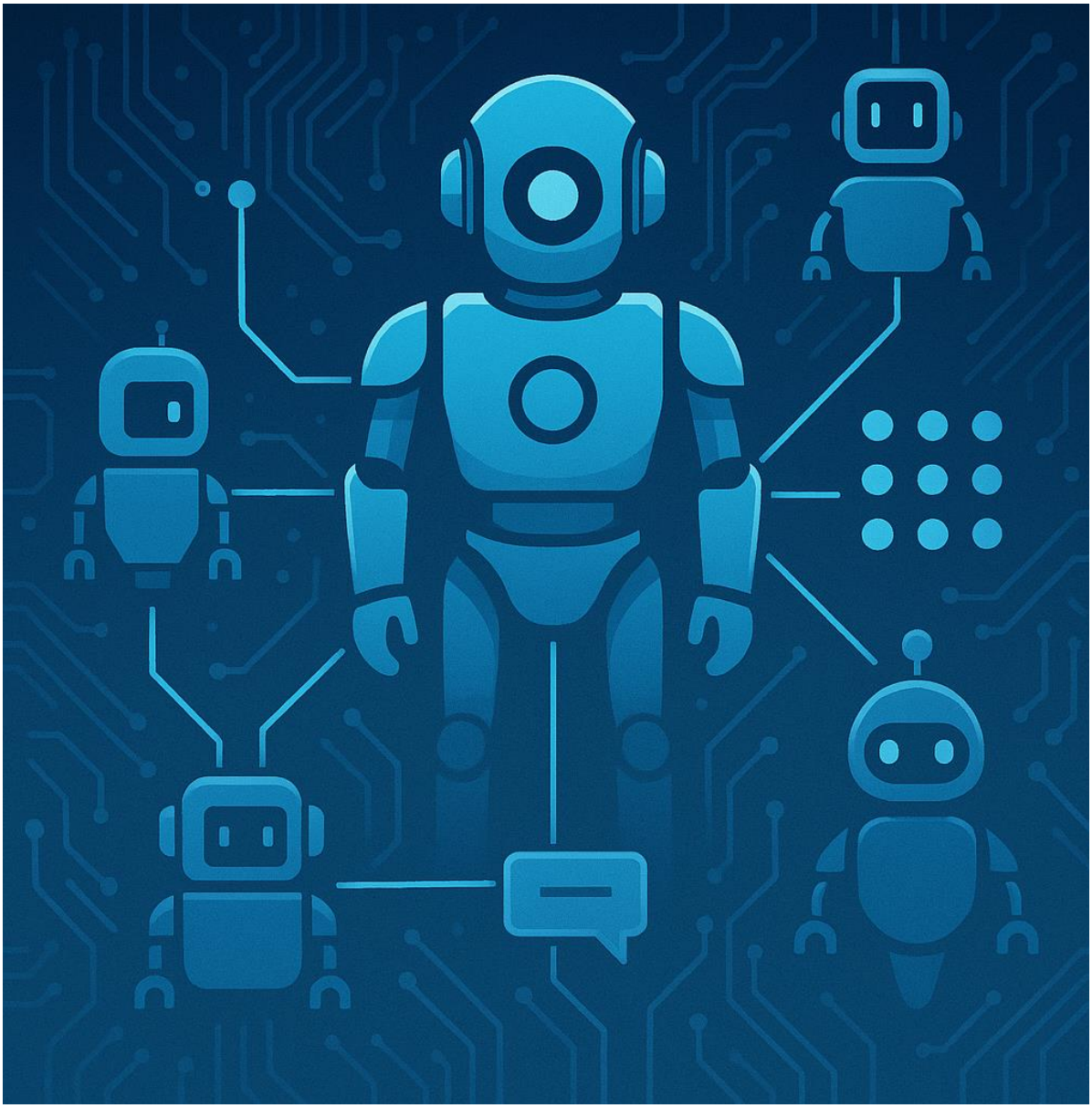
CSIC6843 – Lecture Unit 3

Robotic Operating Systems (ROS) & Software Frameworks



Robotic Operating Systems: Foundation to Future Trends

Table of Contents

Unit 3: Robotic Operating Systems (ROS) & Software Frameworks

Introduction

Robotic Operating Systems (ROS) are modular, open-source middleware platforms providing essential communication, hardware abstraction, and tooling for robot software development. Unlike classical operating systems, ROS enables developers to focus on high-level robotic tasks by offering infrastructure for distributed systems, communication, and package management.

