LEONARDO

LEONARDO TECHNICAL TRAINING

Robotic Software
Lezione 2

Robot Operating System
(ROS)

Introduction to ROS

- Robot Operating System (ROS) represents a flexible framework
- It provides various tools and libraries to write robotic software.
- It offers several powerful features to help developers in such tasks as
  - message passing
  - distributing computing
  - code reusing
  - implementation of state-of-the-art algorithms for robotic applications

Introduction to ROS

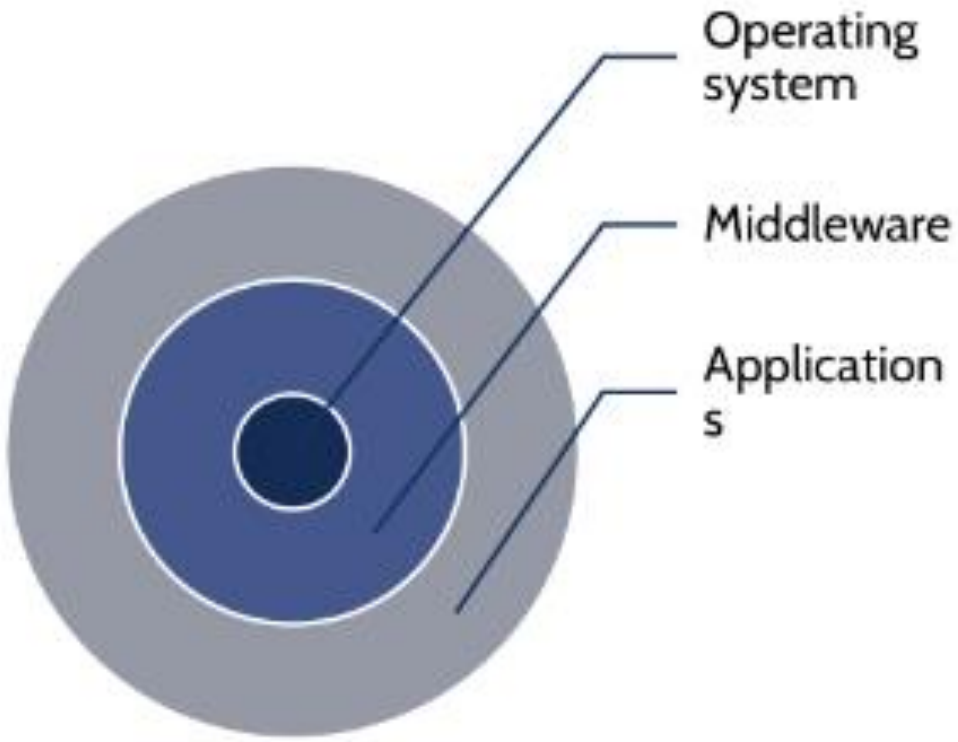
- Originally developed in 2007 at the Willow Garage and Stanford Artificial Intelligence Laboratory under GPL license
- Since 2013 managed by OSRF (Open-Source Robotics Foundation)
- Today used by many robots, universities and companies
- Standard for robots programming

Why ROS?

- Robotics developers spent too much time to re-implement the software infrastructure required to build complex robotics algorithms
  - Drivers of the sensors
  - Drivers of the actuators
  - Communication between different programs inside the same robot
- Time needed to build intelligent robotics programs?
- Es.
  - (1) Add a new camera sensor to your robot: the driver that uses the data generated by the camera must be properly and correctly integrated in your control algorithm
  - (2) Add a new gripper to your robot: the driver that commands the gripper must be implemented and included in your software structure
  - (3) How the sensor and the gripper can be linked in the same control software? How the camera driver program communicates with the gripper controller? Everything in the same source code?
  - (4) Spend too much time to implement stuff already implemented

Why ROS?

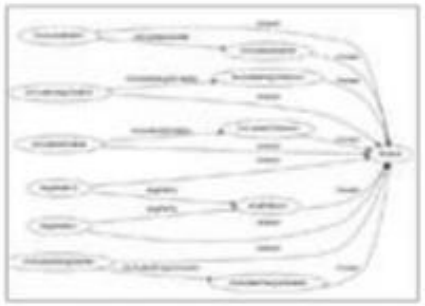
- The correct definition for ROS is a robotic middleware: a software that connects different software components or applications.



Why ROS?

