POLITECNICO DI MILANO

AUTONOMOUS VEHICLES

S. Arrigoni

Introduction to ROS

POLITECNICO MILANO 1863

Software requirements

- Windows users: dual boot **Ubuntu 20.04** (recommended)
  https://www.ubuntu-it.org/download
- Windows + iOS: Virtual machine
  Vmware? https://www.vmware.com/products/desktop-hypervisor.html

Pay attention: you need at least 80Gb of HD space

Follow instructions available at:
https://emanual.robotis.com/docs/en/platform/turtlebot3/quick-start/

Software requirements

ROS

Logo of ROS Noetic Ninjemys

ROS Noetic

Kinetic | Melodic | **Noetic** | Dashing | Foxy | Windows

3. 1. 2. Install ROS on Remote PC

Open the terminal with `Ctrl`+`Alt`+`T` and enter below commands one at a time.
In order to check the details of the easy installation script, please refer to the script file.

```
$ sudo apt update
$ sudo apt upgrade
$ wget https://raw.githubusercontent.com/ROBOTIS-GIT/robotis_tools/master/install_ros_noetic.sh
$ chmod 755 ./install_ros_noetic.sh
$ bash ./install_ros_noetic.sh
```

Software requirements

- Try in a terminal to type:

```
raibuntu@RaiBuntu66:~/catkin_ws$ roscore
... logging to /home/raibuntu/.ros/log/97cf9a90-3cec-11ed-accc-272f0f9d8953/roslaunch-RaiBuntu66-17705.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://localhost:33145/
ros_comm version 1.15.14


SUMMARY
========

PARAMETERS
 * /rosdistro: noetic
 * /rosversion: 1.15.14

NODES
```

- Next time…

Software requirements

- Matlab 2022a(tested) / 2022b

- Type in command window: rosinit

```
>> rosinit
Launching ROS Core...
Invalid Python executable: ''. Use pyenv function to set the path to the Python executable and retry the
command.
```

- Install python (3.9.2) from https://www.python.org/ and type

```
>> pyenv('Version','/usr/bin/python3.9')
```

