robotic control using ros2 on linux, in: 2020 IEEE 16th International Conference on Automation Science and Engineering (CASE), 2020, pp. 1287–1293. doi:10.1109/CASE48305.2020.9217010.
88. J. Zhang, F. Keramat, X. Yu, D. M. Hernández, J. P. Queralta, T. Westerlund, Distributed robotic systems in the edge-cloud continuum with ros 2: a review on novel architectures and technology readiness, in: 2022 Seventh International Conference on Fog and Mobile Edge Computing (FMEC), 2022, pp. 1–8. doi:10.1109/FMEC57183.2022.10062523.
89. A. Jalil, J. Kobayashi, Experimental analyses of an efficient aggregated robot processing with cache-control for multi-robot system, in: 2020 20th International Conference on Control, Automation and Systems (ICCAS), 2020, pp. 1105–1109. doi:10.23919/ICCAS50221.2020.9268225.

 doi:10.20944/preprints202410.1204.v3

90. L. Dauphin, E. Baccelli, C. Adjih, Riot-ros2: Low-cost robots in iot controlled via information-centric networking, in: 2018 IFIP/IEEE International Conference on Performance Evaluation and Modeling in Wired and Wireless Networks (PEMWN), 2018, pp. 1–6. doi:10.23919/PEMWN.2018.8548798.
91. M. De Marchi, N. Bombieri, Orchestration-aware optimization of ros2 communication protocols, in: 2024 Design, Automation & Test in Europe Conference & Exhibition (DATE), 2024, pp. 1–6. doi:10.23919/DATE58400.2024.10546777.
92. A. Jalil, J. Kobayashi, T. Saitoh, Performance improvement of multi-robot data transmission in aggregated robot processing architecture with caches and qos balancing optimization, Robotics 12 (3) (2023). doi:10.3390/robotics12030087.
93. C. Bédard, P.-Y. Lajoie, G. Beltrame, M. Dagenais, Message flow analysis with complex causal links for distributed ROS 2 systems, Robotics and Autonomous Systems 161 (2023) 104361. doi:10.1016/j.robot.2022.104361.
94. L. J. Dust, E. Persson, M. Ekström, S. Mubeen, E. Dean, Quantitative analysis of communication handling for centralized multi-agent robot systems using ros2, in: 2022 IEEE 20th International Conference on Industrial Informatics (INDIN), 2022, pp. 624–629. doi:10.1109/INDIN51773.2022.9976160.
95. P. Thulasiraman, Z. Chen, B. Allen, B. Bingham, Evaluation of the robot operating system 2 in lossy unmanned networks, in: 2020 IEEE International Systems Conference (SysCon), 2020, pp. 1–8. doi:10.1109/SysCon47679.2020.9275849.
96. L. P. Chovet, G. M. Garcia, A. Bera, A. Richard, K. Yoshida, M. A. Olivares-Mendez, Performance comparison of ros2 middlewares for multi-robot mesh networks in planetary exploration (2024). arXiv:2407.03091.
97. S. Macenski, T. Foote, B. Gerkey, C. Lalancette, W. Woodall, Robot operating system 2: Design, architecture, and uses in the wild, Science Robotics 7 (66) (2022) eabm6074. arXiv:https://www.science.org/doi/pdf/10.1126/scirobotics.abm6074, doi:10.1126/scirobotics.abm6074.
URL https://www.science.org/doi/abs/10.1126/scirobotics.abm6074
98. H. Paul, Z. Qiu, Z. Wang, S. Hirai, S. Kawamura, A ros 2 based robotic system to pick-and-place granular food materials, in: 2022 IEEE International Conference on Robotics and Biomimetics (ROBIO), 2022, pp. 99–104. doi:10.1109/ROBIO55434.2022.10011782.
99. A. Mohan, A. R. Krishnan, Design and simulation of an autonomous floor cleaning robot with optional uv sterilization, in: 2022 IEEE 2nd Mysore Sub Section International Conference (MysuruCon), 2022, pp. 1–6. doi:10.1109/MysuruCon55714.2022.9972558.
100. L. Arcos, K. Vicente, P. Cruz, J. Abad, I. Zambrano, A ros2 based trajectory tracking controller of a 3ups-1rpu parallel robot for knee rehabilitation, in: 2022 IEEE Sixth Ecuador Technical Chapters Meeting (ETCM), 2022, pp. 1–6. doi:10.1109/ETCM56276.2022.9935755.
101. C. Eyberg, L. Karstensen, T. Pusch, J. Horsch, J. Langejürgen, A ros2-based testbed environment for endovascular robotic systems, Current Directions in Biomedical Engineering 8 (1) (2022) 89–92 [cited 2022-12-20]. doi:doi:10.1515/cdbme-2022-0023.
URL https://doi.org/10.1515/cdbme-2022-0023
102. J. Lu, S. Hossain, W. Lam, W. Sheng, H. Bai, A research testbed for intelligent and cooperative driving in mixed traffic, IEEE Transactions on Intelligent Transportation Systems (2024) 1–13doi:10.1109/TITS.2024.3375297.
103. E. Sani, A. Sgorbissa, S. Carpin, Improving the ros 2 navigation stack with real-time local costmap updates for agricultural applications (2024). arXiv:2407.18535.
URL https://arxiv.org/abs/2407.18535
104. X. Li, J. Park, C. Reberg-Horton, S. Mirsky, E. Lobaton, L. Xiang, Photorealistic arm robot simulation for 3d plant reconstruction and automatic annotation using unreal engine 5, 2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW) (2024) 5480–5488doi:10.1109/CVPRW63382.2024.00557.
URL https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=10678093
105. B. Chandrasekaran, A framework for multi-robot agricultural field monitoring with fault tolerance using ros, 2024 9th International Conference on Robotics and Automation Engineering (ICRAE) (2024) 55–61doi:10.1109/ICRAE64368.2024.10851661.
URL https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=10851661
106. Z. Qiu, H. Paul, Z. Wang, S. Hirai, S. Kawamura, An evaluation system of robotic end-effectors for food handling, Foods 12 (22) (2023). doi:10.3390/foods12224062.
URL https://www.mdpi.com/2304-8158/12/22/4062