- ROS programs are written in pure C++
  - Additional functionalities used to program robots are included via external libraries (ROS libraries)
  - The rest of the program is written using standard C++ and/or python functionalities
- Compile ROS programs is the same of compiling C++ programs
- Just substitute ROS functions to have classical C++ robotics programs
  - You will learn how to program robots using C++
  - ROS is a framework used to simplify the learning and programming processes

ROS Elements



Plumbing

- Process management
- Inter-process communication
- Device drivers

+

Tools

- Simulation
- Visualization
- Graphical user interface
- Data logging

+

Capabilities

- Control
- Planning
- Perception
- Mapping
- Manipulation

+

Ecosystem

- Package organization
- Software distribution
- Documentation
- Tutorials

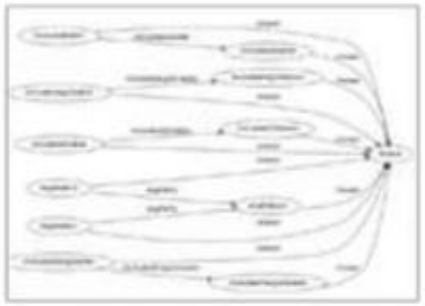
ros.org

ROS Elements (plumbing)

- ROS allows communication between different programs.
- It provides publish-subscribe messaging infrastructure designed to support the quick and easy construction of distributed (local and remote) computing systems.
  - Consider that your application uses data from a camera.
  - You can use the ROS node deployed by the vendor of your camera to implement your application.

- (3) How the sensor and the gripper can be linked in the same control software? How the camera driver program communicates with the gripper controller?

ROS Elements



Plumbing

- Process management
- Inter-process communication
- Device drivers

+

Tools

- Simulation
- Visualization
- Graphical user interface
- Data logging

+

Capabilities

- Control
- Planning
- Perception
- Mapping
- Manipulation

+

Ecosystem

- Package organization
- Software distribution
- Documentation
- Tutorials

ros.org

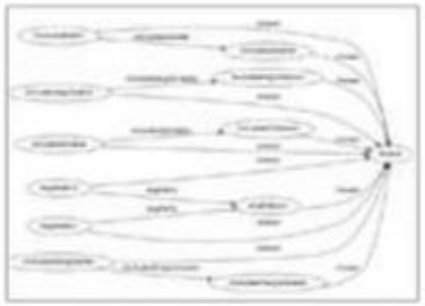
ROS Elements (tools)

- ROS provides an extensive set of tools
  - Configure
  - Manage
  - Debug
  - Visualize data
  - Log
  - Test

Tools

- Simulation
- Visualization
- Graphical user interface
- Data logging

ROS Elements



Plumbing

- Process management
- Inter-process communication
- Device drivers

+

Tools

- Simulation
- Visualization
- Graphical user interface
- Data logging

+

Capabilities

- Control
- Planning
- Perception
- Mapping
- Manipulation

+

ros.org

Ecosystem

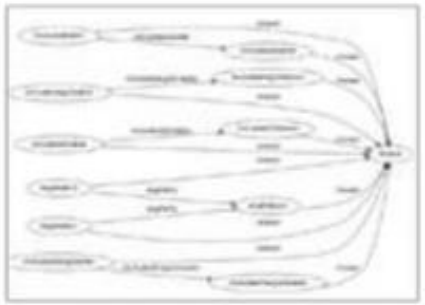
- Package organization
- Software distribution
- Documentation
- Tutorials

ROS Elements (capabilities)

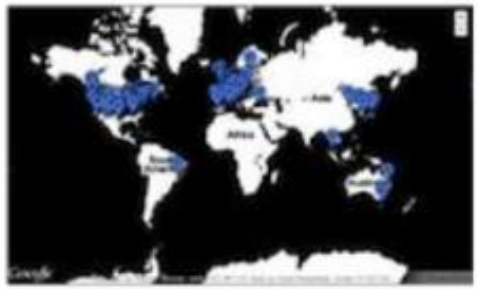
- ROS provides a broad collection of libraries that implement useful robot functionalities
  - Manipulation
  - Control
  - Perception.
- ROS can be connected to other external software like OpenCv, PCL, and so on, thanks to proper wrappers

- (4) Spend too much time to implement stuff already implemented

ROS Elements


+

+

+


ros.org

Plumbing

- Process management
- Inter-process communication
- Device drivers

Tools

- Simulation
- Visualization
- Graphical user interface
- Data logging

Capabilities

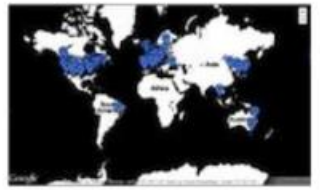
- Control
- Planning
- Perception
- Mapping
- Manipulation

Ecosystem

- Package organization
- Software distribution
- Documentation
- Tutorials

ROS Elements (capabilities)

