Introduction to Mobile Robotics

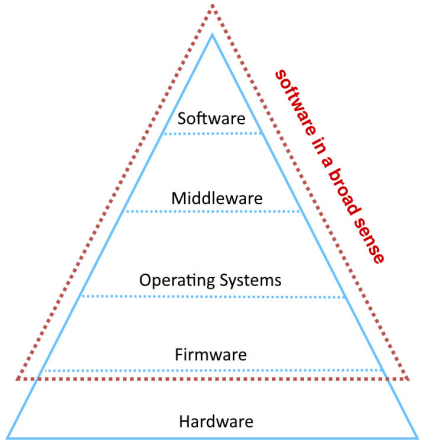
Lecture 2: Software for Robotics

Zhi Yan

ENSTA, Institut Polytechnique de Paris

Context: Computer

- A **computer system** (or computing device) includes: hardware, firmware, operating system (the "main software"), middleware and software.



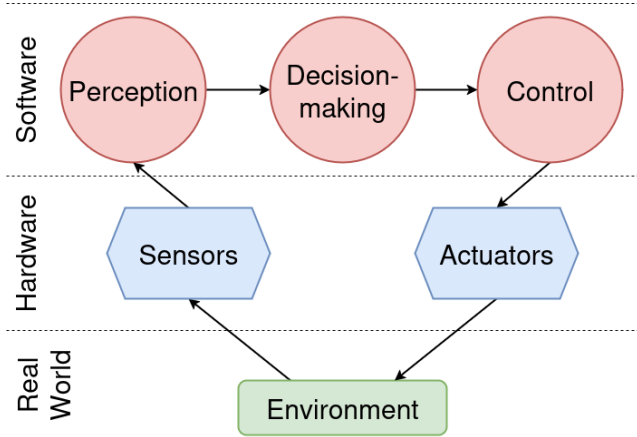
ENSTA IP PARIS

Context: Computer

- **Hardware**: CPU, RAM, motherboard, etc.
- **Firmware (stored in ROM)**: BIOS, OpenCR firmware, etc.
- **Operating System (OS)**: Microsoft Windows, macOS, Linux, etc.
- **Middleware**: Tomcat, ROS, ROS 2, etc.
  $\Longrightarrow$ Bridging the gap between an OS or database and applications.
- **Software**: Minecraft, Photoshop, Firefox, etc.

ENSTA
IP PARIS

A Robot


Software
Perception
Decision-making
Control
Hardware
Sensors
Actuators
Real World
Environment

A Robot

A complex system with many concurrent processes:

- Reading from multiple sensors (camera, LiDAR, IMU).
- Controlling multiple actuators (wheels, arms).
- Running algorithms for localization, perception, and planning.
- Communicating status to a user.



ENSTA IP PARIS

Why ROS?

Challenges:

- How do we manage this complexity?
- How do we make all these components talk to each other reliably and efficiently?
- How to maximize the reusability of developed software (e.g., from robot A to robot B) ?

Solution:

- A **middleware** like ROS[1] (Robot Operating System) and ROS 2 provides a **abstraction layer** that decouples software components, allowing them to be developed, tested, and run independently.
  $\Longrightarrow$ Since 2010, ROS has become the *de facto* standard for robotics software.

[1] `https://www.ros.org/`

ENSTA IP PARIS

ROS & ROS 2

What is ROS & ROS 2:

- A distributed architecture for inter-process and inter-machine communication and configuration.
- A collection of software packages and building tools.
- A set of development tools for system execution and data analysis.

What is ROS & ROS 2 not:

- A OS
- A programming language
- A programming environment (e.g., Visual Studio Code)

ENSTA
IP PARIS

ROS

- Active years: 2007 - 2025 (**end-of-life on May 31st, 2025**).
- Software organization: (high) modularity.
- Communication between programs: XML-RPC (for “Master”) and TCP/UDP (for “Topic”) sockets.
- Underlying OS: Mainly Linux, limited support for other OS.
- Programming language: Mainly written in C++ and Python.
- Code hosting: GitHub.
- License: 3-clause BSD License (for core of ROS).