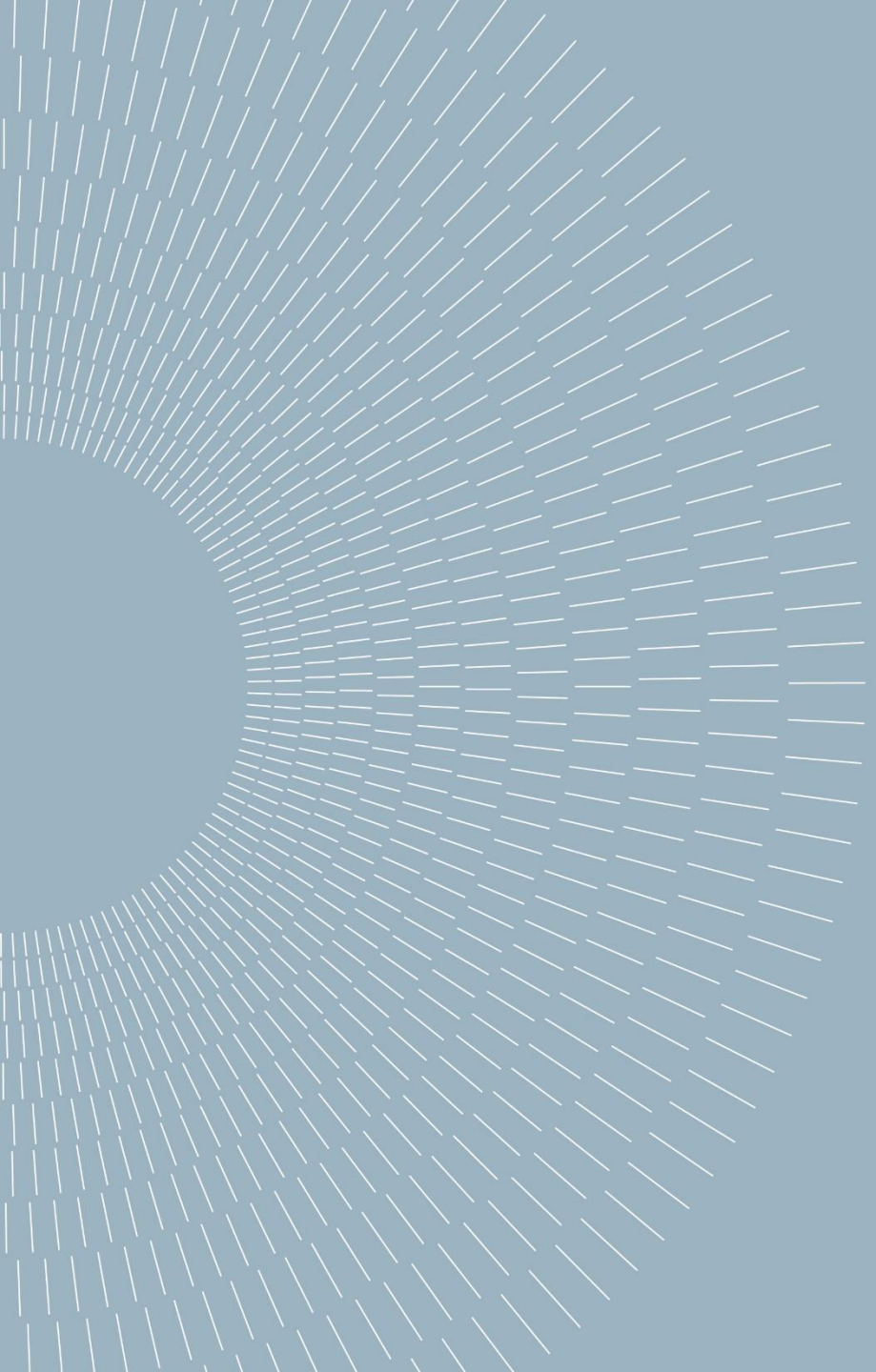
Software requirements

- Matlab 2022a(recommended) / 2022b

- Type in command window (again): rosinit

```
>> rosinit
Launching ROS Core...
Creating a Python virtual environment...Done.
Adding required Python packages to virtual environment...Done.
Done in 0.60235 seconds.
Initializing ROS master on http://192.168.1.77:11311.
Initializing global node /matlab_global_node_70367 with NodeURI http://arrige66:59118/ and MasterURI ht
>>
```

If you install 2022b on you will have to fix a couple of things more…



ROS

Intro

POLITECNICO MILANO 1863

Robot Operating System (ROS) is an open-source, meta-operating system for your robot.

It provides:

- hardware abstraction
- low-level device control (*drivers*)
- implementation of commonly-used functionality (*libraries, visualization*)
- message-passing between processes
- package management

Source: http://wiki.ros.org/ROS/Introduction

ROS is a middleware

middleware= “A bridge between software application and low-level hardware”

Why middleware?

- **Portability**: common programming model regardless specific programming language or system architecture.
- **Abstraction**: low-level aspects are handled by libraries and drivers inside the middleware.

Is it the only one?

"One Ring to rule them all, One Ring to find them, One Ring to bring them all and in the darkness bind them."

A partial list…

- **ROS**
- **YARP**
- **OROCOS**
- **ORCA**
- **BRICS**
- **…**

Why ROS?

Among the others:

- From 2009 is still growing!

+ 25%
7/2020 = 7410
1/2022 = 9260

Total number of papers citing:
"ROS: an open-source Robot Operating System"
(Quigley et al., 2009)

Source: Google Scholar 2022-01-28

- Closest thing to "industrial standard"

Why ROS?

Among the others:

- Plenty online- resources (http://wiki.ros.org)

- Already integrates several helpful projects! (*and hardware*)