- ROS is supported and improved by a large community
  - Integration and documentation.
  - ros.org webpage provides basic and advanced tutorial to learn how to program in ROS
  - Q&A website (answers.ros.org) allow you to directly ask you solution for your own problems (and contains thousand of question already answered).
- Robotics group can easily collaborate since ROS establish a standard for robotics programming



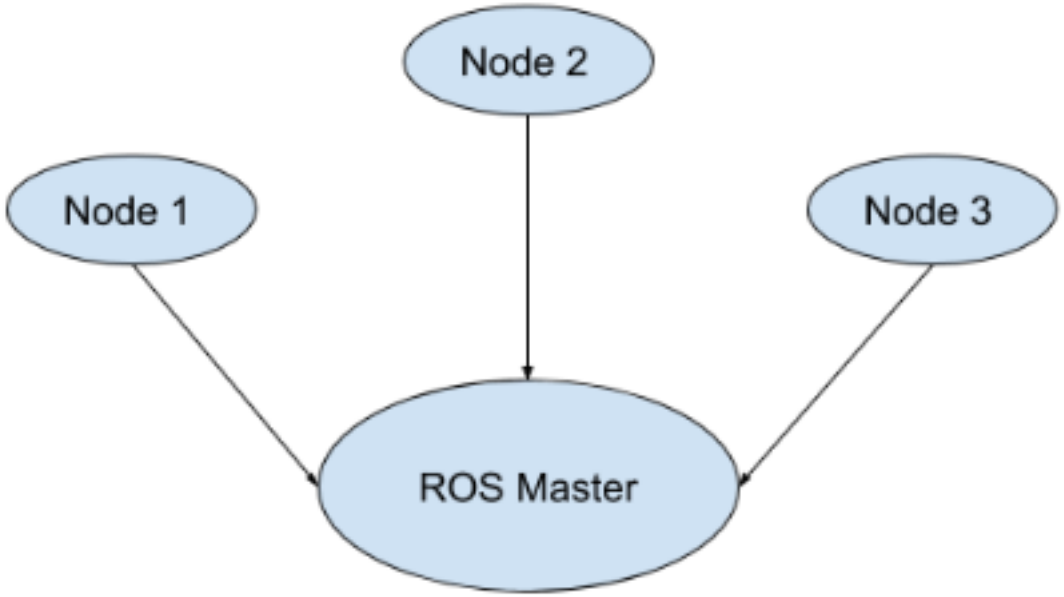
ros.org

Ecosystem

- Package organization
- Software distribution
- Documentation
- Tutorials

ROS Philosophy

- Peer to peer
- Distributed
- Multi-language (C++, Python, Java, Matlab, …)
- Lightweight
- Free and open-source



ROS Nodes

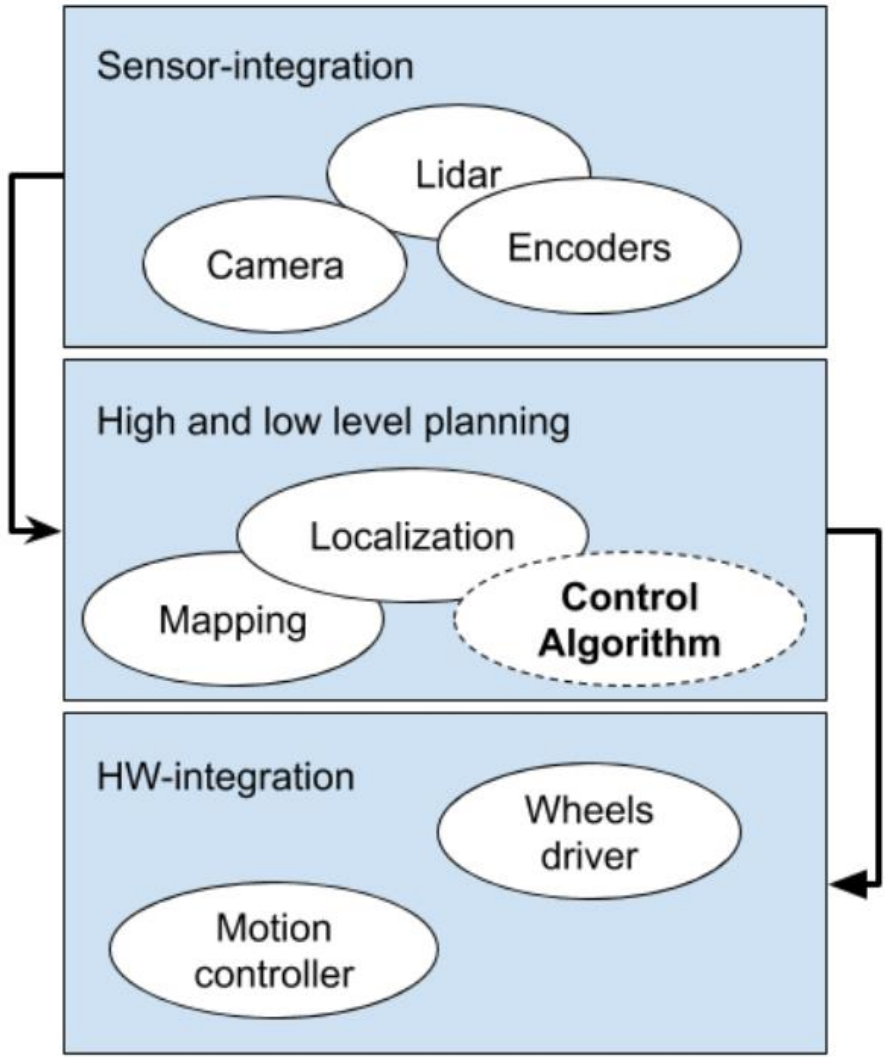
- Nodes are the processes that perform computation (the executable).
- Each ROS node is written using ROS client libraries implementing different ROS functionalities
  - Communication between nodes
  - Robotic functionalities