Initially developed as ROS 1, ROS 2 emerged specifically to address critical requirements such as real-time performance, embedded systems compatibility, and industrial applicability, areas where ROS 1 had limitations. Clarifying these distinctions early sets the stage for a deeper exploration into their architectures and use cases.

This unit deepens your understanding of the evolution, architecture, ecosystem, and industrial relevance of ROS and complementary robotics software frameworks. Through detailed conceptual analysis and industry case studies, you will explore how ROS fits into modern robotics software systems without requiring hands-on coding or system setup.

Prerequisite Knowledge
Before proceeding, it is helpful to have familiarity with:

- Basic Linux system concepts (filesystem hierarchy, user permissions, shell basics)
- Programming fundamentals in Python or C++ (conceptual understanding)
- Robotics basics: sensors, actuators, control principles
- Software engineering concepts: version control, continuous integration (conceptual)

Chapter Learning Objectives
After completing this chapter, you will be able to:

- Explain why ROS is preferred over ad-hoc robot control software for coordinating complex systems.
- Describe core ROS architectural components and their interactions in a distributed robot system.
- Compare ROS 1 and ROS 2 communication paradigms, focusing on middleware abstraction and real-time capabilities.
- Analyse the software toolchain and ecosystem components that support ROS development.
- Discuss industrial application cases highlighting ROS's role and challenges in deployment.
- Understand safety, security, and certification considerations relevant to ROS-based systems.
- Evaluate alternative robotics software frameworks and emerging trends in robotics middleware.

General Information

3.1. Foundation Concepts: Why ROS Matters

3.1.1 The Software Challenge in Robotics

Modern robots integrate multiple subsystems — sensing, perception, planning, and actuation — that must operate in concert. Traditional approaches relying on monolithic, tightly coupled control software often become unmanageable and brittle, presenting significant hurdles for scaling or adapting to new hardware and capabilities.

ROS addresses this challenge by offering a modular, message-oriented middleware architecture. It enables subsystems, implemented as independent processes called **nodes**, to communicate asynchronously through **topics** (publish-subscribe) and synchronously via **services** and **actions**. This decoupled design increases software reusability, flexibility, and maintainability.

3.1.2 Industrial Motivations and Use Cases

- **BMW Autonomous Mobile Robots** employ ROS 2 to coordinate fleets delivering parts with deterministic Quality of Service (QoS). This supports reliable, real-time communication essential for safe factory automation.
- **Amazon Robotics** uses ROS-based middleware to synchronize thousands of warehouse robots, substantially optimising order-picking throughput via distributed software coordination.

3.1.3 Comparing ROS Middleware with Traditional Control Systems

Traditional Control	ROS Middleware Approach	Benefits of ROS
Monolithic, vendor-specific software	Modular, language-agnostic nodes	Scalability, portability, extensibility
Custom messaging protocols	Standardized DDS-based communication (ROS 2)	Interoperability, real-time QoS
Fragile and difficult to reuse	Open ecosystem with reusable packages	Accelerated development and talent availability

3.1.4 Conceptual Linux/Operating System Foundations

ROS development primarily targets Linux environments (commonly Ubuntu) due to several factors:

- **Filesystem layout:** Organized directories like */opt*, */usr*, and */home* hold system software, libraries, and user data, respectively.
- **User Permissions:** Controlled access to hardware devices (e.g., USB, serial ports) is enforced for security and stability.
- **Shell utilities:** Command-line interface enables process management and scripting critical for managing distributed ROS nodes.

