Introduction to ROS2
JNRR 2023 Moliets

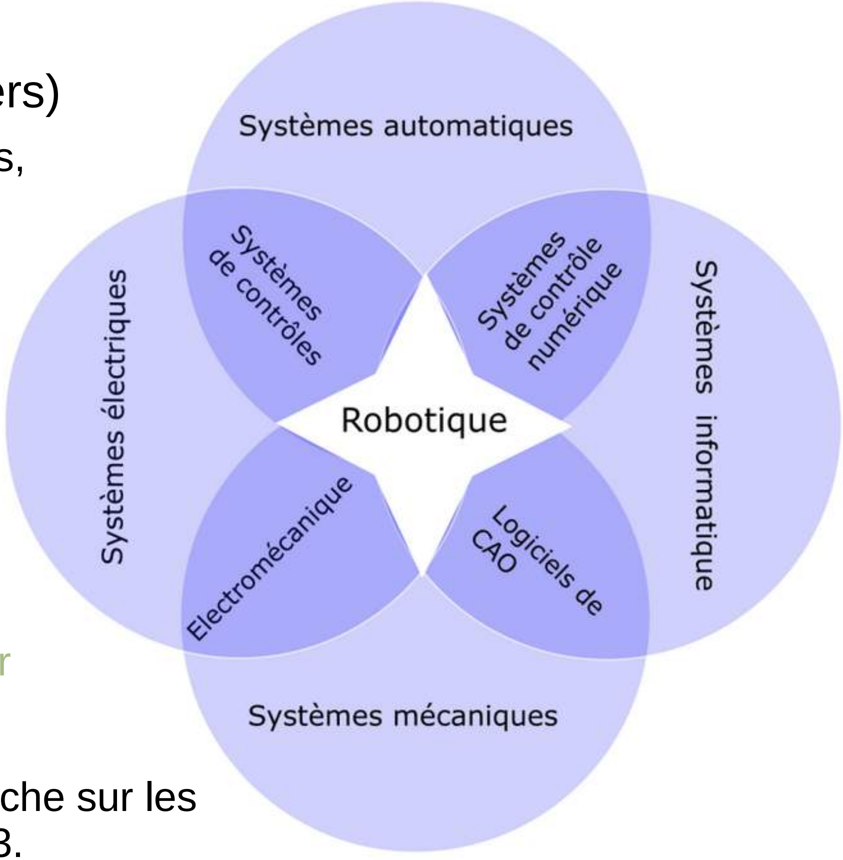
^{2}RM : Professional Network in robotics and mechatronics of CNRS

https://2rm.cnrs.fr/

Gerald Dherbomez
gerald.dherbomez@univ-lille.fr

^{2}RM

- A professional network of CNRS-MITI : https://miti.cnrs.fr/plateforme-reseaux/les-reseaux/
- A mailing list : 2rm@services.cnrs.fr (~250 subscribers)
  - Sharing technical and scientific knowledge, good practices, collaborate on national projects
- A wiki with technical data : https://wiki.2rm.cnrs.fr
- Training session for a week :
  - ANF DeepRobot, Lille, in 2019
  - ANF ROS2, Frejus, in 2022
  - ANF ROS2, Lille, in 2023
  - ANF RUST language, Lille, in 2024
  - If you have needs, contact us : 2rm-copil@services.cnrs.fr
- Tech days with focus on a robotics technology
  - For example, “Outils Logiciels et Matériels pour la Recherche sur les Véhicules Terrestres Autonomes”, Saclay, the 5th Oct 2023.
  - Other events on the website.



Workshop objectives

- Giving you a brief overview of the ROS ecosystem
- Manipulating ROS fundamental concepts
- Knowing the main differences between ROS1 to ROS2 and how to migrate from the first version to the second one
- Playing with ROS tools
- Giving you references to go further with the usage of ROS

Why ROS ?

- The default OS on a lot of robots for the academic research and more and more used in industry.
- Popularity in the robotic scientific community
- Uses default programming tools linux, C++, Python, cmake, bash, yaml, …)
- Open source *and* free (BSD license)
- Supports several hundreds of robot platforms, sensors and actuators
- More than 10 years of existence

What is ROS ?

- ROS (Robot Operating System) is not really an Operating System since it is based on existing distros like Ubuntu but more a framework with a communication middleware and associated tools for programming robots in a standardized and interoperable method.
- It provides functionalities for roboticians :
  - The “**plumbing**” : how to connect your softaware components together without being obliged to go deeply in the network or system layers.
  - The **tools** to help debugging, making diagnosis, displaying data, …
  - The **capabilities** off the shelf to build quickly a robotics application (moving, data processing, navigation libraries …)
  - The **ecosystem** : you and other researchers and engineers ready to collaborate and share knowledges together.