 doi:10.20944/preprints202410.1204.v3

107. H. Paul, Z. Qiu, Z. Wang, S. Hirai, S. Kawamura, A ros 2 based robotic system to pick-and-place granular food materials, in: 2022 IEEE International Conference on Robotics and Biomimetics (ROBIO), 2022, pp. 99–104. doi:10.1109/ROBIO55434.2022.10011782.
108. A. Rana, A. Petitti, A. Ugenti, R. Galati, G. Reina, A. Milella, Toward digital twin of off-road vehicles using robot simulation frameworks, IEEE Access 12 (2024) 178047–178061. doi:10.1109/ACCESS.2024.3509226.
URL https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=10771752
109. M. Desai, H. Mewada, H. Chhajed, B. Kundaliya, Gesture dependable robotics arm movement: Ros and machine learning perspective, 2024 9th South-East Europe Design Automation, Computer Engineering, Computer Networks and Social Media Conference (SEEDA-CECNSM) (2024) 1–6doi:10.1109/SEEDA-CECNSM63478.2024.00010.
URL https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=10734617
110. T. Kołcon, A. Malki, M. Maciaś, Smart warehouse as an example of micro-ros application, in: R. Szewczyk, C. Zieliński, M. Kaliczyńska (Eds.), Automation 2022: New Solutions and Technologies for Automation, Robotics and Measurement Techniques, Springer International Publishing, Cham, 2022, pp. 264–272.
111. D. Tozadore, A. M. Rusu, Teachers, take care of the essential. the rest is story: Using llm and social robots for content approaching by storytelling, 2024 33rd IEEE International Conference on Robot and Human Interactive Communication (ROMAN) (2024) 2125–2130doi:10.1109/RO-MAN60168.2024.10731274.
URL https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=10731274
112. K. K. Pandey, T. Shah, S. Yadrave, S. Shinde, V. Pagare, Design and development of an autonomous robot assistant, in: B. B. V. L. Deepak, M. V. A. R. Bahubalendruni, D. R. K. Parhi, B. B. Biswal (Eds.), Intelligent Manufacturing Systems in Industry 4.0, Springer Nature Singapore, Singapore, 2023, pp. 381–389.
113. A. Barciś, M. Barciś, C. Bettstetter, Robots that sync and swarm: A proof of concept in ros 2, in: 2019 International Symposium on Multi-Robot and Multi-Agent Systems (MRS), 2019, pp. 98–104. doi:10.1109/MRS.2019.8901095.
114. M. Bosello, D. Aguiari, Y. Keuter, E. Pallotta, S. Kiade, G. Caminati, F. Pinzarrone, J. Halepota, J. Panerati, G. Pau, Race against the machine: A fully-annotated, open-design dataset of autonomous and piloted high-speed flight, IEEE Robotics and Automation Letters 9 (4) (2024) 3799–3806. doi:10.1109/LRA.2024.3371288.
115. T. Becker, J. A. Fabro, A. Schneider De Oliveira, L. P. Reis, Adding conscious aspects in virtual robot navigation through baars-franklin's cognitive architecture, in: 2015 IEEE International Conference on Autonomous Robot Systems and Competitions, 2015, pp. 204–209. doi:10.1109/ICARSC.2015.34.
116. G. Abbate, A. Giusti, A. Paolillo, B. Gromov, L. Gambardella, A.-E. Rizzoli, J. Guzzi, Pointit: A ros toolkit for interacting with co-located robots using pointing gestures, in: Proceedings of the 2022 ACM/IEEE International Conference on Human-Robot Interaction, HRI '22, IEEE Press, 2022, p. 608–612.
117. C. Bédard, I. Lütkebohle, M. Dagenais, ros2_tracing: Multipurpose low-overhead framework for real-time tracing of ros 2, IEEE Robotics and Automation Letters 7 (3) (2022) 6511–6518. doi:10.1109/LRA.2022.3174346.
118. G. Asín-Prieto, F. M. Rico, D. Torricelli, A ros2-based approach to enable simultaneous and real-time tracking of humans and exoskeleton motion, in: D. Torricelli, M. Akay, J. L. Pons (Eds.), Converging Clinical and Engineering Research on Neurorehabilitation IV, Springer International Publishing, Cham, 2022, pp. 133–137.
119. A. Bonci, F. Gaudeni, M. C. Giannini, S. Longhi, Robot operating system 2 (ros2)-based frameworks for increasing robot autonomy: A survey, Applied Sciences 13 (23) (2023). doi:10.3390/app132312796.
URL https://www.mdpi.com/2076-3417/13/23/12796
120. J. Ichnowski, K. Chen, K. Dharmarajan, S. Adebola, M. Danielczuk, V. Mayoral-Vilches, N. Jha, H. Zhan, E. LLontop, D. Xu, J. Kubiatowicz, I. Stoica, J. Gonzalez, K. Goldberg, Fogros2: An adaptive platform for cloud and fog robotics using ros 2 (2022). doi:10.48550/ARXIV.2205.09778.
121. M. Chemnitz, H. Schorn, K. TÃ¶reki, A. Fritz, Integrating plcs and robots into the ros 2 ecosystem, 2024 IEEE 29th International Conference on Emerging Technologies and Factory Automation (ETFA) (2024) 1–4doi:10.1109/ETFA61755.2024.10710873.
URL https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=10710873
122. S. Kashid, A. D. Kumat, Hierarchically decentralized heterogeneous multi-robot task allocation system (2024). arXiv:2405.02484.
URL https://arxiv.org/abs/2405.02484
123. Y. Dang, Y. Huang, X. Shen, D. Zhu, Z. Chu, Incremental sparse gaussian process-based model predictive control for trajectory tracking of unmanned underwater vehicles, IEEE Robotics and Automation Letters

 doi:10.20944/preprints202410.1204.v3

10 (3) (2025) 2327–2334. doi:10.1109/LRA.2025.3530115.
URL https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=10851411