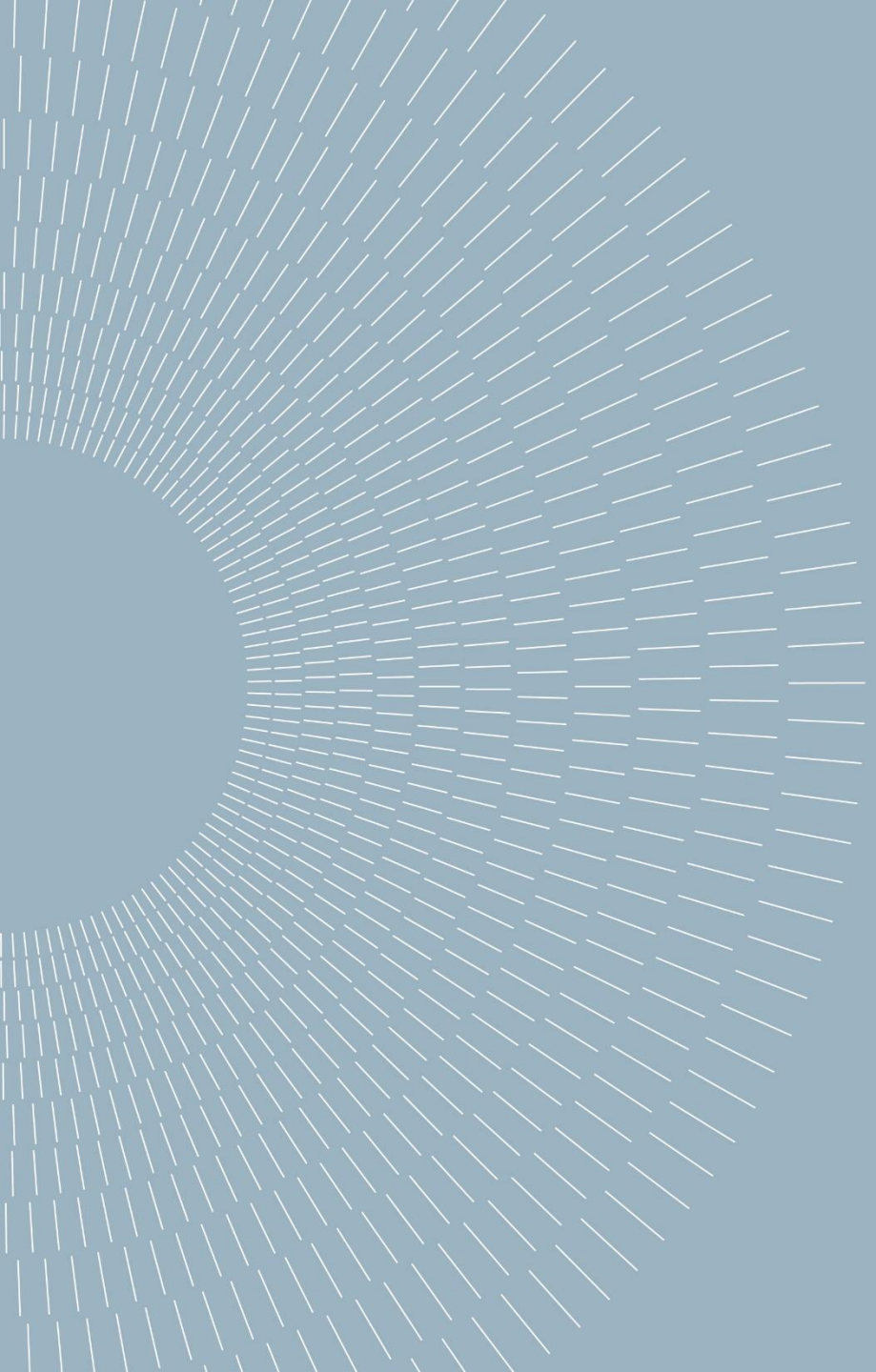
  (***OpenCV*** (computer vision); ***Gazebo*** (3d-simulation); others ROS "***wrappers***" to existing software; …)

…what's next?

ROS2

- Ecosystem brand new (more complex than ROS1)
- Less documents and tutorial available
- Faster learning curve
- Fundamental design pattern is similar to ROS1

- … It's going to be the future!