- ROS nodes: build multiple simple processes rather than a large process with all the functionality (modularity).

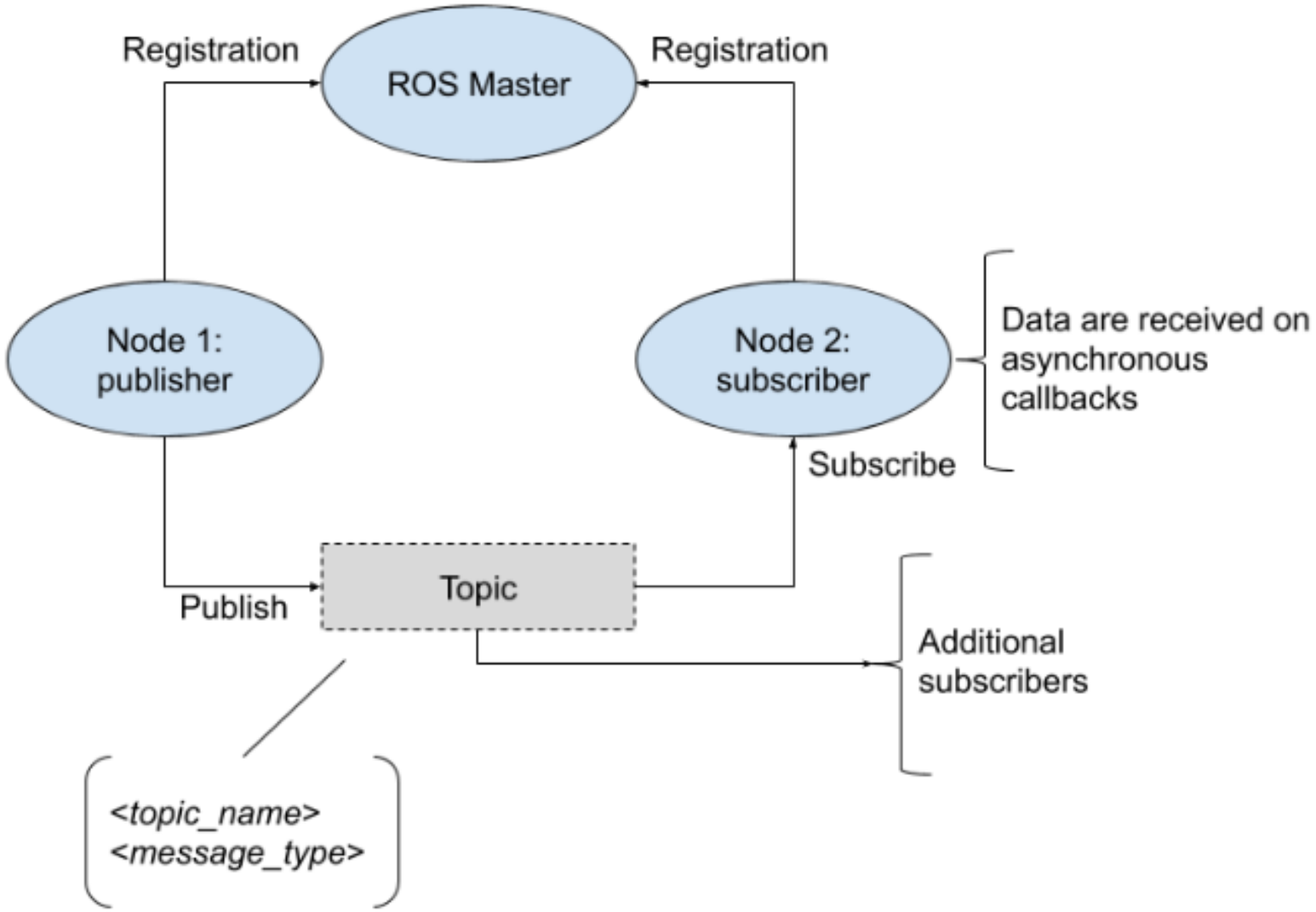
ROS Master

- ROS Master
- A particular ROS node that provide the name registration and lookup to the rest of the nodes.