124. A. Almusayli, T. Zia, E.-u.-H. Qazi, Drone forensics: An innovative approach to the forensic investigation of drone accidents based on digital twin technology, Technologies 12 (1) (2024). doi:10.3390/technologies12010011.
125. Z. Zhao, H. Su, M. Sun, Environment-aware strategy for multi-sensor fusion based on improved yolo networks, 2024 8th International Conference on Imaging, Signal Processing and Communications (ICISPC) (2024) 156–164doi:10.1109/ICISPC63824.2024.00035.
URL https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=10677708
126. F. Maresca, F. Grazioli, A. Albanese, V. Sciancalepore, G. Negri, X. Costa-Perez, Are you a robot? detecting autonomous vehicles from behavior analysis (2024). arXiv:2403.09571.
127. M. Unger, E. Horváth, D. Pup, C. R. Pozna, Towards robust lidar lane clustering for autonomous vehicle perception in ros 2, in: 2024 IEEE International Conference on Mobility, Operations, Services and Technologies (MOST), 2024, pp. 229–234. doi:10.1109/MOST60774.2024.00031.
128. Z. Guo, S. Perminov, M. Konenkov, D. Tsetserukou, Hawkdrive: A transformer-driven visual perception system for autonomous driving in night scene (2024). arXiv:2404.04653.
129. O. Robotics, Gazebo sim: Next generation robot simulation, https://gazebosim.org/docs/latest/tutorials/ (2023).
130. NVIDIA, Isaac sim: Robotics simulation and synthetic data generation, https://docs.isaacsim.omniverse.nvidia.com/latest/index.html (2023).
131. PyBullet, Pybullet physics simulation for robotics, https://pybullet.org/wordpress/index.php/forum-2/ (2023).
132. Cyberbotics, Webots: Robot simulator, https://docs.ros.org/en/humble/Tutorials/Advanced/Simulators/Webots/Simulation-Webots.html (2023).
133. C. Robotics, Coppeliasim: Create, compose, simulate, any robot, https://manual.coppeliarobotics.com/en/ros2Tutorial.htm (2023).
134. A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, V. Koltun, Carla: An open urban driving simulator, in: Proceedings of the 1st Annual Conference on Robot Learning, 2017, pp. 1–16.
135. S. Chitta, Moveit!: An introduction (2016). doi:10.1007/978-3-319-26054-9_1.
136. P. M. Fresnillo, S. Vasudevan, W. M. Mohammed, J. L. M. Lastra, J. A. P. Garcia, Extending the motion planning framework—moveit with advanced manipulation functions for industrial applications, Robotics and Computer-Integrated Manufacturing 83 (2023) 102559. doi:10.1016/j.rcim.2023.102559.
137. ROS-Industrial Consortium, Ros-industrial: Industrial extensions for the robot operating system, https://rosindustrial.org/ (2023).
138. O. Robotics, Micro-ros: Ros 2 for microcontrollers, https://micro.ros.org/ (2024).
139. Z. Wang, S. Liu, D. Ji, Y. Wang, Improving real-time performance of micro-ros with priority-driven chain-aware scheduling, Electronics 13 (91658) (Apr. 2024). doi:10.3390/electronics13091658.
140. J. Ashkanazy, A. Spalter, J. Hays, L. Hiatt, R. Leontie, C. Henshaw, Collision-free inverse kinematics through qp optimization (ikinqp), arXiv preprint (2023). doi:10.48550/arXiv.2308.15268.
141. S. Jotawar, M. Soni, S. Kumar, Motion planning for an automated pick and place robot in a retail warehouse (2017). doi:10.1145/3132446.3134904.
142. J. Staschulat, R. Lange, D. N. Dasari, Budget-based real-time executor for micro-ros, ArXiv preprint, abs/2105.05590, https://api.semanticscholar.org/CorpusID:234469855 (2021).
143. RViz2, 3d visualization tool for the ros 2 ecosystem, https://github.com/ros2/rviz (2017).
144. A. B. Probe, A. Oyake, S. W. Chambers, M. Deans, G. Brat, N. Cramer, B. Kempa, B. Roberts, K. Hambuchen, Space ros: An open-source framework for space robotics and flight software (2023). doi:10.2514/6.2023-2709.
145. S. Macenski, M. Booker, J. Wallace, Open-source, cost-aware kinematically feasible planning for mobile and surface robotics, arXiv preprint (2024). doi:10.48550/arXiv.2401.13078.
146. A. B. Probe, M. C. Deans, Space ros-aco: Space robot operating system (2022).
147. A. Probe, A. Oyake, S. W. Chambers, M. Deans, G. Brat, N. B. Cramer, B. Kempa, B. Roberts, K. Hambuchen, Space ros: An open-source framework for space robotics and flight software (2023). arXiv:https://arc.aiaa.org/doi/pdf/10.2514/6.2023-2709, doi:10.2514/6.2023-2709.
URL https://arc.aiaa.org/doi/abs/10.2514/6.2023-2709

