Robot Operating Systems

Robot Software Development

Abdelbacet Mhamdi

2025-08-17

MT @ ISET Bizerte

Outline

1. ROS2 Foundations

TurtleSim is a beginner-friendly tool for learning **ROS2** concepts through visual, interactive examples. This lightweight simulator offers a 2D environment where we control virtual turtles that can move around, draw lines, and respond to commands.

1.1.1 Installation

TurtleSim comes pre-installed with the **ROS2** desktop installation. We can also install it using the following command:

```bash
sudo apt install ros-humble-turtlesim # For Ubuntu 22: ROS2 Humble
```

The way to verify the installation is by checking the available executables:

```bash
ros2 pkg executables turtlesim
```

1.1.2 Launching TurtleSim

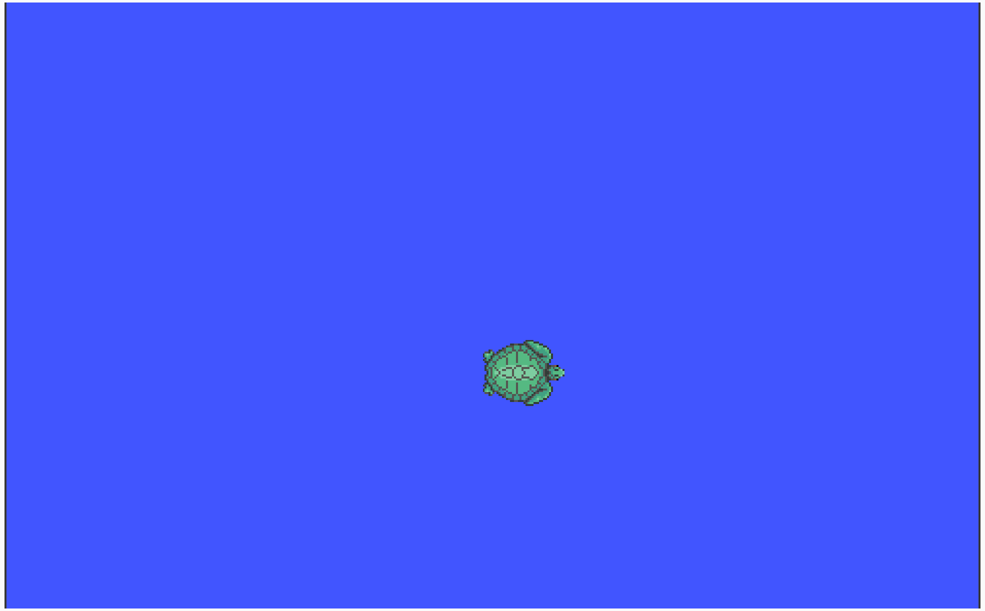
The first step is launching the *turtlesim_node*, which creates the simulation window:

```bash
ros2 run turtlesim turtlesim_node
```

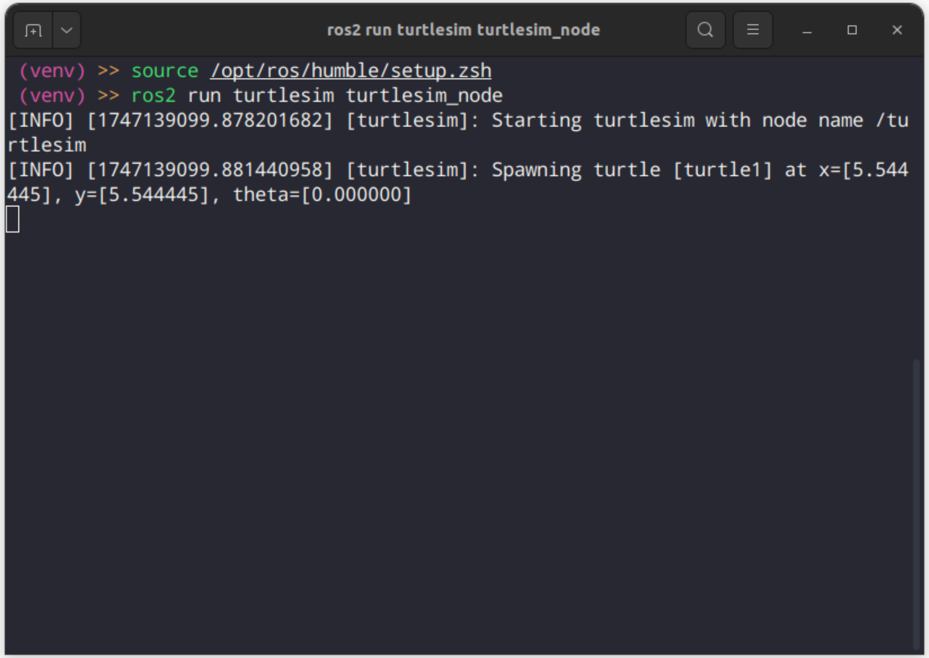
This command starts a blue simulation window with a turtle in the center.