See chapter 1.2 “Concepts de ROS” of Olivier Stasse
https://wiki.2rm.cnrs.fr/AnfRos2/Supports?action=AttachFile&do=view&target=anf-2022-polycopie.pdf

ROS2 in practice

- ROS open source eco-system
  - 2666 repositories, 7616 packages (@16th oct 2023)
  - https://index.ros.org/packages/page/1/time/
- ROS 2 binary packages for the following platforms:
  - Ubuntu Linux - Jammy Jellyfish (22.04)
    - Debian packages (recommended)
    - “fat” archive
  - RHEL 8
    - RPM packages (recommended)
    - “fat” archive
  - Windows (VS 2019)
- Building ROS 2 from source on the following platforms:
  - Ubuntu Linux
  - Windows
  - RHEL
  - macOS

It is advised to use LTS version of ROS (same as Ubuntu LTS) so Humble release in May 2022, end of life May 2027.

A rolling release of ROS2 is also available : https://docs.ros.org/en/rolling/Releases.html#rolling-distribution

↓

⚠ breaking changes

ROS1 functionalities

- Standardisation communication mechanism for robotics
  - std_msgs : http://wiki.ros.org/std_msgs
  - common_msgs : https://wiki.ros.org/common_msgs
  - OS and programming language independent
- Abstract communication layer to send messages between processes
  - Via TCP-IP in ROS1, network layer transparent for the developer http://wiki.ros.org/rostopic
  - Dynamic discovery of components http://wiki.ros.org/rosmaster
  - Serialization of messages for all architectures compatibility
- Two programming langages Python (http://wiki.ros.org/rospy) and C++ (http://wiki.ros.org/roscpp)
  - Several bindings for other langages (Java, Go, LISP, Pharo … http://wiki.ros.org/Client%20Libraries)
- Several tools
  - Data recording : http://wiki.ros.org/rosbag
  - Change of coordinate system : http://wiki.ros.org/tf
    - Démo : http://wiki.ros.org/tf/Tutorials/Introduction%20to%20tf
  - Simulation with Gazebo : http://gazebosim.org/

Focus on some ROS2 functionalities

- Standardization communication mechanism for robotics
  - std_msgs and common_msgs, … available on github https://github.com/ros2/common_interfaces/tree/humble/std_msgs/msg
- Focus on messages :
  - diagnostic_msgs : standard way to publish warnings and errors about your nodes cycle life
  - geometry_msgs : informations about positions (2D, 3D), accelerations, inertia tensors, quaternion, twist (linear and angular commands), wrench (forces and torques)
  - nav_msgs : grid cells, occupancy grids, odometry (pose and twist with covariances), path (waypoints to follow)
  - sensors_msgs : battery, camera, images, IMU, joint state (position, velocity and effort of the joint), joystick, LIDAR data, range (US or IR), temperature, humidity, GNSS receivers (NavSatFix).
  - shape_msgs : mesh, vertex, plane, primitives (box, sphere, cylinder, cone and prism)
  - std_msgs : byte, integers, float, string, array, …
  - stereo_msgs : disparity image
  - trajectory_msgs : trajectory to follow, mainly for manipulators
- Abstract communication layer to send messages between processes
  - Via DDS and RTPS, https://docs.ros.org/en/humble/Installation/DDS-Implementations.html
  - Different implementations : **eProsima's Fast DDS**, RTI's Connext DDS, Eclipse Cyclone DDS, and GurumNetworks GurumDDS
- Simulation :
  - Gazebo simulator not tightly integrated in ROS2
  - See the compatibility of your Gazebo version with ROS2 in this table: https://github.com/gazebosim/ros_gz/blob/ros2/README.md

Documentation and Help

- ROS2 official documentation (Humble) : https://docs.ros.org/en/humble/index.html
- Wiki : http://wiki.ros.org/
- Tutorials : http://wiki.ros.org/ROS/Tutorials
- ANF2022 ROS2 Frejus : https://wiki.2rm.cnrs.fr/AnfRos2/Supports
- Q&A ROS : https://answers.ros.org/questions/
- ROS Developpers Podcast : https://www.theconstructsim.com/category/ros_developers_podcast/
- Discourse : https://discourse.ros.org/
  - A french version launched by CNRS : https://discourse.ros.org/c/local/france
- Conferences :
  - In France ROSCONfr : https://roscon.fr/
  - International ROSCON : https://roscon.ros.org/2023/
- A lot of books available : https://www.theconstructsim.com/ros-books/

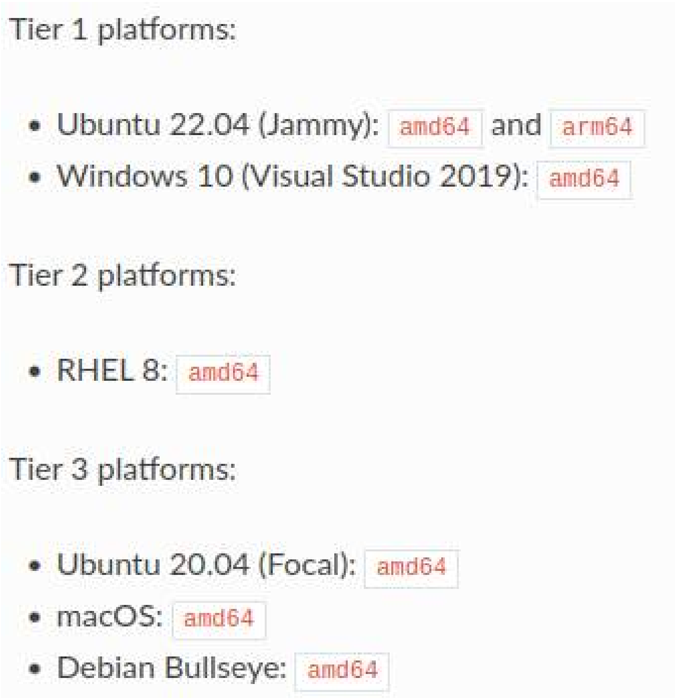
ROS versions (1)