3.2. Core Concepts: ROS Architectural Building Blocks

3.2.1 Nodes

A **node** is an independent process encapsulating a distinct robotic function, such as sensor data acquisition or motion control. Nodes operate concurrently and communicate via well-defined messaging interfaces. For example, a sensor node captures and publishes lidar scans, while a control node subscribes to sensor data and computes actuator commands.

3.2.2 Topics

Topics implement an asynchronous publish/subscribe pattern, allowing nodes to exchange unidirectional data streams anonymously. For instance, a node might publish lidar scans on the topic /scan, and another node could subscribe to */cmd_vel* to receive velocity commands.

3.2.3 Services

Services facilitate synchronous request-response interactions suitable for sporadic commands requiring immediate feedback. An example service would be */reset_odometry*, which, when called, immediately resets the robot's odometry data and returns a confirmation response.

3.2.4 Actions

Actions extend services by supporting long-duration, preemptible tasks that provide continuous feedback. An example is an action named */navigate_to_pose*, where the robot moves toward a specified target position while continuously providing progress updates and allowing cancellation or pre-emption.

3.2.5 Parameter Server and Configuration

ROS employs a centralized parameter server (ROS 1) or a runtime parameter API (ROS 2) to allow dynamic reconfiguration of nodes' behaviours without restarting processes, thereby improving flexibility. For instance, a node controlling robot speed could dynamically update a parameter such as *max_velocity* at runtime.

3.2.6 Middleware Abstraction via RMW

ROS 2 introduces the **Robotics Middleware Wrapper (RMW)** layer, abstracting the underlying data transport mechanism. The default middleware is DDS (Data Distribution Service), offering real-time guarantees, vendor neutrality, and extensive Quality-of-Service (QoS) controls.

3.2.7 Quality of Service (QoS) Profiles

QoS parameters ensure communication reliability, latency constraints, message durability, and deadline enforcement tailored to different robotic applications. For example, sensor data typically uses a "best-effort" QoS setting, whereas critical control commands may use "reliable" QoS for guaranteed message delivery.

3.2.8 Lifecycle Nodes

Lifecycle management introduces explicit node states—from unconfigured through active to shutdown—that support safe transitions, fault recovery, and integration with industrial functional safety standards. An example lifecycle node might transition explicitly from an "inactive" state to an "active" state only after successfully completing system diagnostics.

3.3. ROS Toolchain and Ecosystem Overview

3.3.1 Build and Package Management

ROS 1 uses **catkin** build system, while ROS 2 adopts **colcon**, enabling parallel package building and selective compilation.

3.3.2 Visualization and Simulation Tools

- **rviz2** and **rqt** enable 3D visualization of robot states and introspection of node interactions.
- **Gazebo Harmonic** offers mature, reliable physics-based simulation ideal for general robotic development and validation tasks. In contrast, **Ignition** (now known as Gazebo) provides advanced capabilities, modular architecture, and enhanced performance suited for more complex scenarios or projects requiring scalability and flexibility, such as large-scale digital twin simulations.

3.3.3 Data Recording and Replay

rosbag2 facilitates capturing, storing, and replaying message streams for debugging and analysis without physical hardware.

3.3.4 Command-Line Interface (CLI)

The unified ros2 CLI tool allows process orchestration, diagnostics, and interaction with the ROS graph in a streamlined manner.

3.3.5 Industrial and Vertical Packages

Specific packages support advanced capabilities:

- MoveIt 2 for manipulation
- Navigation 2 for autonomous mobility
- micro-ROS for resource-constrained, embedded applications
- ROS-Industrial drivers for integration with major robot manufacturers (ABB, Fanuc, KUKA)

3.3.6 Community and Package Ecosystem

The ROS package index hosts over 26,000 packages maintained by a vibrant community, governed openly via Technical Steering Committees (TSCs) and Special Interest Groups (SIGs).