This node:

- creates the simulation environment
- manages turtle state (position, orientation, pen status)
- processes incoming commands
- publishes turtle pose information



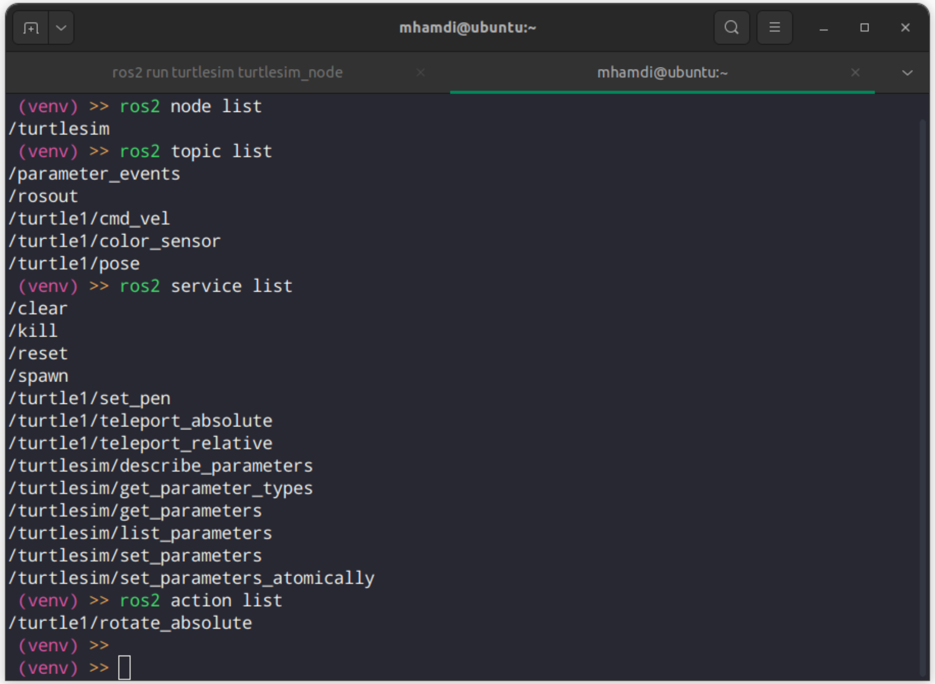
- The turtle's position and orientation are represented in a 2D coordinate system.
- The turtle can move forward, backward, and rotate, and its pen can be raised or lowered to draw on the canvas.



1.1.2.1 List Active Components

The turtle's state is managed by the *TurtleSim* node, which processes incoming commands and publishes the turtle's pose information. This allows us to control the turtle's movement and visualize its position in real time.

```bash
# running nodes
ros2 node list
# active topics
ros2 topic list
# available services
ros2 service list
# display actions
ros2 action list
```



1.1.2.2 Moving the Turtle

```bash
ros2 run turtlesim turtle_teleop_key
```

Bash

Info

Keep the teleop terminal window focused while sending keyboard commands.