- !! ROS1 vs ROS2 !!
  - http://design.ros2.org/articles/changes.html
  - Languages :
    - ROS1 : python2 and C++03
    - ROS2 : python3 and C++11 (C++14)
  - ROS2 has a network layer based on DDS (Data Distribution Service)
    - QoS support
    - Better performance, better reliability, better security
  - ROS2 : some support for real-time computation
    - https://ros-realtime.github.io/
    - https://github.com/ros-realtime

ROS version (2)

- ROS1 versions are aligned with Ubuntu distributions
  - Typically, LTS versions should be preferred
  - http://wiki.ros.org/Distributions
  - Version names : Melodic for Ubuntu 18.04 or **Noetic** on Ubuntu 20.04
- ROS2 is now the advised version to use for starting a new project

Humble supported on the following platforms :



Tier 1 platforms:

- Ubuntu 22.04 (Jammy): `amd64` and `arm64`
- Windows 10 (Visual Studio 2019): `amd64`

Tier 2 platforms:

- RHEL 8: `amd64`

Tier 3 platforms:

- Ubuntu 20.04 (Focal): `amd64`
- macOS: `amd64`
- Debian Bullseye: `amd64`

ROS in practice

- ROS components
- ROS handling via CLI
  - rosrun, roslaunch
  - rostopic for data messaging
  - rosservice et rosparam
- Node programming in ROS
  - Python or C++
  - Tf2

Nodes

- Nodes
  - A node is a simple component of a ROS application that does **one job** (one “main function”)
- Examples :
  - **Node A** computes the position (x,y) of the robot and its orientation
  - **Node B** reads the data coming from the lidar (to detect the relative position of obstacles)
  - **Node C** computes the trajectory the robots should follow, as a sequence of points (x_1,y_1), (x_2,y_2), …, (x_n,y_n)
  - **Node D** sends commands to the motors to move the robot
  - Etc.

The publish-subscriber pattern

- ROS Node ⇒ Process
  - Nodes produce and consume data
- A node that produces data is a **Publisher**
  - Each type of published data is a **Topic**
  - Every time some data is published in a *topic*, it’s a **message**
  - Example : a sensor node reads the robot position from the GPS receiver and publishes it into a « */gps_fix* » topic. Data transmitted are in format NavSatFix.
- A node that reads the data published in a *topic* is a **Subscriber**
  - Node can subscribe to *topics*
  - They will be notified every time a new *message* is produced
  - Example : a node needs the position of the robot to compute its trajectory : it subscribes to the « */gps_fix* » topic to be informed every time the position is updated. It will activate a callback function in which NavSatFix message with the latest position is passed in parameter.

Communication

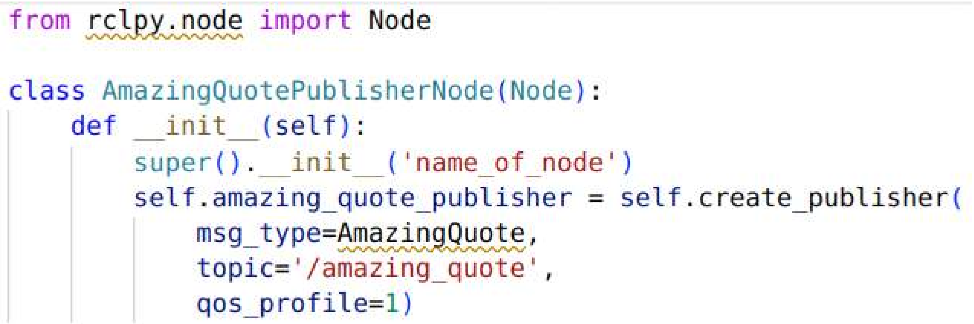
Communication is many-to-many

Any node can send a message to a topic

Any node can subscribe to a topic

- ROS1
  - Master node
    - Every distributed ROS application has a **Master node**
    - It keeps a list of all other nodes in the application and of the published topics
  - An application node
    - Must register itself and its topics into the master
    - To subscribe to a topics, it contacts the master node

- ROS2
  - An application node
    - Must register itself and its topics via the dedicated DDS service