- Nodes will not be able to find each other, exchange messages, or invoke services without a ROS Master.
- In a distributed system, we should run the master on one computer, and other remote nodes can find each other by communicating with this master.

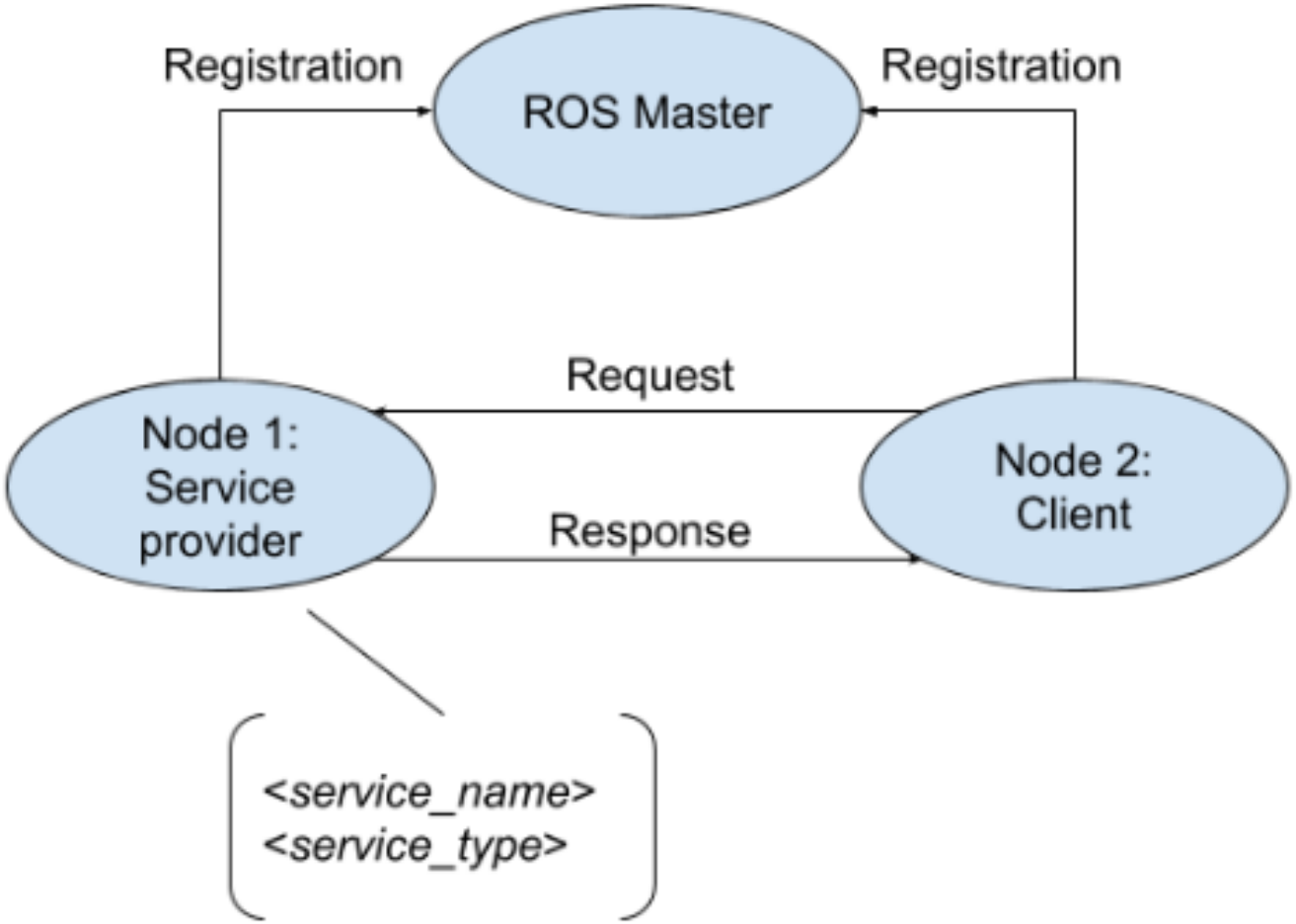
ROS: an application example



ROS Communication (publish-subscribe)