Memorize

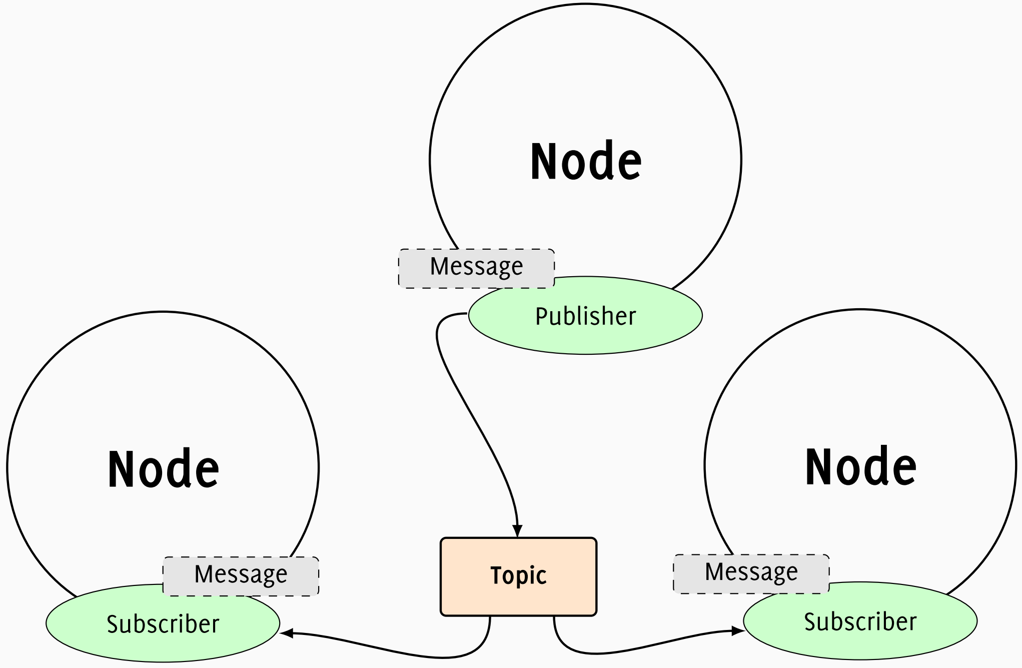
Arrow keys Move the turtle forward/backward and rotate left/right

Space Stop the turtle

1.1.3 Programmatic Control

1.1.3.1 Using Topics

Topics are named communication channels for message passing.



We can explore active topics via:

```bash
# List all active topics
ros2 topic list
```

Key topics include:

/turtle1/cmd_vel For sending velocity commands

/turtle1/pose Published turtle position and orientation

/turtle1/color_sensor Color detected by the turtle

Examine topic types and message structures:

```bash
ros2 topic type /turtle1/cmd_vel # Result: geometry_msgs/msg/Twist
```

```bash
ros2 topic echo /turtle1/pose # Echo messages on a topic
```

1. **Forward**

```bash
ros2 topic pub /turtle1/cmd_vel geometry_msgs/msg/Twist "{linear: {x: 2., y: 0., z: 0.}, angular: {x: 0., y: 0., z: 0.}}"
```

Bash

2. **Backward**

```bash
ros2 topic pub /turtle1/cmd_vel geometry_msgs/msg/Twist "{linear: {x: -2., y: 0., z: 0.}, angular: {x: 0., y: 0., z: 0.}}"
```

Bash

3. **Rotate Left**

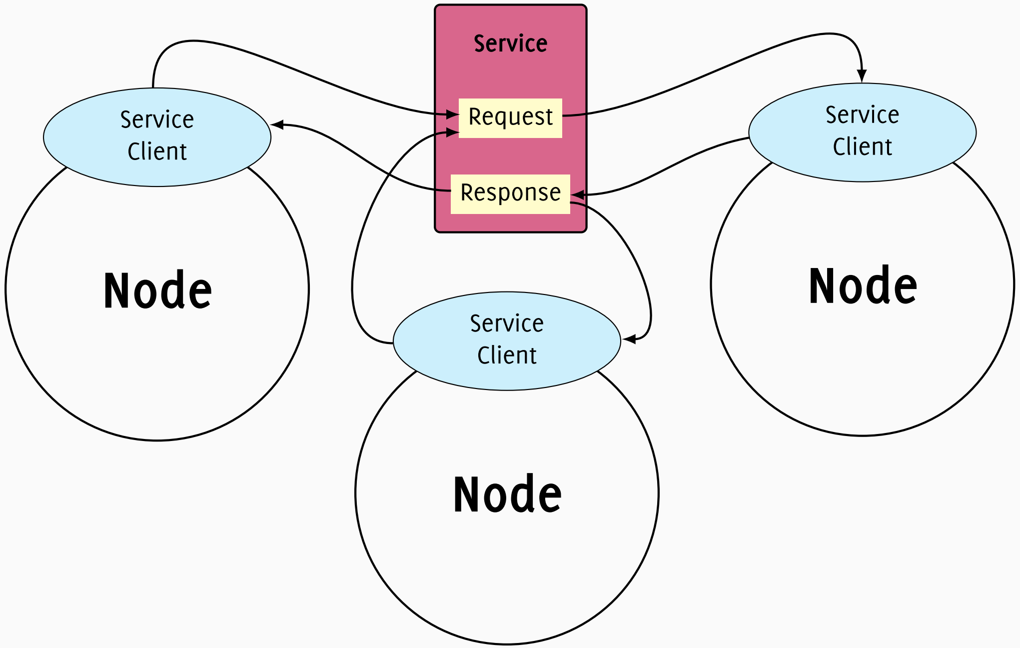
```bash
ros2 topic pub /turtle1/cmd_vel geometry_msgs/msg/Twist "{linear: {x: 0., y: 0., z: 0.}, angular: {x: 0., y: 0., z: 1.}}"
```