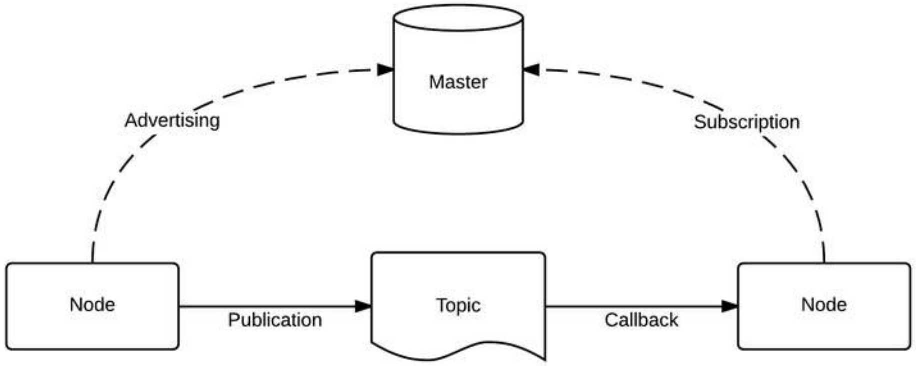
```
from rclpy.node import Node

class AmazingQuotePublisherNode(Node):
    def __init__(self):
        super().__init__('name_of_node')
        self.amazing_quote_publisher = self.create_publisher(
            msg_type=AmazingQuote,
            topic='/amazing_quote',
            qos_profile=1)
```

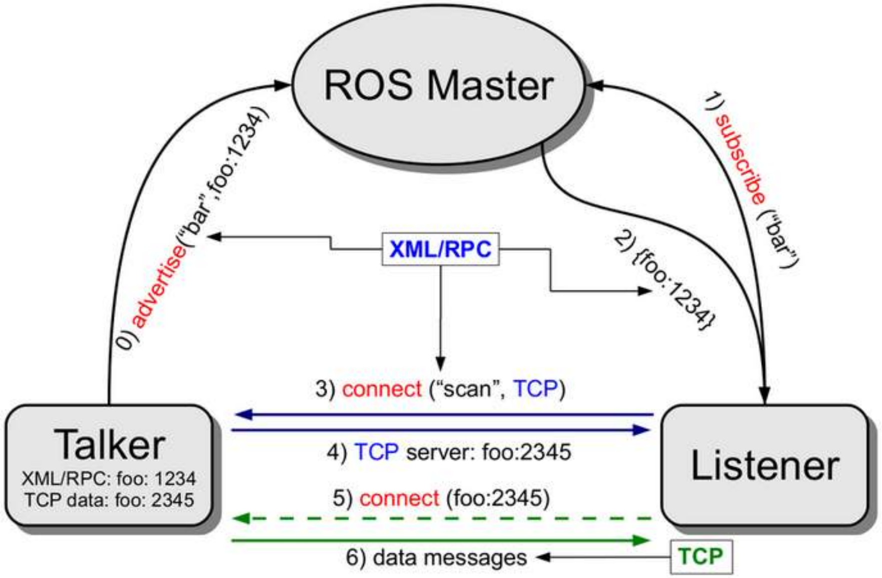
- After this it can subscribe to topics
- Quality of Service :

  https://docs.ros.org/en/rolling/Concepts/Intermediate/About-Quality-of-Service-Settings.html

Communication principles in ROS1 & ROS2

ROS1 with roscore (= master)





ROS 2 + DDS Interoperation from rti

rti Your systems. Working as one.

DDS Introduction

Neil Puthuff <neil@rti.com>
Software Integration Engineer, RTI

June 11, 2020

https://www.youtube.com/watch?v=GGqcrccWfeE

- **Participant Discovery Phase (PDP) :**
  - Connect nodes together
  - Use multicast protocol
- **Endpoint Discovery Phase (EDP) :**
  - Declaration of DataReaders and DataWriters
  - Use the PDP channels
- Existence of DDS Domain (network isolation) => improve security

More details :
https://fast-dds.docs.eprosima.com/en/latest/fastdds/discovery/discovery.html

Exercise 1 : Playing with ROS2 topics

- Goals : launch Turtlesim and Move the turtle
- https://docs.ros.org/en/foxy/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Topics/Understanding-ROS2-Topics.html

- Tips and tricks in the terminal (see next slides)
  - .bashrc configuration
  - Auto-completion
  - Choose a good terminal => terminator is life
    - Shortcuts

CTRL+MAJ+O	split horizontally
CTRL+MAJ+E	split vertically
CTRL+MAJ+W	close terminal
ALT+arrows	navigate between terminals

- VM : ubuntu20.04_ROS_training
  - User : ros
  - Passwd : hal9000

Tips: .bashrc configuration

- Create some useful aliases:

```
alias sb='source /home/gdherbom/.bashrc'
alias nb='nano /home/gdherbom/.bashrc'
alias noetic='source /opt/ros/noetic/setup.bash'
alias foxy='source /opt/ros/foxy/setup.bash'
```

- For ROS1, manage your master URI & ROS_IP:

```
# robot
export ROS_MASTER_URI=http://192.168.2.20:11311
export ROS_HOSTNAME=192.168.2.42
# localhost
#export ROS_MASTER_URI=http://127.0.0.1:11311
#export ROS_HOSTNAME=127.0.0.1
```

You can switch easily between 2 networks configuration by commenting/uncommenting the 2 lines.