3.4. Simulation, Digital Twins, and Hardware-in-the-Loop (HIL)

Simulation frameworks allow developers to test and validate robot software virtually:

- **Gazebo Harmonic** offers sensor-rich, physics-faithful environments with support for multiple physics engines.
- **Digital twins** synchronize virtual models with real robotic systems to predict behaviour and diagnose issues.
- **Hardware-in-the-loop setups** blend real hardware and simulation for hybrid testing, crucial for safety-critical applications.

3.5. Embedded, Edge, and Hard Real-Time Robotics

3.5.1 micro-ROS

micro-ROS is a tailored subset of ROS 2 specifically designed for microcontrollers (<512kB flash). It enables constrained devices to participate in ROS graphs by leveraging efficient communication transports (serial/UDP) and real-time executors. micro-ROS typically integrates closely with real-time operating systems, such as FreeRTOS or Zephyr, ensuring predictable and deterministic behaviour critical for embedded robotic applications.

3.5.2 Edge Computing and FogROS 2

Frameworks like FogROS 2 extend ROS beyond individual devices into cloudlets and edge servers, facilitating distributed computation, teleoperation, and resilience against network variability.

3.5.3 Hard Real-Time Control

In conjunction with tools like Orocos RTT and ros2_control interfaces, ROS can be integrated into hard real-time control loops offering latency guarantees (sub-50 µs), essential for industrial robotics applications.

3.6. Alternative and Complementary Robotics Software Frameworks

Framework	Domain	Key Features
MOOS-IvP	Autonomous Underwater Vehicles	Mailbox messaging, behaviour autonomy. Preferred over ROS for simpler integration and optimized behaviour autonomy in resource-constrained underwater vehicles.
YARP	Humanoid Robotics	Device abstraction, plugin-based drivers. Preferred when flexible hardware abstraction and modularity are paramount, especially in complex humanoid robotic systems.
LCM	Real-time Perception	UDP multicast, log playback integration. Chosen over ROS when low-latency, lightweight multicast communication is critical, such as in high-speed perception systems.
Orocos RTT	Industrial Hard Real-Time	Component dataflow, deterministic scheduling. Ideal for strict real-time industrial applications requiring deterministic task execution beyond ROS capabilities.
Microsoft ROS for Windows	Robotics on Windows IoT	Visual Studio Integration, WinML support. Selected specifically when full Windows IoT ecosystem integration or direct support for Windows-based ML workflows is desired.

Bridges and interoperability solutions support mixed-framework deployments.

3.7. Certification, Safety, and Security in ROS Systems

3.7.1 Functional Safety

ROS 2 is increasingly aligned with standards such as:

- ISO 26262 (automotive safety)
- IEC 61508 (industrial safety)
- ISO 10218 and 15066 (robotics and cobots)

Achieving compliance with these standards—particularly through certified safety frameworks such as Apex.OS—uniquely positions ROS 2 to meet stringent safety and certification requirements (up to the required Safety Integrity Levels, SIL). As a result, ROS 2 becomes particularly attractive for adoption in safety-critical industries such as autonomous vehicles and industrial robotics, where adherence to these standards is a crucial differentiator.

3.7.2 Cybersecurity Architecture

- DDS-Security integration enforces mutual authentication, encryption, and access control.
- SROS 2 offers tooling for keystore management and node isolation policies to mitigate privilege escalation and unauthorized access.

3.7.3 Safety Patterns

Emergency-stop protocols, fault detection, and graceful degradation mechanisms are essential for deployment in safety-critical environments.

3.8. Community Governance and Market Insights

ROS governance is transparent and inclusive, organized around TSC and SIGs focusing on feature areas like Security, Navigation, and Hardware Acceleration.

The ecosystem is maturing into a $1.2B market by 2033, with widespread adoption in manufacturing, logistics, healthcare robotics, and education.

Annual events such as ROSCon promote knowledge sharing, standardization, and collaborative development.