4. **Rotate right**

```bash
ros2 topic pub /turtle1/cmd_vel geometry_msgs/msg/Twist "{linear: {x: 0., y: 0., z: 0.}, angular: {x: 0., y: 0., z: -1.}}"
```

1.1.3.2 Using Services

Services enable us to request specific actions or information from the turtle's services.



To list all active services, we use:

```bash
ros2 service list -t
```

Bash

1. **Set Absolute Position**

```bash
ros2 service call /turtle1/teleport_absolute turtlesim/srv/TeleportAbsolute "{x: 5., y: 5., theta: 0.}"
```

Bash

2. **Set Relative Position**

```bash
ros2 service call /turtle1/teleport_relative turtlesim/srv/TeleportRelative "{linear: 1., angular: 1.}"
```

Bash

3. **Changing the Turtle's Color**

```bash
ros2 service call /turtle1/set_pen turtlesim/srv/SetPen "{r: 255, g: 0, b: 0, width: 2, 'off': 0}"
```

Bash

4. **Clearing the Screen**

```bash
ros2 service call /clear std_srvs/srv/Empty "{}"
```

Bash

5. **Resetting the Turtle**

```bash
ros2 service call /reset std_srvs/srv/Empty "{}"
```

Bash

The **Robot Operating System** (**ROS**) is an open-source, flexible framework for writing robot software. It is not an operating system in the traditional sense but rather a collection of software libraries, tools, and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms.

ROS provides functionalities typically expected from an **OS**, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management.

🏁 **Goal**

Its primary goal is to support code reuse in robotics research and development.

1.2.1 Key Characteristics of ROS

1.2.1.1 Modular Architecture

ROS is built on a distributed computing model, allowing developers to create modular software packages that can easily communicate with each other. This approach enables:

- Flexible robot software design
- Easy integration of different components
- Reusability of code across different robotic projects

1.2.1.2 Communication Infrastructure

At its core, **ROS** provides a robust communication infrastructure that allows different parts of a robotic system to exchange information:

- Publish-subscribe messaging model
- Service-client communication
- Action servers for long-running tasks
- Support for multiple programming languages (primarily C++ and Python)

1.2.1.3 Extensive Toolset

ROS offers a comprehensive set of tools for various aspects of robotics development:

- Simulation environments (like **Gazebo**)
- Visualization tools (like **RViz**)
- Debugging and logging utilities
- Hardware abstraction layers
- Package management system

1.2.2 Applications and Usage

ROS is widely adopted in:

- Academic research institutions
- Robotics laboratories
- Industrial robotics
- Autonomous vehicle development
- Research and development in various domains (manufacturing, healthcare, aerospace)

Warning

Limitations of **ROS1**

While **ROS1** achieved significant success, the evolving robotics landscape highlighted several limitations in its design, including:

Multi-robot systems Struggled with decentralized communication and discovery in large or dynamic networks.

Real-time control Lacked native support for real-time performance requirements.

Production environments Offered limited security features, making it less suitable for commercial deployments.

Resource-constrained platforms Proved resource-heavy for embedded systems.

Network reliability Depended on a single master node, introducing a single point of failure.

Idea

ROS2 Overview

ROS2 was designed from scratch to overcome the limitations of **ROS1**, delivering a more robust, secure, and adaptable platform. It supports a wide range of applications, from research prototypes to industrial and commercial systems.

Enhanced Multi-Robot Support Enables decentralized communication and discovery, addressing **ROS1**'s challenges with dynamic, large-scale robot networks.

Real-Time Capabilities Provides native support for real-time control, making it suitable for time-critical robotic applications.

Improved Security Incorporates robust security features to meet the needs of commercial and industrial deployments.

Resource Efficiency Optimized for resource-constrained platforms, such as embedded systems, unlike the heavier **ROS1**.

Network Reliability Eliminates the single point of failure by removing dependency on a single master node, improving system resilience.