ENSTA
IP PARIS

ROS

- Supported robot platforms: TurtleBot, HSR, BARAKUDA, and many more.
- Supported sensors: Camera, LiDAR, IMU, and many more.
- Main components:
  - Communication infrastructure (master, publisher, subscriber, etc.).
  - Robot specific features (*_msg, tf, urdf, actionlib, amcl, gmapping, navigation, etc.).
  - Tools (command-line tools, rviz, rqt, etc.).
- Powerful support (integration with other libraries):
  - Gazebo
  - OpenCV
  - PCL
  - MoveIt!
  - etc.

ENSTA IP PARIS

ROS 2

- Active years: 2014 - present.
- Release cycle: once a year.
- Does not break ROS, nor does it rollout into ROS.
- Breaking API with ROS, but conceptually very similar.
- Building on DDS (Data Distribution Service) for real-time systems.



ENSTA IP PARIS

Why Move to ROS 2?

- Modern API, minimal dependencies, and better portability (e.g., small embedded platforms).
- Benefits of underlying DDS middleware:
  - **Master-less** discovery (i.e. decentralized computation graph)
  - **Hard real-time** capable
  - Reliability
  - Efficiency (UDP Multicast, shared memory, TLS over TCP/IP, etc.)
- Easier to work with multiple nodes in one process.
- Lifecycle management and verifiable systems.
- etc.

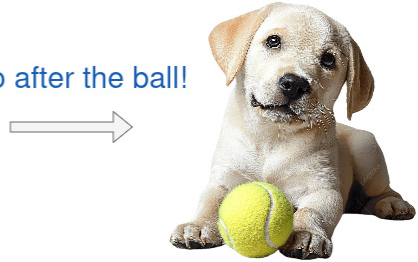
ENSTA IP PARIS

The ROS 2 Computation Graph





Go after the ball!



The ROS 2 Computation Graph

A ROS 2 system is a network of independent programs called **nodes**.

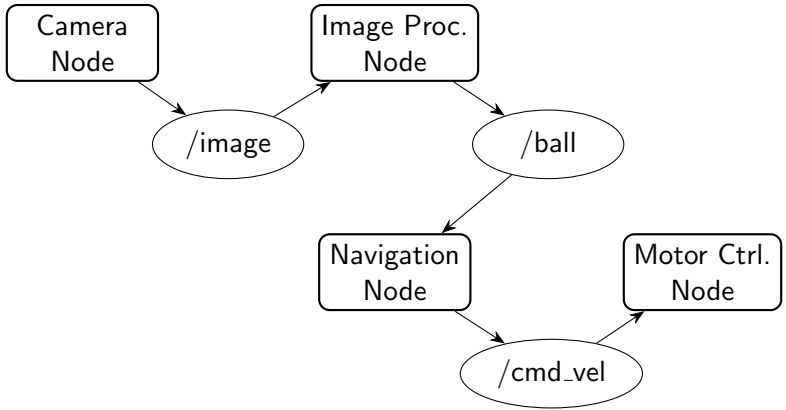


Figure: A simplified representation of a ROS 2 system.

ENSTA IP PARIS

The ROS 2 Computation Graph

- **Nodes:** Executable programs performing a specific task (e.g., controlling a camera, planning a path).
- **Communication:** Nodes communicate via mechanisms like “topics”, “services”, and “actions”.

Topics: Asynchronous Streaming Data

Publish / Subscribe model:

Topics are used for continuous data streams. One node **publishes** data to a topic, and any number of nodes can **subscribe** to that topic to receive the data.

- **Decoupled:** The publisher doesn't know or care who is subscribed.
- **Many-to-Many:** Many nodes can publish to the same topic, and many can subscribe.
- **Asynchronous:** "Fire and forget." The publisher does not wait for a response.

Examples:

- Images from a camera (`/image`)
- Motor commands (`/cmd_vel`)

ENSTA
IP PARIS

Topics: Asynchronous Streaming Data

Topic-MultiplePublisherandMultipleSubscriber.gif

Services: Synchronous Request / Reply

Client / Server model:

Services are used for remote procedure calls. A **client** node sends a request, and a **server** node performs a task and sends back a response.