ROS Communication (Service)



ROS Messages

- ROS messages:
  - Set of standard and custom data structures
- The message definition consists in a typical data structure composed by two main types:
  - fields and constants

ROS Messages

- `geometry_msgs::PoseStamped` is used to share the pose of an object:

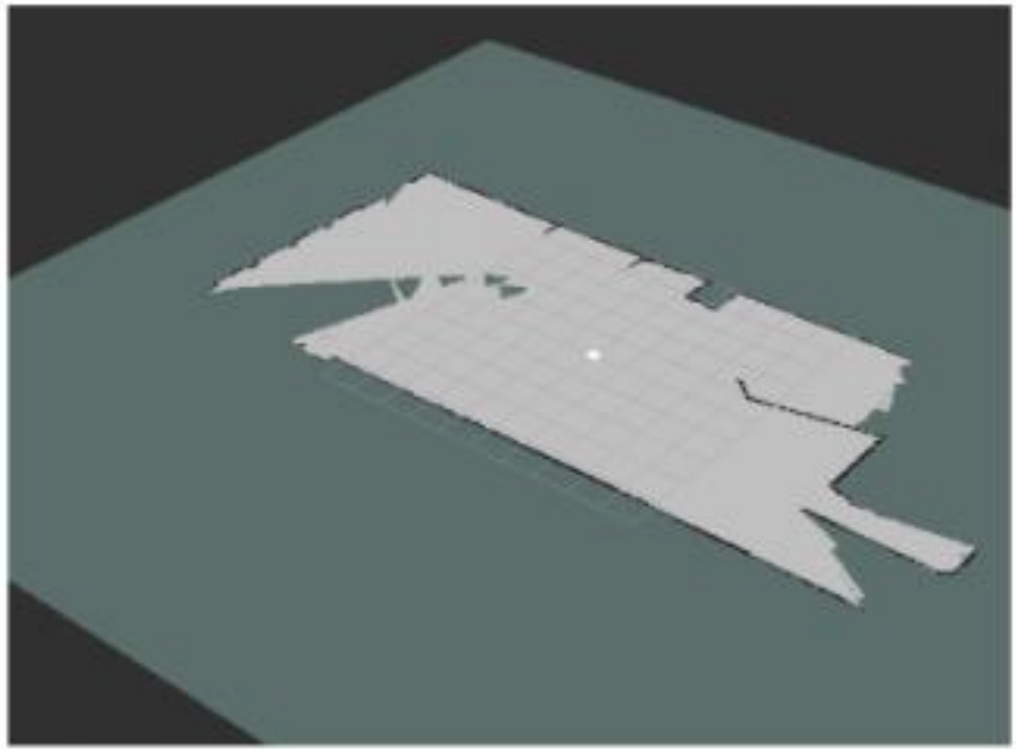
```
std_msgs/Header header
    uint32 seq
    time stamp
    string frame_id
geometry_msgs/Pose pose
    geometry_msgs/Point position
        float64 x
        float64 y
        float64 z
    geometry_msgs/Quaternion orientation
        float64 x
        float64 y
        float64 z
        float64 w
```