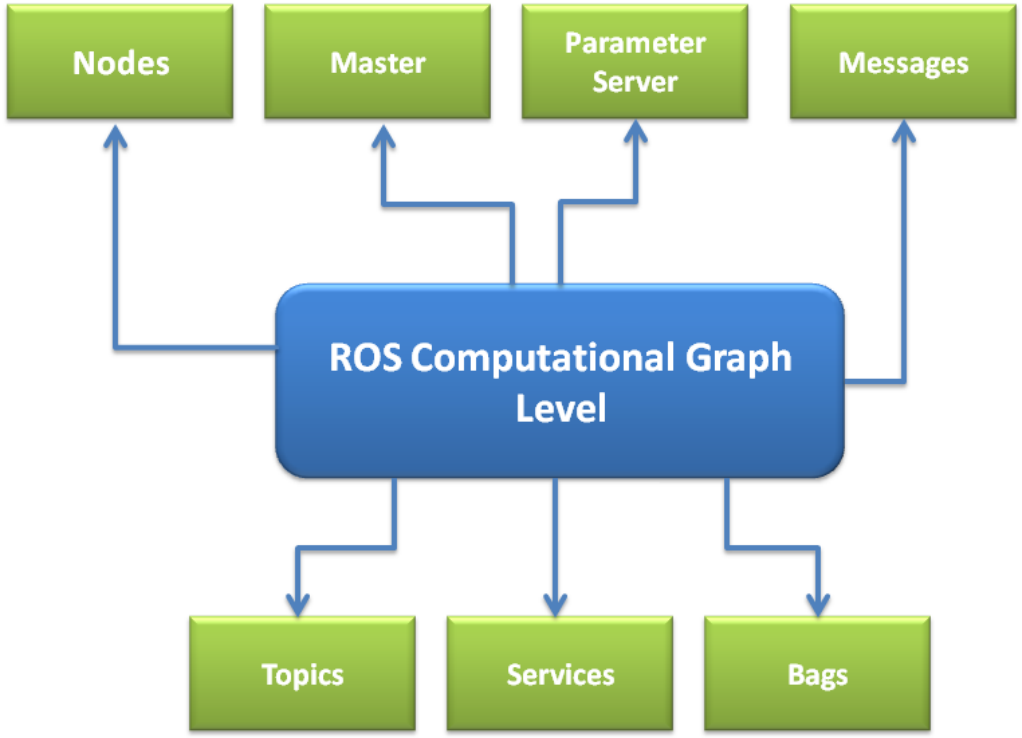


ROS

Main aspects

POLITECNICO MILANO 1863

How it works



Computation in ROS is done using a peer-to-peer network of processes called nodes. Any node can access this network, interact with other nodes, see the information that they are sending, and transmit data to the network

This network can be called computational graph. The main components are:
Nodes, ***Master***, ***Parameter server***, ***Messages***, ***Topics***, ***Services***

Nodes

Process that performs computation.
Each ROS node can be done in python, C++, …
It can communicate with each other and exchange data using the ROS communication methods

rosrun package_name node_name

Master

Provides naming and registration services
allows nodes to locate one another and to interact
One master for each system (even on distributed architectures)

Inside *roscore*

Parameter server

Shared, dictionary that is accessible via network
Nodes use this to store and retrieve parameters at runtime
(Not for performance and data exchange)