- **Coupled:** The client and server are directly connected for the transaction.
- **One-to-One:** A single server handles requests from one or more clients.
- **Synchronous:** The client sends a request and **waits** until it receives a response from the server.

Examples:

- Resetting a simulation (`/reset`).
- Pawning a new robot in a simulator (`/spawn`).

ENSTA IP PARIS

Services: Synchronous Request / Reply



Actions: Asynchronous Long-Running Tasks

Client / Server model with feedback:

Actions are for long-running, goal-oriented tasks that can be preempted. A client sends a goal to an action server. The server executes the task, provides periodic **feedback**, and sends a final **result**.

- **Asynchronous:** The client does not block while the goal is being executed.
- **Preemptible:** The client can cancel the goal at any time.
- **Provides Feedback:** The client is kept informed of the task's progress.

Examples:

- "Navigate to coordinate (x, y)." The feedback could be the robot's current distance to the goal, and the result indicates success or failure.

ENSTA IP PARIS

Actions: Asynchronous Long-Running Tasks

Action-SingleActionClient.gif

Messages, Services, and Actions

How is data structured?

- **Messages (`.msg`):** Define the data structure for topics. For example, a simple `Twist.msg` for velocity might contain linear and angular components.
- **Services (`.srv`):** Define the request and response data structures, separated by `---`.
- **Actions (`.action`):** Define the goal, result, and feedback data structures, each separated by `---`.

Good-to-know:

These files (`.msg`, `.srv`, `.action`) are interface definitions, following the **Interface Definition Language (IDL)** format.

ENSTA IP PARIS

Essential Command-Line Tools

The `ros2` command is your main entry point for interacting with a running ROS 2 system.

Introspection Tools

- `ros2 node list`: See all running nodes.
- `ros2 topic list`: See all active topics.
- `ros2 service list`: See all available services.
- `ros2 action list`: See all available actions.
- `ros2 topic echo <topic_name>`: View data being published on a topic.
- `ros2 node info <node_name>`: See a node's publishers, subscribers, etc.

ENSTA IP PARIS

Essential Command-Line Tools

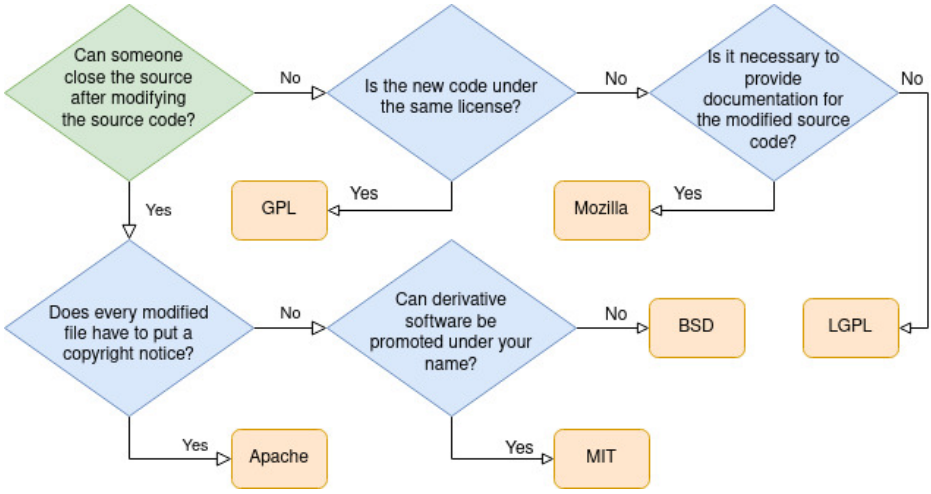
Execution Tools

- `ros2 run <package_name> <executable_name>`: Launch a single node.
- `ros2 launch <package_name> <launch_file>`: Launch a group of nodes and their configurations.

We will use these extensively in the practical work session.

ENSTA IP PARIS

How to Choose a Free Software License



Questions?

Next: Practical Work 2 - ROS 2 Beginner Level

ENSTA IP PARIS