 doi:10.20944/preprints202410.1204.v3

148. R. B. Rusu, S. Cousins, 3d is here: Point cloud library (pcl), in: IEEE International Conference on Robotics and Automation (ICRA), Shanghai, China, 2011, pp. 1–4. doi:10.1109/ICRA.2011.5980567.
149. TF2, Tf2: The transform library for ros 2, https://docs.ros.org/en/foxy/Tutorials/Intermediate/Tf2/Introduction-To-Tf2.html, rOS 2 Documentation (2017).
150. BehaviorTree.CPP, Behavior tree library for robotics and ai, https://www.behaviortree.dev/docs/3.8/category/tutorial---basics/, version 3.8 (2021).
151. T. Ribeaud, C. Sprenger, Behavior trees based flexible task planner built on ros2 framework, in: IEEE 27th International Conference on Emerging Technologies and Factory Automation (ETFA), 2022, pp. 1–4. doi:10.1109/ETFA52439.2022.9921683.
152. MAVROS, Mavlink communication bridge for ros 2, https://github.com/mavlink/mavros (2017).
153. SMACC, State machine library for complex robot decision-making in ros 2, https://smacc.dev/ (2017).
154. Autoware.Auto, Open-source autonomous driving framework built on ros 2, https://autowarefoundation.github.io/autoware.auto/ (2017).
155. CyberRT, Apollo cyber rt: Real-time middleware for autonomous driving, https://github.com/ApolloAuto/cyber (2017).
156. ROS 2 Lifecycle, Managed node lifecycle in ros 2, https://design.ros2.org/articles/node_lifecycle.html (2017).
157. ros_control, Hardware interface and controller framework for ros, https://control.ros.org/ (2017).
158. ROS 2 Community Packages, Domain-specific and open-source extensions for ros 2, https://index.ros.org/packages/ (2017).
159. M. Fernandez-Cortizas, M. Molina, P. Arias-Perez, R. Perez-Segui, D. Perez-Saura, P. Campoy, Aerostack2: A software framework for developing multi-robot aerial systems (2023). arXiv:2303.18237.
160. N. Passalis, S. Pedrazzi, R. Babuska, W. Burgard, D. Dias, F. Ferro, M. Gabbouj, O. Green, A. Iosifidis, E. Kayacan, J. Kober, O. Michel, N. Nikolaidis, P. Nousi, R. Pieters, M. Tzelepi, A. Valada, A. Tefas, Opendr: An open toolkit for enabling high performance, low footprint deep learning for robotics (2022). arXiv:2203.00403.
URL https://arxiv.org/abs/2203.00403
161. A. Angleraud, A. Ekrekli, K. Samarawickrama, G. Sharma, R. Pieters, Sensor-based human–robot collaboration for industrial tasks, Robotics and Computer-Integrated Manufacturing 86 (2024) 102663. doi:https://doi.org/10.1016/j.rcim.2023.102663.
162. G. Rong, B. H. Shin, H. Tabatabaee, Q. Lu, S. Lemke, M. Možeiko, E. Boise, G. Uhm, M. Gerow, S. Mehta, E. Agafonov, T. H. Kim, E. Sterner, K. Ushiroda, M. Reyes, D. Zelenkovsky, S. Kim, Lgsvl simulator: A high fidelity simulator for autonomous driving (2020). arXiv:2005.03778.
163. L. Pichierri, A. Testa, G. Notarstefano, Crazychoir: Flying swarms of crazyflie quadrotors in ros 2, ArXiv abs/2302.00716 (2023).
164. Y. Zhang, C. Wurll, B. Hein, Kuberos: A unified platform for automated and scalable deployment of ros2-based multi-robot applications, in: 2023 IEEE International Conference on Robotics and Automation (ICRA), 2023, pp. 9097–9103. doi:10.1109/ICRA48891.2023.10160632.
165. T. K. Kaiser, M. J. Begemann, T. Plattenteich, L. Schilling, G. Schildbach, H. Hamann, Ros2swarm - a ros 2 package for swarm robot behaviors, in: 2022 International Conference on Robotics and Automation (ICRA), 2022, pp. 6875–6881. doi:10.1109/ICRA46639.2022.9812417.
166. J. Blumenkamp, A. Shankar, M. Bettini, J. Bird, A. Prorok, The cambridge robomaster: An agile multi-robot research platform (2024). arXiv:2405.02198.
167. A. Pacheco, U. Denis, R. Zakir, V. Strobel, A. Reina, M. Dorigo, Toychain: A simple blockchain for research in swarm robotics (2024). arXiv:2407.06630.
URL https://arxiv.org/abs/2407.06630
168. J.-B. Castillo-Sánchez, E. González-Parada, J. M. Cano-García, A novel testbed for evaluating ros 2 robot swarm wireless communications, 2024 IEEE 22nd Mediterranean Electrotechnical Conference (MELECON) (2024) 68–73.
169. A. Testa, A. Camisa, G. Notarstefano, Choirbot: A ros 2 toolbox for cooperative robotics, IEEE Robotics and Automation Letters 6 (2) (2021) 2714–2720. doi:10.1109/LRA.2021.3061366.
170. F. Alzetta, P. Giorgini, Towards a real-time bdi model for ros 2 (2019).
171. A. Koubaa, Rosgpt: Next-generation human-robot interaction with chatgpt and ros (04 2023). doi:10.20944/preprints202304.0827.v1.