Tips: Auto completion in ROS2

Example to generate an occupancy grid in one command line

- Start typing the command in the terminal and double “tab”:

```
ros2 topic pub -r 10 /my_topic
Display all 220 possibilities? (y or n)
```

- We know that occupancy grid is in **nav_msgs**:

```
ros2 topic pub -r 10 /my_topic nav_msgs/msg/
nav_msgs/msg/GridCells      nav_msgs/msg/MapMetaData    nav_msgs/msg/OccupancyGrid  nav_msgs/msg/Odometry       nav_msgs/msg/Path
```

- After the command, open a double quote and double “tab”:

```
ros2 topic pub -r 10 /my_topic nav_msgs/msg/OccupancyGrid "
-1
header:^J  stamp:^J    sec: 0^J    nanosec: 0^J  frame_id: ''^Jinfo:^J  map_load_time:^J    sec: 0^J    nanosec: 0^J  resolution:
0.0^J  width: 0^J  height: 0^J  origin:^J    position:^J      x: 0.0^J      y: 0.0^J      z: 0.0^J    orientation:^J      x: 0.0^J
   y: 0.0^J      z: 0.0^J      w: 1.0^Jdata: []
--keep-alive
-n
--node-name
--once
-p
--print
--qos-depth
--qos-durability
--qos-history
--qos-profile
--qos-reliability
-r
--rate
-t
--times
```

Reminder about terminal shortcuts:

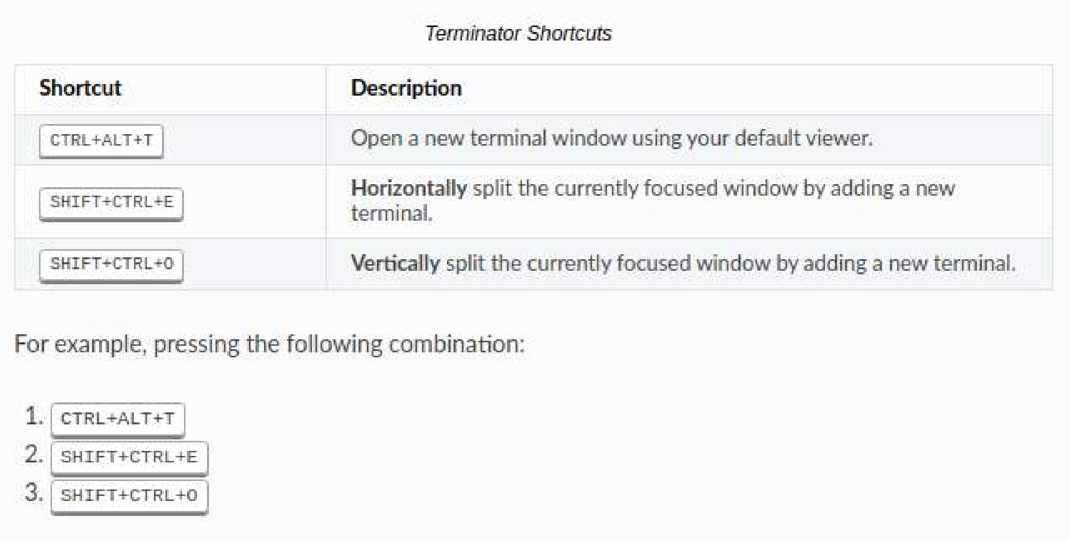
- CTRL**+MAJ**+C : copy text
- CTRL**+MAJ**+V : paste text
- CTRL+R : recursive search in history

To have auto-completion in ROS 2 :
apt install python3-argcomplete

- Complete with the starting letter of your message, here “h”, double “tab” and it is done !!

Tips : Terminator is life

https://ros2-tutorial.readthedocs.io/en/latest/terminator.html#terminator-is-life