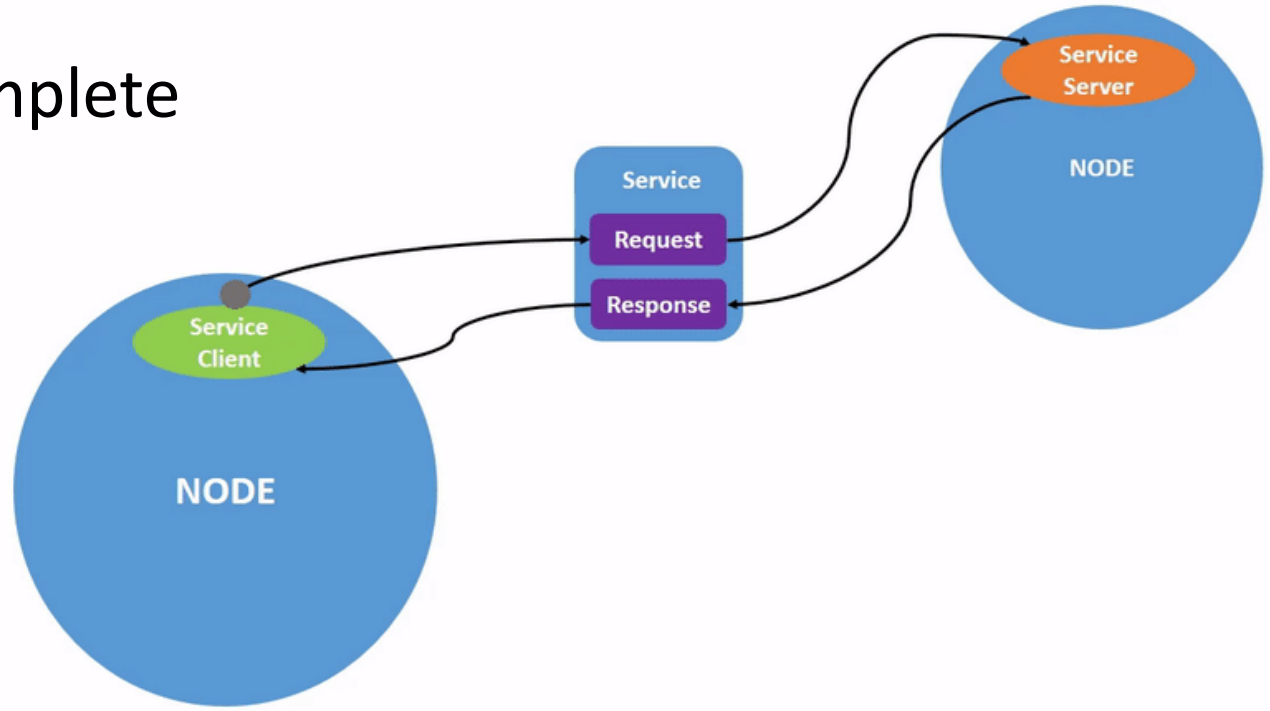
rosparam set par_name par_value

rosparam get par_name

Inside *roscore*

Services

- Work like remote function calls
- Implement the client/server paradigm
- Code waits for service call to complete
- Guarantee of execution



Messages

Nodes communicate with each other by publishing messages to topics
They are simple data structure, comprising typed fields

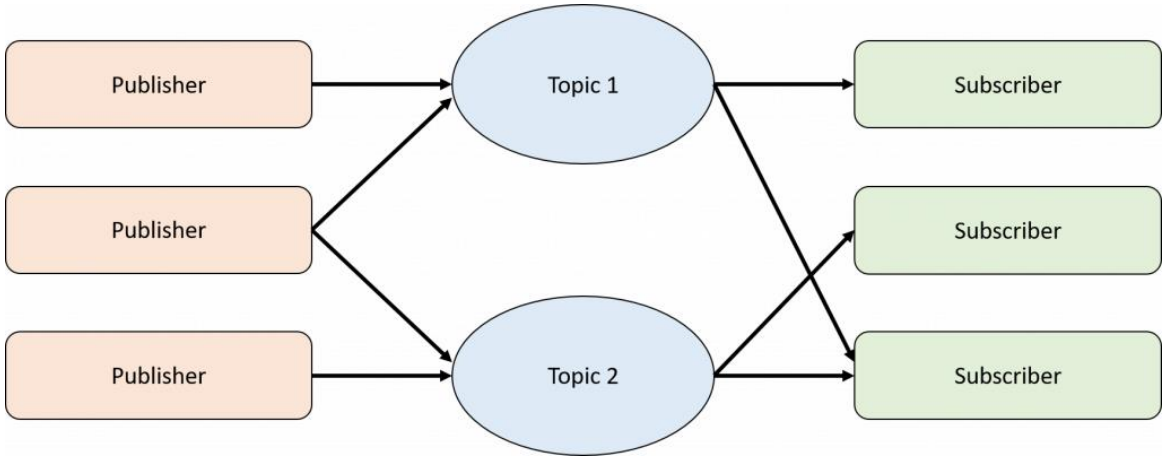
Various already available messages (http://wiki.ros.org/std_msgs)

It is possible to define new messages in a simple way (existing types can be used)

rosmsg list

Topics

Nodes route messages with publish/subscribe paradigm through topics:
topic is a name that is used to identify the content of the message



- No guarantee of delivery
- specific message type
- multiple concurrent publishers and subscribers for a single topic
- single node may publish and/or subscribe to multiple topics

Bags

data format (*.bag) for saving and playing back messages

mechanism for data logging: record anything exchanged on the ROS graph (messages, services, parameters, actions)

useful for:
analyzing, storing, visualizing data and testing algorithms

rosbag record /topic1 /topic2 (-a)