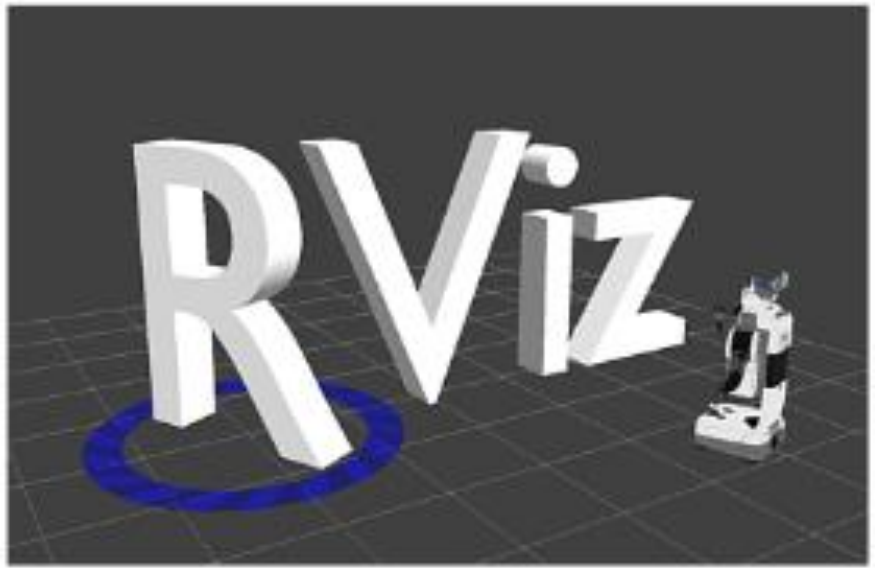
Header

Payload

ROS Visualization

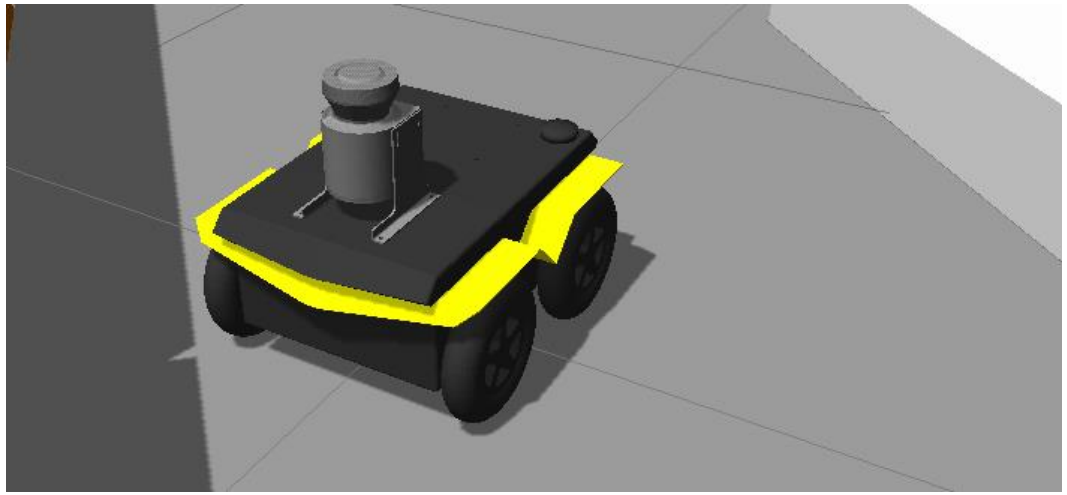
- Rviz: ROS visualization
  - Using Rviz you can directly visualize the content of some messages

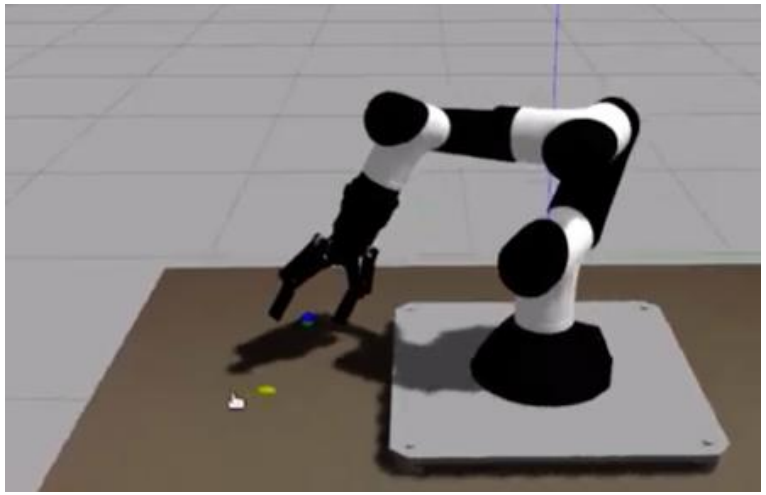




Simulation

- Simulation in robotics is fundamental in robotics
  - Prevents dangerous situations
  - Allows to work without the real hardware
- ROS has an official simulator: Gazebo
  - Dynamic simulation: physics and collisions are considered
    - The behavior of the robot is quite realistic





ROS issues

- No QoS in communication
  - Communication delay is not predictable
  - Message priority can not be managed
- Security: the ROS master will respond to requests from any device on the network
  - ROS cannot be used in industry as is
  - You can easily control ROS based device from an external network
- The core of ROS is out-of-date, and its support will end in 2025
- After 2025 ROS project officially ends
- Are we wasting our time?