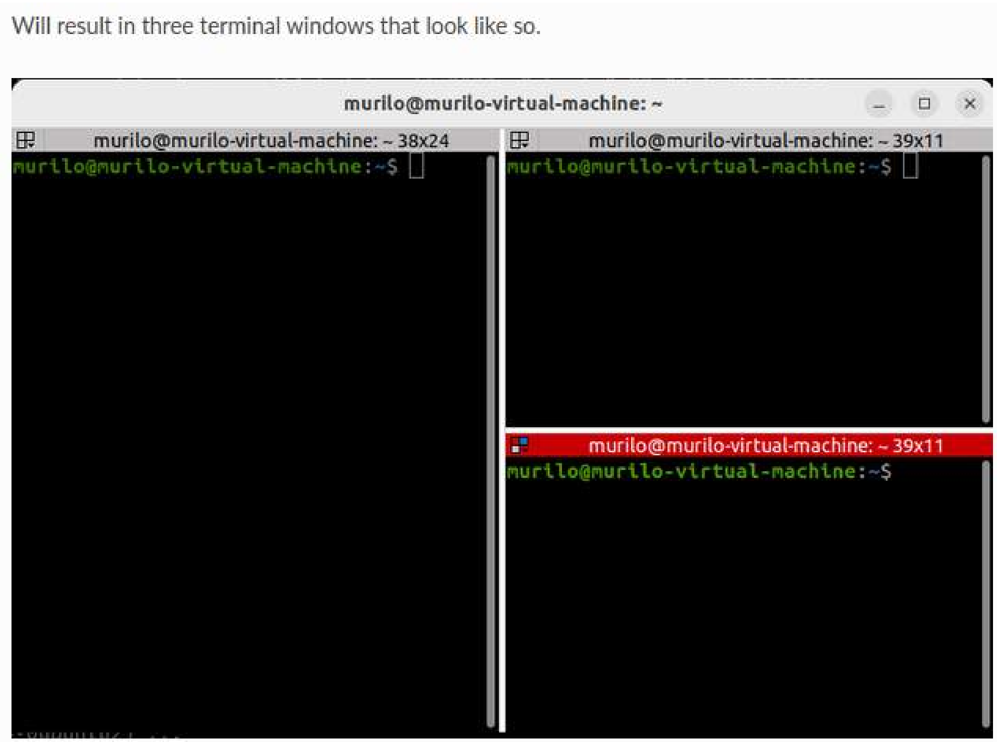


Terminator Shortcuts

Shortcut	Description
CTRL+ALT+T	Open a new terminal window using your default viewer.
SHIFT+CTRL+E	**Horizontally** split the currently focused window by adding a new terminal.
SHIFT+CTRL+O	**Vertically** split the currently focused window by adding a new terminal.

For example, pressing the following combination:

1. CTRL+ALT+T
2. SHIFT+CTRL+E
3. SHIFT+CTRL+O



Will result in three terminal windows that look like so.

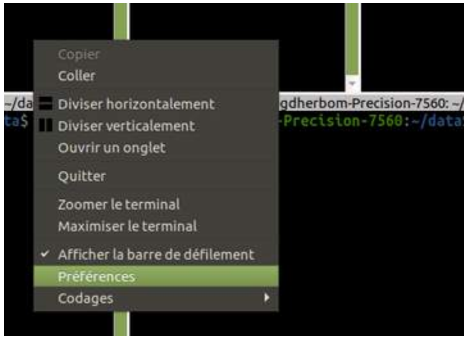
```
murilo@murilo-virtual-machine: ~
murilo@murilo-virtual-machine: ~ 38x24
murilo@murilo-virtual-machine:~$
murilo@murilo-virtual-machine: ~ 39x11
murilo@murilo-virtual-machine:~$
murilo@murilo-virtual-machine: ~ 39x11
murilo@murilo-virtual-machine:~$
```

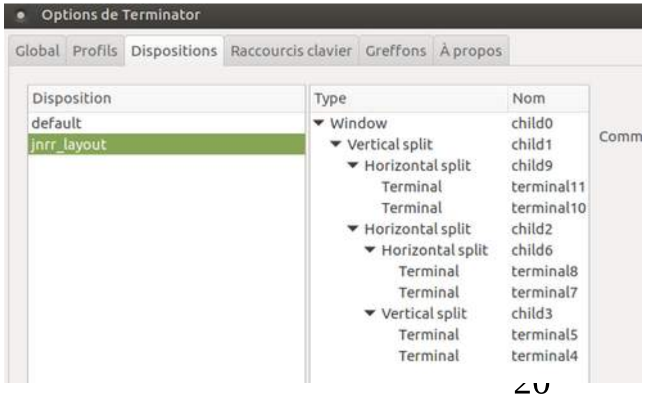
Manual : `man terminator` or `terminator -h`

Useful parameters for automate the launch of ROS nodes:

Open a new tab : `terminator -new-tab`

Open with a preconfigured layout :
`terminator --layout=LAYOUT`





Disposition
default
jnrr_layout

Type	Nom
Window	child0
Vertical split	child1
Horizontal split	child9
Terminal	terminal11
Terminal	terminal10
Horizontal split	child2
Horizontal split	child6
Terminal	terminal8
Terminal	terminal7
Vertical split	child3
Terminal	terminal5
Terminal	terminal4

Gerald Dherbomez
gerald.dherbomez@univ-lille.fr

Exercise 2 : How to read data coming from ROS1 in ROS2?

- Solution : ros1_bridge
- Practical application on real data from Zoé car
  - In folder */home/ros/data* launch the replay of data:

`noetic` (and don't forget to launch the master before with `roscore`)

```
rosbag play -l jnrr_data_cristal_zoe.bag
```

  - In another terminal we can see data in ROS1 but not in ROS2

`rostopic list` vs `ros2 topic list`

- Next steps : testing some concepts of ROS1

Don't forget to `source` the correct ROS environment (`noetic` or `foxy`) when you open a terminal

Ex2.1/ ROS1 : Network Configuration

- Need to specify the address of the master node, so the other nodes can communicate
  - Address : IP + Port
- One single PC (master and nodes on the same system)
  - `export ROS_HOSTNAME=localhost`
  - `export ROS_MASTER_URI=http://localhost:11311`
- Network of PCs
  - First, verify all PC are on the same local network (typically addresses start with 192.168.1.xxx)
  - On the Master PC
    - `export ROS_HOSTNAME=192.168.1.10`
    - `export ROS_MASTER_URI=http://192.168.1.10:11311`
  - On the other PCs
    - `export ROS_HOSTNAME=192.168.1.n`
    - `export ROS_MASTER_URI=http://192.168.1.10:11311`
  - Save these in the respective .bashrc files for automatic configuration