 doi:10.20944/preprints202410.1204.v3

172. A. Bono, K. Brameld, L. D'Alfonso, G. Fedele, Open access nao (oan): a ros2-based software framework for hri applications with the nao robot (2024). arXiv:2403.13960.
173. S. Fabian, O. v. Stryk, Open-source tools for efficient ros and ros2-based 2d human-robot interface development, in: 2021 European Conference on Mobile Robots (ECMR), 2021, pp. 1–6. doi:10.1109/ECMR50962.2021.9568801.
174. G. Abbate, A. Giusti, A. Paolillo, B. Gromov, L. Gambardella, A.-E. Rizzoli, J. Guzzi, Pointit: A ros toolkit for interacting with co-located robots using pointing gestures, in: Proceedings of the 2022 ACM/IEEE International Conference on Human-Robot Interaction, HRI '22, IEEE Press, 2022, p. 608–612.
175. W. Jo, R. Wilson, J. Kim, S. McGuire, B. Min, Toward a wearable biosensor ecosystem on ROS 2 for real-time human-robot interaction systems, CoRR abs/2110.03840 (2021). arXiv:2110.03840.
176. J.-L. Blanco-Claraco, B. Tymchenko, F. J. Mañas-Alvarez, F. Cañadas-Aránega, Ángel López-Gázquez, J. C. Moreno, Multivehicle simulator (mvsim): lightweight dynamics simulator for multiagents and mobile robotics research (2023). arXiv:2302.11033.
177. N. Pérez-Higueras, R. Otero, F. Caballero, L. Merino, Hunavsim: A ros 2 human navigation simulator for benchmarking human-aware robot navigation (2023). arXiv:2305.01303.
178. D. Pieczyński, B. Ptak, M. Kraft, P. Drapikowski, Lunarsim: Lunar rover simulator focused on high visual fidelity and ros 2 integration for advanced computer vision algorithm development, Applied Sciences 13 (22) (2023). doi:10.3390/app132212401.
179. M. Andreasen, P. Holler, M. Jensen, M. Albano, Maes: a ros 2-compatible simulation tool for exploration and coverage algorithms, Artificial Life and Robotics (2023).
180. M. M. M. Manhães, S. A. Scherer, M. Voss, L. R. Douat, T. Rauschenbach, Uuv simulator: A gazebo-based package for underwater intervention and multi-robot simulation, in: OCEANS 2016 MTS/IEEE Monterey, 2016, pp. 1–8. doi:10.1109/OCEANS.2016.7761080.
181. B. Benjdira, A. Koubaa, A. M. Ali, Rosgpt_vision: Commanding robots using only language models' prompts (2023). arXiv:2308.11236.
182. K. Koide, M. Yokozuka, S. Oishi, A. Banno, Glim: 3d range-inertial localization and mapping with gpu-accelerated scan matching factors, Robotics and Autonomous Systems 179 (2024) 104750. doi:https://doi.org/10.1016/j.robot.2024.104750.
183. E. G. A. Rolland, K. A. R. Grøntved, A. L. Christensen, M. Watson, T. Richardson, Autonomous uav volcanic plume sampling based on machine vision and path planning, in: 2024 International Conference on Unmanned Aircraft Systems (ICUAS), 2024, pp. 1064–1071. doi:10.1109/ICUAS60882.2024.10556912.
184. Z. Ge, S. Liu, F. Wang, Z. Li, J. Sun, Yolox: Exceeding yolo series in 2021, arXiv preprint arXiv:2107.08430 (2021).
185. K. Koide, S. Oishi, M. Yokozuka, A. Banno, General, single-shot, target-less, and automatic lidar-camera extrinsic calibration toolbox, in: 2023 IEEE International Conference on Robotics and Automation (ICRA), 2023, pp. 11301–11307. doi:10.1109/ICRA48891.2023.10160691.
186. L. Connolly, A. Deguet, S. Leonard, J. Tokuda, T. Ungi, A. Krieger, P. Kazanzides, P. Mousavi, G. Fichtinger, R. H. Taylor, Bridging 3d slicer and ros2 for image-guided robotic interventions, Sensors 22 (14) (2022). doi:10.3390/s22145336.
187. J. Li, B. Xu, S. Schwertfeger, High-quality, ros compatible video encoding and decoding for high-definition datasets (2024). arXiv:2408.00538.
188. D. P. Leal, M. Sugaya, H. Amano, T. Ohkawa, Fpga acceleration of ros2-based reinforcement learning agents, in: 2020 Eighth International Symposium on Computing and Networking Workshops (CANDARW), 2020, pp. 106–112. doi:10.1109/CANDARW51189.2020.00031.
189. N. G. Lopez, Y. L. E. Nuin, E. B. Moral, L. U. S. Juan, A. S. Rueda, V. M. Vilches, R. Kojcev, gym-gazebo2, a toolkit for reinforcement learning using ros 2 and gazebo, ArXiv abs/1903.06278 (2019).
190. A. Orsula, S. Bøgh, M. Olivares-Mendez, C. Martinez, Learning to Grasp on the Moon from 3D Octree Observations with Deep Reinforcement Learning, in: 2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2022, pp. 4112–4119. doi:10.1109/IROS47612.2022.9981661.
191. S. Agarwal, R. Muthukrishnan, W. Gosrich, V. Kumar, A. Ribeiro, Lpac: Learnable perception-action-communication loops with applications to coverage control (2024). arXiv:2401.04855.
192. Y. L. E. Nuin, N. G. Lopez, E. B. Moral, L. U. S. Juan, A. S. Rueda, V. M. Vilches, R. Kojcev, Ros2learn: a reinforcement learning framework for ros 2 (2019). doi:10.48550/ARXIV.1903.06282.

 doi:10.20944/preprints202410.1204.v3

193. F. Almeida, G. Leão, L. Sousa, Armando editor=Marques, C. Santos, J. L. Lima, D. Tardioli, M. Ferre, An educational kit for simulated robot learning in ros 2, in: Robot 2023: Sixth Iberian Robotics Conference, Springer Nature Switzerland, Cham, 2024, pp. 513–525.
194. F. Abawi, P. Allgeuer, D. Fu, S. Wermter, Wrapyfi: A python wrapper for integrating robots, sensors, and applications across multiple middleware, in: Proceedings of the 2024 ACM/IEEE International Conference on Human-Robot Interaction, HRI '24, Association for Computing Machinery, New York, NY, USA, 2024, p. 860–864. doi:10.1145/3610977.3637471.
URL https://doi.org/10.1145/3610977.3637471
195. B. Peng, A. Hasegawa, T. Azumi, Scheduling performance evaluation framework for ros 2 applications, in: 2022 IEEE 24th Int Conf on High Performance Computing & Communications; 8th Int Conf on Data Science & Systems; 20th Int Conf on Smart City; 8th Int Conf on Dependability in Sensor, Cloud & Big Data Systems & Application (HPCC/DSS/SmartCity/DependSys), 2022, pp. 2031–2038.
196. M.-Y. Shen, C.-C. Hsu, H.-Y. Hou, Y.-C. Huang, W.-F. Sun, C.-C. Chang, Y.-L. Liu, C.-Y. Lee, Driveenv-nerf: Exploration of a nerf-based autonomous driving environment for real-world performance validation (2024). arXiv:2403.15791.
197. V. Mayoral-Vilches, J. Jabbour, Y.-S. Hsiao, Z. Wan, M. Crespo-Álvarez, M. Stewart, J. M. Reina-Muñoz, P. Nagras, G. Vikhe, M. Bakhshalipour, M. Pinzger, S. Rass, S. Panigrahi, G. Corradi, N. Roy, P. B. Gibbons, S. M. Neuman, B. Plancher, V. J. Reddi, Robotperf: An open-source, vendor-agnostic, benchmarking suite for evaluating robotics computing system performance (2024). arXiv:2309.09212.
198. T. Kuboichi, A. Hasegawa, B. Peng, K. Miura, K. Funaoka, S. Kato, T. Azumi, Caret: Chain-aware ros 2 evaluation tool, in: 2022 IEEE 20th International Conference on Embedded and Ubiquitous Computing (EUC), 2022, pp. 1–8. doi:10.1109/EUC57774.2022.00010.
199. M. Moore, J. Sooknanan, J. Holley, Bobble-bot: An educational platform for real-time control with ros, in: 2019 IEEE International Symposium on Measurement and Control in Robotics (ISMCR), 2019, pp. B3–2–1–B3–2–7. doi:10.1109/ISMCR47492.2019.8955713.
200. S. Salimi, J. P. Queralta, T. Westerlund, Hyperledger fabric blockchain and ros 2 integration for autonomous mobile robots, in: 2023 IEEE/SICE International Symposium on System Integration (SII), 2023, pp. 1–8. doi:10.1109/SII55687.2023.10039326.
201. V. M. Vilches, L. U. S. Juan, B. Dieber, U. A. Carbajo, E. Gil-Uriarte, Introducing the robot vulnerability database (rvd), arXiv preprint arXiv:1912.11299 (2019).
202. I. Vizzo, T. Guadagnino, B. Mersch, L. Wiesmann, J. Behley, C. Stachniss, KISS-ICP: In Defense of Point-to-Point ICP – Simple, Accurate, and Robust Registration If Done the Right Way, IEEE Robotics and Automation Letters (RA-L) 8 (2) (2023) 1029–1036. doi:10.1109/LRA.2023.3236571.
203. G. O. Mendia, L. U. S. Juan, X. P. Bascaran, A. B. Calvo, A. H. Cordero, I. Z. Ugarte, A. M. Rosas, D. M. Vilches, U. A. Carbajo, L. A. Kirschgens, V. M. Vilches, E. Gil-Uriarte, Robotics ctf (rctf), a playground for robot hacking (2021). arXiv:1810.02690.
204. V. M. Vilches, R. White, G. Caiazza, M. Arguedas, Sros2: Usable cyber security tools for ros 2 (2022). doi:10.48550/ARXIV.2208.02615.
205. V. M. Vilches, G. O. Mendia, X. P. Baskaran, A. H. Cordero, L. U. S. Juan, E. Gil-Uriarte, O. O. S. de Urabain, L. A. Kirschgens, Aztarna, a footprinting tool for robots, arXiv e-prints (2018) arXiv:1812.09490arXiv:1812.09490.
206. C. Morales, C. Fuenzalida, G. Sierra, Cfv2: an open-source robot controller board for education and research, in: 2023 IEEE CHILEAN Conference on Electrical, Electronics Engineering, Information and Communication Technologies (CHILECON), 2023, pp. 1–6. doi:10.1109/CHILECON60335.2023.10418618.
207. M. Mayr, F. Rovida, V. Krueger, Skiros2: A skill-based robot control platform for ros, in: 2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2023, pp. 6273–6280. doi:10.1109/IROS55552.2023.10342216.
208. W. Jo, J. Kim, R. Wang, J. Pan, R. K. Senthilkumaran, B.-C. Min, Smartmbot: A ros2-based low-cost and open-source mobile robot platform (2022). doi:10.48550/ARXIV.2203.08903.
209. G. Szabó, Toward the automatic network resource management of robot operating system in programmable mobile networks, IEEE Access 11 (2023) 65934–65955. doi:10.1109/ACCESS.2023.3289922.
210. P. Karle, F. Török, M. Geisslinger, M. Lienkamp, Mixnet: Physics constrained deep neural motion prediction for autonomous racing, IEEE Access 11 (2023) 85914–85926. doi:10.1109/ACCESS.2023.3303841.
211. P. Karle, F. Fent, S. Huch, F. Sauerbeck, M. Lienkamp, Multi-modal sensor fusion and object tracking for autonomous racing, IEEE Transactions on Intelligent Vehicles (2023) 1–13doi:10.1109/TIV.2023.3271624.