ROS2

- ROS was born in 2007
  - A lot has changed in the robotics and ROS community.
- The goal of the ROS 2 project is to adapt to these changes, leveraging what is great about ROS 1 and improving what isn’t.
- ROS and ROS2 shares the programming philosophy
- ROS2 is not mature enough and more difficult to learn
  - Let’s start learning ROS (1)
- During the second part of the course, we will introduce ROS2

Install ROS

- Ubuntu 18.04
  - ROS Melodic
    - http://wiki.ros.org/melodic/Installation/Ubuntu
  - sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list’
  - sudo apt install curl # if you haven't already installed curl curl -s https://raw.githubusercontent.com/ros/rosdistro/master/ros.asc | sudo apt-key add –
  - sudo apt update
  - sudo apt install ros-melodic-desktop-full
- Workspace configuration

Install ROS

- Workspace configuration
  - After installed ROS on your machine, a new directory is added to your system
    - The default location is /opt/ros/ROS_VERSION/
  - So, for ubuntu 18.04 the default location is:
    - /opt/ros/melodic
  - To configure the workspace, we need some special command from ROS
  - Load the ROS configuration
    - $ source /opt/ros/melodic/setup.bash
  - Test that the commands have been correctly loaded
    - Use the command:
      - $ roscd
        - Now you should be moved into the /opt/ros/melodic directory

Install ROS

- Workspace configuration
  - We need to configure a workspace in the user space:
    - Create a directory in a desired location of your space and initialize it
      - $ cd /home/isaac/Desktop
      - $ mkdir –p ros_ws/src
      - $ cd ros_ws/src
      - $ catkin_init_workspace
      - $ cd ..
      - $ catkin_make
    - ros_ws is your workspace. 3 directories are included in this workspace
      - Build: temporary directory. Here are put the compiled files (executables)
      - Devel: temporary directory. Here are put compiled shared libraries
      - Src: source code directory. Only the code put in this folder will be compiled
    - To clean your workspace:
      - $ rm –rf build
      - $ rm –rf devel

Install ROS

- Load the workspace
  - The command:
    - $ source /opt/ros/melodic/setup.bash
  - Loads the workspace installed with ROS
  - This is part of the super-user space. We can not access to this directory

- Load your user-space workspace
  - $ source /home/isaac/Desktop/ros_ws/devel/setup.bash
- Put this command in your bashrc in order to load the workspace every time you open a new bash terminal
  - $ echo “source /home/isaac/Desktop/ros_ws/devel/setup.bash” >> /home/isaac/.bashrc

Example 1.2

- Create a ROS package called ros_topic with two nodes, a publisher and a subscriber
  - Publish an integer value on a topic called /numbers
  - Read the integer value on the topic /numbers
- Handle the execution of the node with the rosnode command
- Use the rostopic command the inspect and get information about the active topics

Example 1.2

- To create a new ros package use the command
  - $ catkin_create_pkg [NAME_PKG] [DEPENDENCIES]
- This command must be run in the src directory of the ROS workspace
- To move directly into the workspace, used the roscd command
  - $ roscd
  - This command brings you into the devel sub-folder
- For the Example 1.2 we consider two dependencies
  - roscpp: this dep allows us to use the C++ libraries of ROS
  - std_msgs: this dep allows us to use the standard messages class
    - http://wiki.ros.org/std_msgs
- Let’s create our package:
  - $ roscd
  - $ cd ../src
  - $ catkin_create_pkg ros_topic roscpp std_msgs

Example 1.2

- To compile a ROS package, we can use the catkin_make command
  - This command must be run in the ROOT of your ROS workspace
    - $ roscd
    - $ cd ../src
    - $ catkin_make
  - This command compiles ALL the packages in the workspace
    - If you have also only one package that doesn’t compile, the whole compilation fails

Example 1.2

- To run a node, use the *rosrun* command
  - $ rosrun [PACKAGE NAME] [NODE NAME]
- To start the topic publisher:
  - $ rosrun ros_topic topic_publisher
- Before to start the node with the rosrun command we need to start the master node
  - In a separate terminal, use the command
    - $ roscore

Example 1.2

- When a new package is created in the ROS Workspace, the list of ROS packages is updated accordingly.
- You can test using the roscd command:
  - $ roscd ros_topic
  - This command should bring you in the root of the ros_topic package
- Sometimes the update of the package list is slower than expected
- You can force this update with the command:
  - $ rospack profile
  - This command can be issued from any location of your shell

Example 1.2

- ROS commands can be used to inspect and handle topics
  - $ rostopic list: used to see the list of active topics
  - $ rostopic echo [TOPIC NAME]: get the active value published on the topic
  - $ rostopic info [TOPIC_NAME]: get info about the topics active inn the system
    - The type of the topic
    - The publisher node
    - The subscriber nodes (when existing)
- Messages can be inspected via commands:
  - $ rosmsg show [MESSAGE_TYPE]: prints info about the message data structure
    - $ rosmsg show std_msgs/Int32

Example 2.2

- Replicate Example 1.2 in python
- Create a ROS package called ros_topic with two nodes, a publisher and a subscriber
  - Publish an integer value on a topic called /numbers
  - Read the integer value on the topic /numbers
- Handle the execution of the node with the rosnode command
- Use the rostopic command the inspect and get information about the active topics

Fine lezione 2