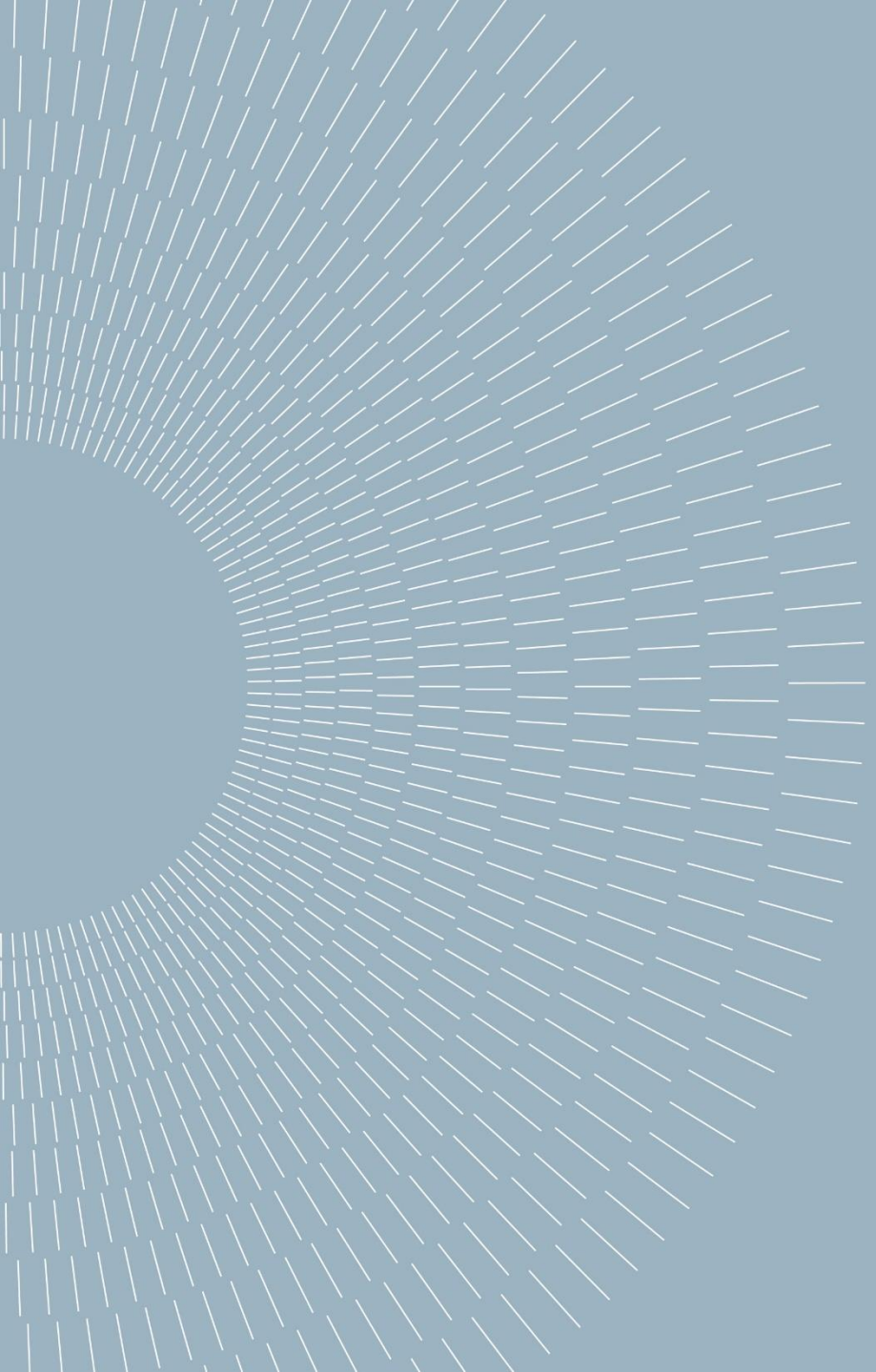
rosbag play ~/folder/name_log.bag
rqt_bag ~/folder/name_log.bag

Roscore

roscore is a collection of nodes and programs that are pre-requisites of a ROSbased system

Elements of roscore:

- a ROS Master
- a ROS Parameter Server
- a rosout logging node



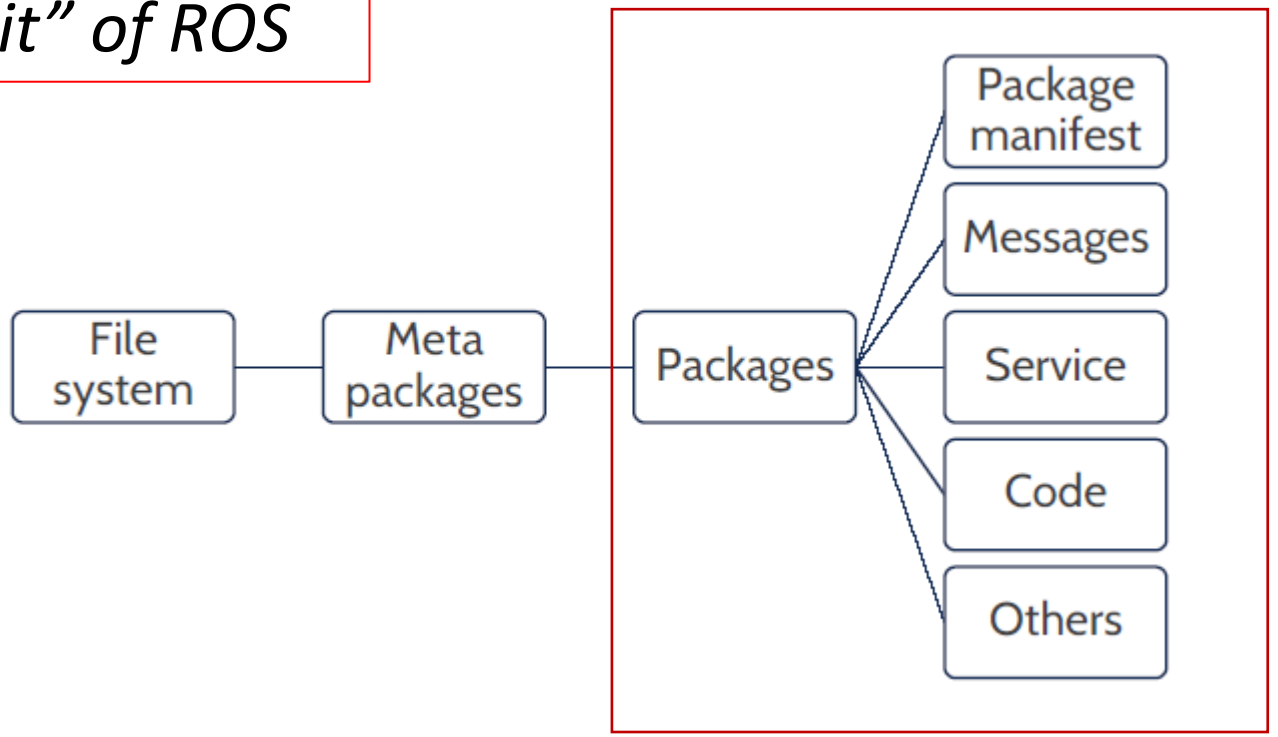
ROS

Create our first package

POLITECNICO MILANO 1863

Ros Packages

Package is the "atom unit" of ROS



Package

Folder structure:

`/src`, `/include`, `/scripts` (coding)

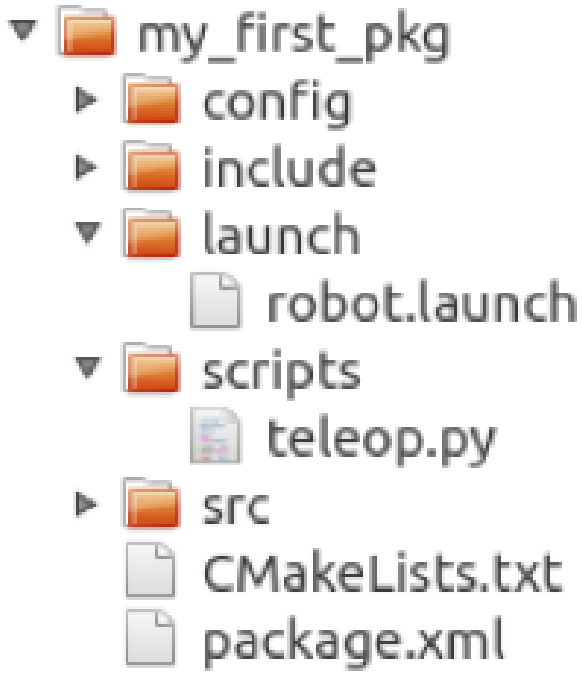
`/launch` (launch files)