 doi:10.20944/preprints202410.1204.v3

212. V. Walker, F. Vanegas, F. Gonzalez, Nanomap: A gpu-accelerated openvdb-based mapping and simulation package for robotic agents, Remote Sensing 14 (21) (2022). doi:10.3390/rs14215463.
213. E. Horvath, C. Pozna, P. Koros, C. Hajdu, A. Ballagi, Theoretical background and application of multiple goal pursuit trajectory follower, Hungarian Journal of Industry and Chemistry 48 (1) (2020) 11–17. doi:10.33927/hjic-2020-03.
214. O. Nacar, M. Abdelkader, L. Ghouti, K. Gabr, A. Al-Batati, A. Koubaa, Vector: Velocity-enhanced gru neural network for real-time 3d uav trajectory prediction, Drones 9 (1) (2025). doi:10.3390/drones9010008.
URL https://www.mdpi.com/2504-446X/9/1/8
215. M. Abdelkader, K. Gabr, I. Jarraya, A. AlMusalami, A. Koubaa, Smarttrack: A novel kalman filter-guided sensor fusion for robust uav object tracking in dynamic environments, IEEE Sensors Journal 25 (2) (2025) 3086–3097. doi:10.1109/JSEN.2024.3505939.
216. S. Macenski, M. Booker, J. Wallace, Open-source, cost-aware kinematically feasible planning for mobile and surface robotics (2024). arXiv:2401.13078.
217. F. Martín, J. G. Clavero, V. Matellan, F. J. Rodriguez, Plansys2: A planning system framework for ros2, in: 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2021, pp. 9742–9749. doi:10.1109/IROS51168.2021.9636544.
218. M. Á. Gonzalez-Santamarta, F. J. Rodriguez-Lera, V. M. Olivera, Sailor: Perceptual anchoring for robotic cognitive architectures (2023). arXiv:2303.08204.
219. M. A. GonzalezSantamarta, F. J. RodriguezLera, V. MatellanOlivera, C. FernandezLlamas, Yasmin: Yet another state machine, in: D. Tardioli, V. Matellan, G. Heredia, M. F. Silva, L. Marques (Eds.), ROBOT2022: Fifth Iberian Robotics Conference, Springer International Publishing, Cham, 2023, pp. 528–539.
220. J. L. Blanco-Claraco, A flexible framework for accurate lidar odometry, map manipulation, and localization (2024). arXiv:2407.20465.
221. M. Drwięga, J. Jakubiak, A set of depth sensor processing ROS tools for wheeled mobile robot navigation, Journal of Automation, Mobile Robotics & Intelligent Systems (JAMRIS) (2017). doi:10.14313/JAMRIS_2-2017/16.
222. L. Fernández-Becerra, M. A. González-Santamarta, D. Sobrín-Hidalgo, Á. M. Guerrero-Higueras, F. J. R. Lera, P. Olivera, Vicente Matellán editor=García Bringas, H. Pérez García, F. J. Martínez de Pisón, F. Martínez Álvarez, A. Troncoso Lora, Á. Herrero, J. L. Calvo Rolle, H. Quintián, E. Corchado, Accountability and explainability in robotics: A proof of concept for ros 2- and nav2-based mobile robots, in: International Joint Conference 16th International Conference on Computational Intelligence in Security for Information Systems (CISIS 2023) 14th International Conference on EUropean Transnational Education (ICEUTE 2023), Springer Nature Switzerland, Cham, 2023, pp. 3–13.
223. J. Galvis, D. Pediaditis, K. S. Almazrouei, N. Aspragathos, An autonomous navigation approach based on bird's-eye view semantic maps, in: 2023 27th International Conference on Methods and Models in Automation and Robotics (MMAR), IEEE, 2023, pp. 81–86.
224. H. V. Do, Y. H. Kim, J. H. Lee, M. H. Lee, J. W. Song, Dero: Dead reckoning based on radar odometry with accelerometers aided for robot localization, in: 2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)., to be published, 2024, pp. 8547–8554.
225. F. Atas, G. Cielniak, L. Grimstad, Elevation state-space: Surfel-based navigation in uneven environments for mobile robots, in: 2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2022, pp. 5715–5721. doi:10.1109/IROS47612.2022.9981647.
226. F. Atas, L. Grimstad, G. Cielniak, Evaluation of sampling-based optimizing planners for outdoor robot navigation, CoRR abs/2103.13666 (2021). arXiv:2103.13666.
227. M. Grupp, evo: Python package for the evaluation of odometry and slam., https://github.com/MichaelGrupp/evo (2017).
228. M. Leitenstern, F. Sauerbeck, D. Kulmer, J. Betz, Flexmap fusion: Georeferencing and automated conflation of hd maps with openstreetmap (2024). arXiv:2404.10879.
229. G. Lvov, M. Zolotas, N. Hanson, A. Allison, X. Hubbard, M. Carvajal, T. Padir, Mobile mocap: Retroreflector localization on-the-go (2023). arXiv:2303.13681.
230. F. Martin, J. M. Guerrero, A. A. Garcia, F. Rodriguez, V. Matellan, Mocap4ros2: An open source framework for motion capture systems in robotics (2022). doi:10.1145/3555051.3555076.
URL https://doi.org/10.1145/3555051.3555076