1.2 What is ROS?

Distro	Logo	Release	EOL
Kilted Kaiju		23 mai 2025	December 2026
Jazzy Jalisco		23 mai 2024	May 2029
Iron Irwini		23 mai 2023	November 2024
Humble Hawksbill		23 mai 2022	mai 2027
Galactic Geochelone		23 mai 2021	9 december 2022

Foxy Fitzroy		5 juin 2020	20 june 2023
Eloquent Elusor		22 novembre 2019	novembre 2020
Dashing Diademata		31 mai 2019	mai 2021
Crystal Clemmys		14 december 2018	december 2019
Bouncy Bolson		2 july 2018	july 2019
Ardent Apalone		8 december 2017	december 2018

Notification

Introducing ROS2 Humble Hawksbill

ROS2 Humble Hawksbill, the eighth major release of **ROS2**, was officially launched on May 23, 2022. It marks a key milestone in the **ROS2** ecosystem, enhancing previous versions with new features and improvements.

- **Humble Hawksbill** is an LTS release, offering stability and extended maintenance for developers.
- Support includes bug fixes and security patches until May 2027.
- Its long support period suits commercial products, extended research projects, and educational use.
- Primarily targets **Ubuntu 22.04 Jammy Jellyfish** (`amd64` and `arm64` architectures).
- Also compatible with **Windows 10**.

Improvements in Humble

- ROSBag Enhancements
- Performance and Stability Gains
- Enhanced Documentation
- Developer Ergonomics
- Gazebo Fortress Integration
- FogROS2
- Foxglove Studio

2. Nodes and Communication

A **ROS2** node is a fundamental computational unit in the Robot Operating System 2 that represents an executable process within a robotic system. Unlike **ROS1**, **ROS2** introduces significant improvements in performance, security, and real-time capabilities.

2.1.1 Core Components

- Written in C++ or Python
- Uses `rclcpp` (C++) or `rclpy` (Python) client libraries
- Supports multiple communication patterns