`/config` (configuration files)

Required files:

`CMakeList.txt`: Build rules for catkin

`package.xml`: Metadata for ROS



Ros environment

Workspace: folder where modify, install and bould packages

mkdir -p ~/ws_folder/src
cd ~/ws_folder/
catkin_make

```
workspace_folder/
├─ build/
├─ devel/
└─ src/
   ├─ package1/
   |  ├─ CMakeLists.txt
   |  ├─ package.xml
   |  ├─ config/
   |  ├─ launch/
   |  ├─ include/
   |  ├─ src/
   |  └─ scripts/
   |
   …
   └── packageN/
```

Package

In order to create a package:

cd ~/ws_folder/src

catkin_create_pkg <package_name> [dep1] [dep2] [dep3]

Example in python

catkin_create_pkg pub_node std_msgs rospy

Publisher (python)

Let’s create a publisher in Python!

mkdir scripts

cd scripts

Let’s include scripts in a specific folder!

Create script and make executable : (gedit publisher_node.py)

chmod +x publisher_node.py

Make it executable

Publisher (python)

Let’s create a publisher in Python!

modify CMakeLists.txt to make sure the python script gets installed properly

catkin_install_python(PROGRAMS scripts/publisher_node.py DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION})

Publisher (python)

```
#!/usr/bin/env python
import rospy
from std_msgs.msg import String

def say_something():
        pub = rospy.Publisher('topic_name', String, queue_size=10)
        rospy.init_node('publisher_node', anonymous=False)
        rate = rospy.Rate(10) # 10hz
        while not rospy.is_shutdown():
                msg = "hello world %s" % rospy.get_time()
                pub.publish(msg)
                rate.sleep()
if __name__ == '__main__':
        try:
                say_something()
        except rospy.ROSInterruptException:
                pass
```

Publisher (python)

```
#!/usr/bin/env python


import rospy
from std_msgs.msg import String


def say_something():


    pub = rospy.Publisher('topic_name', String, queue_size=10)


    rospy.init_node('publisher_node', anonymous=False)
```

declaration to execute as Python script.

import for a python ROS Node
std_msgs.msg import to use String message type

Definition of say_something function

*declares that your node is publishing "**topic_name**" topic using message type String. (queue_size limits queued messages)*

tells rospy the name of your node (anonymous = True ensures your node a unique name by adding random numbers)

Publisher (python)

```
    rate = rospy.Rate(10) # 10hz


    while not rospy.is_shutdown():
            msg = "hello world %s" % rospy.get_time()
            pub.publish(msg)
            rate.sleep()



if __name__ == '__main__’:
    try:
             say_something()
    except rospy.ROSInterruptException:
            pass
```

creates a Rate object which method sleep() allows for looping at the desired rate

*Checks **rospy.is_shutdown()** flag and then doing work. In this case **pub.publish(msg)** publishes a string to our topic.*
rate.sleep() maintain the desired rate.
***rospy.get_time()** gets the current time*

After standard Python main check, runs say_something() The reason of this exception is to not accidentally continue executing code after the sleep()

Publisher (python)

Create and initialize the node

cd ~/ws_folder
catkin_make

Create the node

source devel/setup.bash

make it visible

rosrun pub_sub publisher

Run it!

+

roscore

Initialize ROS env

tools

Monitor your ROS environment:

Check nodes *rosnode list*

Check topics *rostopic list*

See topic *rostopic echo /publisher*

```
raibuntu@RaiBuntu66:~/catkin_ws$ rosnode list
/publisher_node
/rosout
raibuntu@RaiBuntu66:~/catkin_ws$ rostopic list
/rosout
/rosout_agg
/topic_name
raibuntu@RaiBuntu66:~/catkin_ws$ rostopic echo /topic_name
data: "hello world 1664122649.520881"
---
data: "hello world 1664122649.6207736"
---
data: "hello world 1664122649.7208068"
---
data: "hello world 1664122649.8208137"
---
data: "hello world 1664122649.9209445"
---
data: "hello world 1664122650.0209439"
---
data: "hello world 1664122650.1209655"
---
data: "hello world 1664122650.2209675"
---
^Cdata: "hello world 1664122650.3210337"
---
```

That's it for today...

See you next time!

S. Arrigoni