 doi:10.20944/preprints202410.1204.v3

231. A. Serov, J. Clemens, K. Schill, Multi-robot graph slam using lidar, in: 2024 10th International Conference on Automation, Robotics and Applications (ICARA), 2024, pp. 339–346. doi:10.1109/ICARA60736.2024.1055 3070.
232. T. Goelles, B. Schlager, S. Muckenhuber, S. Haas, T. Hammer, 'pointcloudset': Efficient analysis of large datasets of point clouds recorded over time, Journal of Open Source Software 6 (65) (2021) 3471. doi: 10.21105/joss.03471.
233. J. M. Zutell, D. C. Conner, P. Schillinger, Ros 2-based flexible behavior engine for flexible navigation, in: SoutheastCon 2022, 2022, pp. 674–681. doi:10.1109/SoutheastCon48659.2022.9764047.
234. A. Kampmann, A. Wüstenberg, B. Alrifaee, S. Kowalewski, A portable implementation of the real-time publish-subscribe protocol for microcontrollers in distributed robotic applications, in: 2019 IEEE Intelligent Transportation Systems Conference (ITSC), 2019, pp. 443–448. doi:10.1109/ITSC.2019.8916835.
235. D. P. Leal, M. Sugaya, H. Amano, T. Ohkawa, Automated integration of high-level synthesis fpga modules with ros2 systems, in: 2020 International Conference on Field-Programmable Technology (ICFPT), 2020, pp. 292–293. doi:10.1109/ICFPT51103.2020.00052.
236. C. Lienen, M. Platzner, Design of distributed reconfigurable robotics systems with reconros, ACM Trans. Reconfigurable Technol. Syst. 15 (3) (dec 2022). doi:10.1145/3494571.
237. C. Lienen, M. Platzner, B. Rinner, Reconros: Flexible hardware acceleration for ros2 applications, in: 2020 International Conference on Field-Programmable Technology (ICFPT), 2020, pp. 268–276. doi:10.1109/ ICFPT51103.2020.00046.
238. K. Chen, J. Yuan, N. Jha, J. Ichnowski, J. Kubiatowicz, K. Goldberg, Fogros g: Enabling secure, connected and mobile fog robotics with global addressability (2022). arXiv:2210.11691.
239. J. Ichnowski, K. Chen, K. Dharmarajan, S. Adebola, M. Danielczuk, V. Mayoral-Vilches, N. Jha, H. Zhan, E. Llontop, D. Xu, C. Buscaron, J. Kubiatowicz, I. Stoica, J. Gonzalez, K. Goldberg, Fogros2: An adaptive platform for cloud and fog robotics using ros 2, in: 2023 IEEE International Conference on Robotics and Automation (ICRA), 2023, pp. 5493–5500. doi:10.1109/ICRA48891.2023.10161307.
240. K. Chen, R. Hoque, K. Dharmarajan, E. LLontop, S. Adebola, J. Ichnowski, J. Kubiatowicz, K. Goldberg, Fogros2-sgc: A ros2 cloud robotics platform for secure global connectivity (2023). arXiv:2306.17157.
241. J. Ichnowski, K. Chen, K. Dharmarajan, S. Adebola, M. Danielczuk, V. Mayoral-Vilches, N. Jha, H. Zhan, E. LLontop, D. Xu, J. Kubiatowicz, I. Stoica, J. Gonzalez, K. Goldberg, Fogros2: An adaptive platform for cloud and fog robotics using ros 2 (2022). doi:10.48550/ARXIV.2205.09778.
242. V. Mayoral-Vilches, S. M. Neuman, B. Plancher, V. J. Reddi, Robotcore: An open architecture for hardware acceleration in ros 2, in: 2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2022, pp. 9692–9699. doi:10.1109/IROS47612.2022.9982082.
243. A. Sarabakha, anafi_ros: from off-the-shelf drones to research platforms (2023). arXiv:2303.01813.
244. N. D. W. Mudalige, I. Zhura, I. Babataev, E. Nazarova, A. Fedoseev, D. Tsetserukou, Hyperdog: An open-source quadruped robot platform based on ros2 and micro-ros (2022). doi:10.48550/ARXIV.2209.09171.
245. F. F. Nyboe, N. H. Malle, E. Ebeid, Mpsoc4drones: An open framework for ros2, px4, and fpga integration, in: 2022 International Conference on Unmanned Aircraft Systems (ICUAS), 2022, pp. 1246–1255. doi: 10.1109/ICUAS54217.2022.9836055.
246. M. Abdelkader, K. Gabr, uav_dataset_generation_ros (Jan. 2024).
URL https://github.com/mzahana/uav_dataset_generation_ros
247. G. R. Silva, J. Päßler, J. Zwanepol, E. Alberts, S. L. T. Tarifa, I. Gerostathopoulos, E. B. Johnsen, C. H. Corbato, Suave: An exemplar for self-adaptive underwater vehicles, in: 2023 IEEE/ACM 18th Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), 2023, pp. 181–187. doi: 10.1109/SEAMS59076.2023.00031.
248. E. Palmer, C. Holm, G. Hollinger, Angler: An Autonomy Framework for Intervention Tasks with Lightweight Underwater Vehicle Manipulator Systems, in: IEEE International Conference on Robotics and Automation, 2024, pp. 6126–6132.
249. Z. Li, A. Hasegawa, T. Azumi, Autoware_perf: A tracing and performance analysis framework for ros 2 applications, Journal of Systems Architecture 123 (2022) 102341. doi:https://doi.org/10.1016/j.sysarc. 2021.102341.
250. S. Sivashangaran, A. Eskandarian, Xtenth-car: A proportionally scaled experimental vehicle platform for connected autonomy and all-terrain research (2022). arXiv:2212.01691.
251. M. A. GonzalezSantamarta, F. J. RodriguezLera, C. Alvarez-Aparicio, A. M. GuerreroHigueras, C. Fernandez-Llamas, Merlin a cognitive architecture for service robots, Applied Sciences 10 (17) (2020) 5989.