EX2.2/ rostopic

- The rostopic programs allows to read/write a topic
- Here are some example of commands :
- Publishing messages : `rostopic pub`
  - `rostopic pub /mytopic std_msgs/String "data: 'test'"`
  - Mode latching : -l ⇒ keep data available to a subscriber even it comes after the publication (default mode). CTRL+C to quit.
  - use -1 or –once to avoid blocking. Quit after 3 sec.
  - Periodic publication : -r
- List existing messages : `rostopic list`
- Subscribing to a topic: `rostopic echo`
  - `rostopic echo /mytopic`
- Check that you are not able to display CAN data (because you don’t know the format) :

```
rostopic echo /can/speed
```

ERROR: Cannot load message class for [can_zoe_msgs/Speed]. Are your messages built?

EX2.3/ create a ROS1 package and build it

- ROS messages created for a specific project are called **Custom Messages** and must be built to be used.
- build system : catkin_make ⇒ mix of cmake and python scripts
- Create a ROS 1 workspace:
  - Create a top folder for exercise 2 : `mkdir jnrr_ex2` and go in it: `cd jnrr_ex2`
  - Create the ROS1 ws : `mkdir -p ros1_ws/src` then go to the created folder cd `ros1_ws/src` and clone the repos :
  - Go to https://gitlab.cristal.univ-lille.fr/open-pretil/zoe_msgs/white_zoe/can_zoe_msgsand clone the git repository
  - Come back to the root of ws : `cd ..` and build : `catkin_make_isolated --install`
- The directory tree

```
jnrr_ex2
└── ros1_ws
    ├── build_isolated
    ├── devel_isolated
    ├── install_isolated
    └── src
```

Visualize the data:

```
cd jnrr_ex2/ros1_ws
source install_isolated/setup.bash
rostopic echo /can/speed
```

EX2.4/ ROS2 messages package and ros1_bridge

- In ROS 2 the build tool is colcon
- Create a ROS 2 workspace in the top folder:
  - `cd jnrr_ex2`
  - Create the ROShttps://github.com/ros2/ros1_bridge/blob/master/doc/index.rst2 ws : `mkdir -p ros2_ws/src` then go to the created folder cd `ros2_ws/src` and clone the repos :
  - Go to https://gitlab.cristal.univ-lille.fr/open-pretil/zoe_msgs/white_zoe_ros2/can_zoe_msgs and clone the git repository
  - Come back to the root of ws : `cd ..` and build : `colcon build`
- Now we will build the bridge between ROS1 and ROS2:
  - Go to the top folder `cd jnrr_ex2` and clone the ros1_bridge
  - `git clone -b foxy https://github.com/ros2/ros1_bridge.git`
  - **Warning** : clone the branch corresponding to your ROS2 version or git checkout to it.
  - How to build: you must sources 4 files : ROS1 setup, ROS2 setup, ROS1 custom messages workspace and ROS2 messages workspace: ajouter package et cmakelists

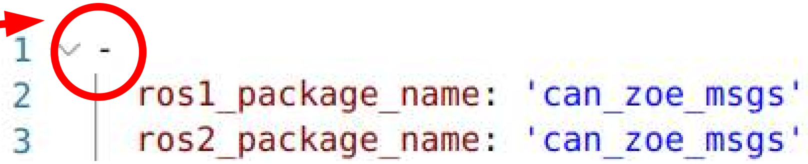
```
noetic
foxy
source ../ros1_ws/install_isolated/setup.bash
source ../ros2_ws_install/setup.bash
```

  - And finally build : `colcon build`

Doc : https://github.com/ros2/ros1_bridge/blob/master/doc/index.rst

EX2.5/ Running the ros1_bridge

- The yaml file `my_bridge.yaml`, the contents must be a list
- You can specify translations between messages in packages.



```yaml
-
  ros1_package_name: 'can_zoe_msgs'
  ros2_package_name: 'can_zoe_msgs'
```

- To know if your messages are known: `ros2 run ros1_bridge dynamic_bridge --print-pairs | grep can`

The final directory tree:

```
└── jnrr_ex2
    ├── ros1_bridge
    │   ├── bin
    │   ├── build
    │   ├── CHANGELOG.rst
    │   ├── cmake
    │   ├── CMakeLists.txt
    │   ├── CONTRIBUTING.md
    │   ├── doc
    │   ├── include
    │   ├── install
    │   ├── LICENSE
    │   ├── log
    │   ├── package.xml
    │   ├── README.md
    │   ├── resource
    │   ├── ros1_bridge
    │   ├── src
    │   └── test
    ├── ros1_ws
    │   ├── build_isolated
    │   ├── devel_isolated
    │   ├── install_isolated
    │   └── src
    └── ros2_ws
        ├── build
        ├── install
        ├── log
        └── src
```

To run the bridge:

```
Cd jnrr_ex2/ros1_bridge
source install/setup.bash
ros2 run ros1_bridge dynamic_bridge --bridge-all-topics
```

Visualize the data, in a new terminal:

```
cd jnrr_ex2/ros2_ws
source install
rostopic echo /can/speed
```