3.9. Emerging Trends and Future Outlook

Several emerging technologies have the potential to significantly shape the future of ROS, each with growing industry interest and practical relevance:

- WebAssembly Runtimes: Lightweight WebAssembly runtimes allow ROS nodes to execute directly in browsers or embedded environments. This simplifies deployment and enables cross-platform compatibility, attracting interest for rapid prototyping and flexible application deployment.
- GraphQL Integration: Integrating GraphQL facilitates sophisticated querying of ROS graph state, streamlining the development of complex, responsive ROS-based applications. Its adoption is increasing as developers seek efficient, flexible APIs for real-time robot monitoring and control.
- Heterogeneous Hardware Acceleration: Growing support for FPGA and GPU acceleration significantly boosts ROS performance, particularly for computation-intensive tasks like perception and machine learning. FPGA-based solutions are especially appealing in real-time systems where low latency and high efficiency are critical.
- Mixed-Reality Tooling: Combining ROS frame transforms with augmented reality tools provides intuitive interfaces for programming and debugging robots. Practical interest is evident in industries such as manufacturing and logistics, where rapid development and visualization of robotic workflows are valuable.
- Minimal Risk Conditions (MRC): Industry-led standardization efforts to define MRCs are critical for autonomous mobile robots' safety. These standards help ensure reliability, driving regulatory acceptance and broader deployment in safety-critical applications.

Reflective Questions

- Why does middleware-centric design outperform monolithic robot control architectures?
- How does DDS middleware underpin ROS 2's real-time communication guarantees?
- What roles do lifecycle nodes and safety profiles play in industrial ROS deployments?
- Compare ROS to MOOS or YARP in terms of design philosophy and application focus.
- In what ways does the ROS community governance model foster quality and innovation?
- Given a specific robotic application scenario, how would you determine whether to use ROS 1 or ROS 2, and what factors would influence your decision?

Summary

This unit provided a study of Robotic Operating Systems (ROS), emphasizing their importance as flexible middleware solutions that overcome the limitations of traditional monolithic control architectures. It introduced the evolution from ROS 1 to ROS 2, highlighting significant improvements such as real-time capabilities, middleware abstraction, and suitability for embedded and industrial applications.

Core architectural components—including nodes, topics, services, and actions—were described with practical examples to illustrate their roles within distributed robotic systems. The unit examined critical ecosystem tools, such as visualization with rviz2, simulation with Gazebo variants, and data management with rosbag2, demonstrating how they support robust ROS development workflows.

A detailed discussion covered embedded robotics through micro-ROS, edge computing with FogROS 2, and real-time control strategies via Orocos RTT, emphasizing their applicability in resource-constrained and latency-sensitive environments. The content also compared alternative frameworks like MOOS-IvP, YARP, and LCM, clarifying scenarios where they might be preferred.

Furthermore, the unit addressed essential considerations for deploying ROS in safety-critical industries, highlighting functional safety standards compliance (ISO 26262, IEC 61508) and cybersecurity strategies through DDS-Security and SROS 2.

Finally, emerging trends such as WebAssembly runtimes, GraphQL integration, FPGA/GPU acceleration, mixed-reality tools, and standardization of minimal risk conditions were discussed, underscoring their potential impact on future robotics applications. This comprehensive understanding positions students effectively for advanced study and professional engagement with modern robotic middleware.