2.1.2 Types of Nodes

2.1.2.1 Sensor Nodes

- Collect and process sensor data

Example

- Camera node
- LIDAR node
- IMU sensor node

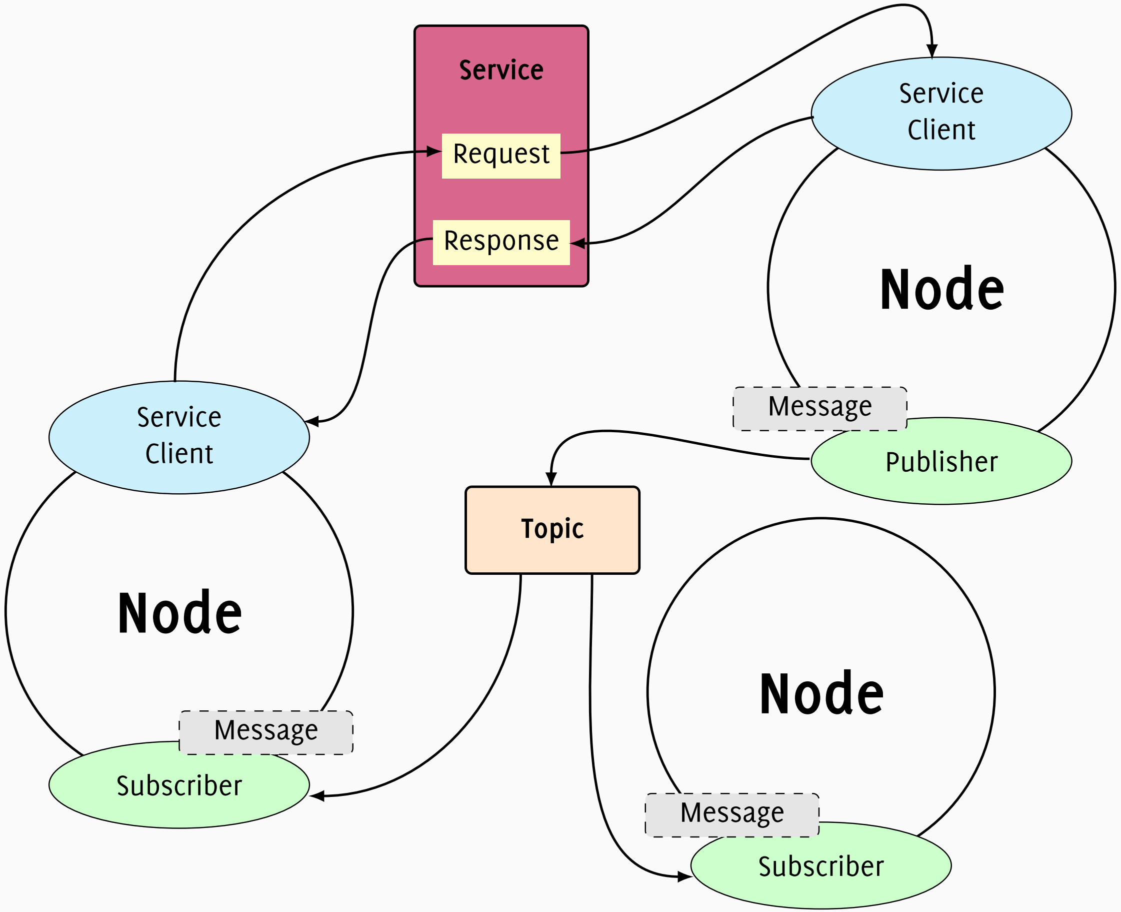
2.1.2.2 Actuator Nodes

- Control robotic actuators
- Manage precise motor movements

2.1.2.3 Processing Nodes

- Implement computational algorithms
- Perform data analysis and decision-making

2.1 What is a ROS2 Node?



Bash

```bash
# Launch individual nodes
ros2 run <pkg_name> <node_name>
# Launch multiple nodes with configuration
ros2 launch <pkg_name> <launch_file>
# Display active nodes
ros2 node list
# Show detailed information about a specific node
ros2 node info <node_name>
```

Topics, **services**, and **actions** are core communication mechanisms used to enable nodes to exchange data, request computations, or manage long-running tasks.

2.2.1 Topics

- Provide a publish-subscribe communication model
- Allow nodes to send messages to multiple subscribers
- Useful for broadcasting information

```bash
# Publisher
ros2 topic pub /chatter std_msgs/msg/String "data: 'Hello, ROS 2!'"
# Subscriber
ros2 topic echo /chatter
```

2.2.2 Services

- Provide a request-response communication model
- Allow nodes to call functions on other nodes
- Useful for tasks that require a single response

Bash

```
# Server
ros2 run demo_nodes_cpp add_two_ints_server
# Client
ros2 service call /add_two_ints example_interfaces/srv/AddTwoInts "{a: 5, b: 3}"
```

2.2.3 Actions

- Designed for long-running tasks
- Allow feedback during execution
- Support preemption and cancellation

Thank you for your attention

Bibliography

Bibliography

Brito, B., Marques, H., Oso, A., Camacho, A., Olivares-Alarcos, A., Foix, S., Perera, A., Porzi, L., Santos, C., & Kappler, D. (2021). ROS2Learn: A reinforcement learning framework for ROS 2. *Journal of Intelligent & Robotic Systems*, *102*(4), 1–19.

Castelló, J., Macenski, S., & Martín, F. (2021). Navigation2: A new generation of navigation for robots using ROS 2. *IEEE Robotics & Automation Magazine*, *28*(4), 87–98.

Di Nardo, D., Cicconetti, C., Giambene, G., Muhammad, T., Spada, F., & Bechini, A. (2022). Security challenges in Robot Operating System 2 (ROS 2): An empirical analysis. *2022 IEEE 23rd International Symposium on a World of Wireless, Mobile and Multimedia Networks (Wowmom)*, 466–472.

Hernández, E., Pérez, V., Martín, F., & Fernández, J. (2021). ROS 2 for robotics applications: A comprehensive review. *Sensors*, *21*(24), 8240.

Joseph, L., & Cacace, J. (2018). *Mastering ROS for Robotics Programming: Design, build, and simulate complex robots using the Robot Operating System*. Packt Publishing Ltd.

Koubaa, A. (2018). Service-oriented Robot Operating System for distributed robotics: A case study. *Robotics and Autonomous Systems*, *108*, 91–109.

Macenski, S., Foote, T., Gerkey, B., Lalancette, C., & Woodall, W. (2022). Robot Operating System 2: Design, architecture, and uses in the wild. *Science Robotics*, *7*(66), eabm6074.

Maruyama, Y., Kato, S., & Azumi, T. (2016). Exploring the performance of ROS2. *Proceedings of the 13th International Conference on Embedded Software*, 1–10.

Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., & Ng, A. Y. (2009). ROS: an open-source Robot Operating System. *ICRA Workshop on Open Source Software*, *3*(3.2), 5.

Reke, M., Peter, D., Schulte-Tigges, J., Schiffer, S., Ferrein, A., Walter, T., & Matheis, D. (2020). Real-time ROS 2 communication for cooperative automated vehicles. *2020 IEEE Intelligent Vehicles Symposium (IV)*, 958–963.