Playing with ROS2 topics: “see 2.2.7 Topic” of https://wiki.2rm.cnrs.fr/AnfRos2/Supports?action=AttachFile&do=view&target=anf-2022-polycopie.pdf

Summary at mid-term

- What did you learn ?
  - Overview of the ROS ecosystem, links to documentation, how to ask the community
  - How to manipulate topics in ROS1 and ROS2
  - How to configure your environment
  - How to build a ROS1 and a ROS2 workspaces
  - How to run ROS1 and ROS2 nodes together and implement gateway between the 2 versions
- You have a VM to test all these concepts easily without own installation

Exercise 3 : Tools for ROS2

- Several tools are available natively :
  - **rqt_graph** : connection graph, shows node communication
    - `ros2 run rqt_graph rqt_graph`
  - **rqt_plot** : plot curves
    - `foxy`
    - `source install/setup.bash`
    - `ros2 run rqt_plot rqt_plot` ==> display speed of front left wheel, put in topic field : /can/speed/wheel_speed_fl
  - **rviz2** : to visualize data from sensors and information about the robot
    - `ros2 run rviz2 rviz2`
    - Display data of camera (/camera/color/image_raw topic) and LIDAR. Click on Add button, choose By topic.
    - Tips: change the “Fixed Frame” field in Global Options to fit your sensor. For LIDAR, use jnrr_data_cristal_zoe_lidar.bag
  - **ROS1 : rqt_bag**, to visualise data recorded on disk, needs the bagfile as a parameter
    - rqt_bag xxx.bag
- PlotJuggler
  - Installation : `sudo apt install ros-foxy-plotjuggler ros-foxy-plotjuggler-ros`
  - Objectives :
    - display time series data, /can/speed/wheel_speed_fl for example
    - Display 2D data, `ros2 topic echo /gps_novatel/fix`

Exercise 4 : programming a ROS2 publisher / subscriber

- Creating ROS2 package

  https://docs.ros.org/en/foxy/Tutorials/Beginner-Client-Libraries/Creating-Your-First-ROS2-Package.html

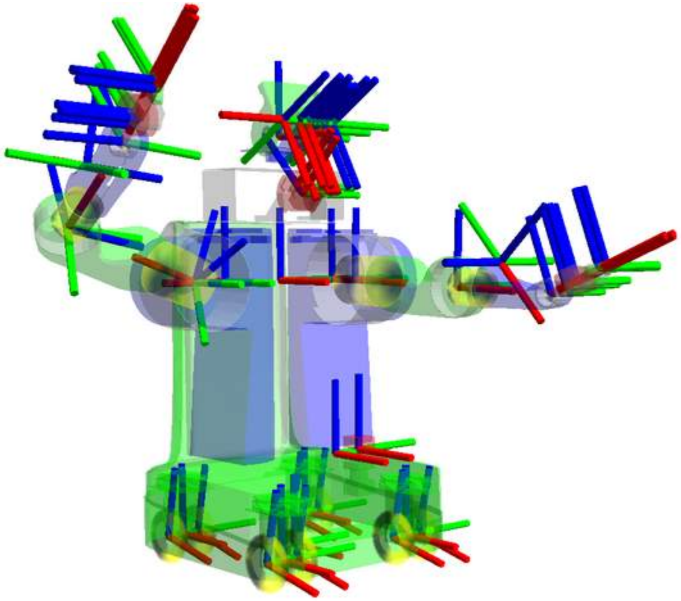
- Writing a simple publisher and subscriber (C++)

  https://docs.ros.org/en/foxy/Tutorials/Beginner-Client-Libraries/Writing-A-Simple-Cpp-Publisher-And-Subscriber.html

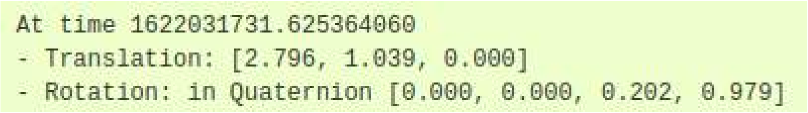
- Writing a simple publisher and subscriber (Python)

  https://docs.ros.org/en/foxy/Tutorials/Beginner-Client-Libraries/Writing-A-Simple-Py-Publisher-And-Subscriber.html

Exercise 5 tf2 in ROS2



- **What is tf2 ?**
  - tf2 is a library that offers you an easy way to create and manipulate coordinate frames.



```
At time 1622031731.625364060
- Translation: [2.796, 1.039, 0.000]
- Rotation: in Quaternion [0.000, 0.000, 0.202, 0.979]
```

- ROS2 official tutorial for tf2: https://docs.ros.org/en/foxy/Tutorials/Intermediate/Tf2/Introduction-To-Tf2.html
- Questions :
  - 1. Follow the previous tutorial in order to create a node that will publish the tf of the turtle
  - 2. In rviz, display the TFs and thanks to the tf2_tools node display the frames architecture
  - 3. In the ROS2 node, add an additional TF that display the position of the turtle in the past (1 second of delay)

https://gitlab.cristal.univ-lille.fr/open-pretil/pretil-tutorials/jnrr_tf2

Q&A

ROS.org

Discussions