Glossary of Key Terms

Term	**Description**
Actions	An architectural building block in ROS that supports long-duration, preemptible tasks requiring continuous feedback, extending the capabilities of Services.
catkin	The build system primarily used in ROS 1 for managing and compiling packages.
colcon	The modern build system adopted by ROS 2, offering improvements like parallel package building and selective compilation over catkin.
Command-Line Interface (CLI)	A text-based interface used for interacting with and managing ROS processes, diagnostics, and the ROS graph.
Data Distribution Service (DDS)	The default middleware in ROS 2, offering real-time guarantees, vendor neutrality, and extensive Quality of Service controls.
Digital Twins	Virtual models of physical robotic systems that are synchronized with their real counterparts, used for predicting behaviour, diagnosing issues, and optimising performance.
Edge Computing	A distributed computing paradigm that brings computation and data storage closer to the data source (the robot or "edge") to reduce latency and bandwidth usage.
FogROS 2	A framework that extends ROS beyond individual devices into cloudlets and edge servers, facilitating distributed computation and teleoperation.
Functional Safety	The part of the overall safety that depends on a system or equipment operating correctly in response to its inputs, including the safe management of likely operator errors, hardware failures, and environmental changes.
Gazebo Harmonic	A mature, reliable physics-based simulation tool offering sensor-rich environments for general robotic development and validation.
GraphQL Integration	An emerging trend in ROS that facilitates sophisticated querying of the ROS graph state, streamlining the development of complex, responsive applications.
Hard Real-Time Control	Control systems requiring guaranteed latency, often in the microsecond range, critical for industrial robotics applications where deterministic behaviour is paramount.
Hardware-in-the-Loop (HIL)	A testing method that combines real hardware components with simulated environments to test and validate robotic software and interactions.
Heterogeneous Hardware Acceleration	The growing support in ROS for accelerators like FPGAs and GPUs to boost performance for computation-intensive tasks like perception and machine learning.
Ignition (now Gazebo)	An advanced physics-based simulation tool offering modular architecture and enhanced performance for complex scenarios and large-scale digital twin simulations.
ISO 26262	An international standard for the functional safety of electrical and electronic systems in road vehicles.
IEC 61508	A generic international standard for the functional safety of electrical, electronic, and programmable electronic safety-related systems.

Lifecycle Nodes	A ROS 2 concept introducing explicit node states (e.g., unconfigured, active, shutdown) to support safe transitions, fault recovery, and integration with industrial functional safety standards.
micro-ROS	A tailored subset of ROS 2 specifically designed for resource-constrained microcontrollers, enabling them to participate in a ROS graph.
Minimal Risk Conditions (MRC)	Industry-led standardization efforts to define safe operating conditions for autonomous mobile robots, crucial for regulatory acceptance and broader deployment.
MoveIt 2	A ROS 2 package providing advanced capabilities for robotic manipulation tasks.
Navigation 2	A ROS 2 package offering functionalities for autonomous robot mobility, including path planning and localisation.
Nodes	Independent processes in the ROS architecture, each encapsulating a distinct robotic function, such as sensor data acquisition or motion control.
Parameter Server	A mechanism in ROS (centralized in ROS 1, API in ROS 2) that allows dynamic reconfiguration of nodes' behaviours at runtime without restarting processes.
Quality of Service (QoS) Profiles	Parameters in ROS 2 that define communication reliability, latency, message durability, and deadline enforcement, tailored to different application needs.
Robotics Middleware Wrapper (RMW)	A layer in ROS 2 that abstracts the underlying data transport mechanism, allowing for different middleware implementations (e.g., DDS).
Robotic Operating System (ROS)	An open-source, modular middleware platform providing essential communication, hardware abstraction, and tooling for robot software development.
rosbag2	A tool in ROS 2 for capturing, storing, and replaying message streams, useful for debugging and analysis.
ROS-Industrial	A set of ROS packages providing drivers and tools for integrating industrial robots (e.g., from ABB, Fanuc, KUKA) with the ROS ecosystem.
rviz2 and rqt	Visualization tools in ROS 2 that enable 3D visualization of robot states and introspection of node interactions.
Services	An architectural building block in ROS that facilitates synchronous request-response interactions, suitable for episodic commands requiring immediate feedback.
SROS 2	Security tooling for ROS 2 that provides keystore management and node isolation policies to enhance cybersecurity through authentication, encryption, and access control.
Technical Steering Committees (TSCs)	Groups responsible for the technical direction and governance of the ROS project.
Topics	An architectural building block in ROS that implements an asynchronous publish/subscribe pattern, allowing nodes to exchange unidirectional data streams anonymously.
WebAssembly Runtimes	Lightweight execution environments that allow ROS nodes to run directly in web browsers or embedded systems, simplifying deployment and enabling cross-platform compatibility.