 doi:10.20944/preprints202410.1204.v3

252. M. Á. Gonzalez-Santmarta, F. J. Rodriguez-Lera, C. Fernandez-Llamas, V. Matellan-Olivera, Merlin2: Machined ros 2 planing, Software Impacts 15 (2023) 100477. doi:https://doi.org/10.1016/j.simpa.2023.100477.
253. A. Serrano-Muñoz, Í. Elguea-Aguinaco, D. Chrysostomou, S. Bøgh, N. Arana-Arexolaleiba, A scalable and unified multi-control framework for kuka lbr iiwa collaborative robots, in: 2023 IEEE/SICE International Symposium on System Integration (SII), IEEE, 2023, pp. 1–5.
254. I. Zamalloa, I. Muguruza, A. Hernández, R. Kojcev, V. Mayoral, An information model for modular robots: the hardware robot information model (hrim), arXiv preprint arXiv:1802.01459 (2018).
255. C. Heggem, N. M. Wahl, L. Tingelstad, Configuration and control of kmr iiwa mobile robots using ros2, in: 2020 3rd International Symposium on Small-scale Intelligent Manufacturing Systems (SIMS), 2020, pp. 1–6. doi:10.1109/SIMS49386.2020.9121554.
256. M. Huber, C. E. Mower, S. Ourselin, T. Vercauteren, C. Bergeles, Lbr-stack: Ros 2 and python integration of kuka fri for med and iiwa robots (2023). arXiv:2311.12709.
257. T. Winiarski, Meros: Sysml-based metamodel for ros-based systems (2023). arXiv:2303.08254.
258. T. M. C. Sears, M. R. Cooper, S. R. Button, J. A. Marshall, Otterros: Picking and programming an uncrewed surface vessel for experimental field robotics research with ros 2 (2024). arXiv:2404.05627.
259. A. R. Mosteo, Rclada, or bringing ada to the robot operating system, Ada Lett. 39 (2) (2020) 35–40. doi:10.1145/3394514.3394518.
260. S. Kim, T. Kim, Robofuzz: fuzzing robotic systems over robot operating system (ros) for finding correctness bugs, in: Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2022, Association for Computing Machinery, New York, NY, USA, 2022, p. 447–458. doi:10.1145/3540250.3549164.
261. A. Merzlyakov, S. Macenski, A comparison of modern general-purpose visual slam approaches, in: 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2021, pp. 9190–9197. doi:10.1109/IROS51168.2021.9636615.
262. Z. Guo, A. Lykov, Z. Yagudin, M. Konenkov, D. Tsetserukou, Co-driver: Vlm-based autonomous driving assistant with human-like behavior and understanding for complex road scenes (2024). arXiv:2405.05885.
263. F. Cañadas-Aránega, J. L. Blanco-Claraco, J. C. Moreno, F. Rodriguez-Diaz, Multimodal mobile robotic dataset for a typical mediterranean greenhouse: The greenbot dataset, Sensors 24 (6) (2024). doi:10.3390/s24061874.
264. P.-Y. Lajoie, G. Beltrame, Swarm-slam : Sparse decentralized collaborative simultaneous localization and mapping framework for multi-robot systems (2023). arXiv:2301.06230.
265. T. Puccetti, S. Nardi, C. Cinquilli, T. Zoppi, A. Ceccarelli, Rospace: Intrusion detection dataset for a ros2-based cyber-physical system and iot networks, Scientific Data 11 (1) (2024) 481. doi:10.1038/s41597-024-03311-2.
266. M. L. Anjum, O. Ahmad, S. Rosa, J. Yin, B. Bona, Skeleton tracking based complex human activity recognition using kinect camera, in: Social Robotics: 6th International Conference, ICSR 2014, Sydney, NSW, Australia, October 27-29, 2014. Proceedings 6, Springer, 2014, pp. 23–33.
267. C. Grigioni, F. Corradini, A. Antonucci, J. Guzzi, F. Flammini, Safe road-crossing by autonomous wheelchairs: a novel dataset and its experimental evaluation (2024). arXiv:2403.08984.
268. S. B. Rosende, D. S. J. Gavilán, J. Fernandez-Andrés, J. Sánchez-Soriano, An urban traffic dataset composed of visible images and their semantic segmentation generated by the carla simulator, Data 9 (1) (2024). doi:10.3390/data9010004.
269. P.-Y. Lajoie, G. Beltrame, Swarm-slam : Sparse decentralized collaborative simultaneous localization and mapping framework for multi-robot systems (2023). arXiv:2301.06230.