Universität Bremen

Institute for Artificial Intelligence

Faculty 03

Mathematics &

Computer Science

Robot Programming with ROS

3. Robots and Communication

Arthur Niedźwiecki

30th April. 2025

Artificial Intelligence

Overview

Overview

Industrial Robots

Logistics

Image courtesy: BIBA

Automotive

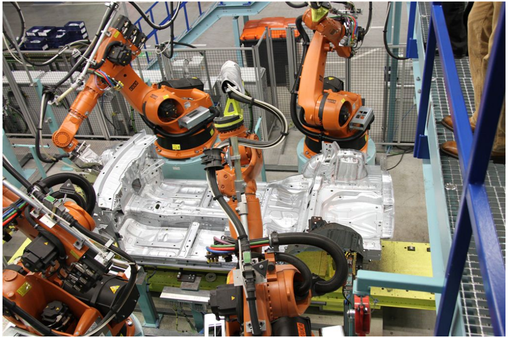


Image courtesy: Mercedes Benz Bremen

- Extremely heavy, precise and dangerous, not really smart
- Mostly no sensors, only high-precision motor encoders
- Programmable through PLCs (using block diagrams or Pascal / Basic like languages)

Industrial Light-weight Robots

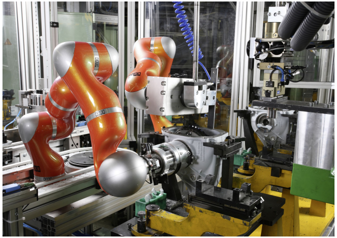
Production:

Copyright: Universal Robots

Medicine:

Copyright: Intuitive Surgical

Automotive:



Copyright: KUKA Roboter GmbH

- Very precise, moderately dangerous, somewhat smart
- High-precision motor encoders, sometimes force sensors, cameras
- Native programming and simulation tools (C++, Java, Python, GUIs)

Service Robots

Autonomous aircrafts

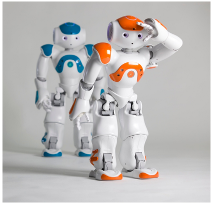
Courtesy DJI

Mobile platforms

Courtesy NASA/JPL-Caltech

Manipulation platforms

Humanoids



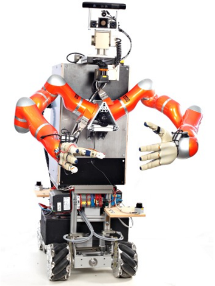
- Usually not very precise
- Not really dangerous
- Usually cognition-enabled
- Equipped with lots of sensors
- Usually running Linux

Service Robots with Light-weight Arms

DLR Justin

Courtesy of DLR

TUM Rosie

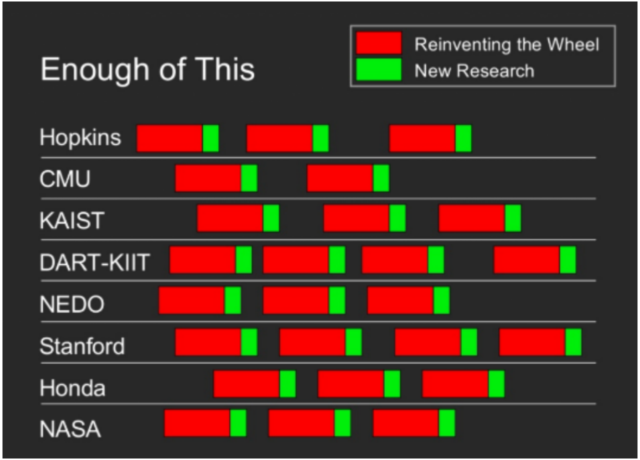


- Moderately precise and dangerous
- Cognition-enabled
- Equipped with lots of sensors
- Usually running a combination of a real-time and non real-time OS.

Overview

1. What is a Robot?
2. ROS
   - ROS Overview
   - ROS Build System
   - ROS Communication Layer
3. Organizational

Motivation



Reinventing the wheel

Motivation

- Numerous different robotics labs, each with their own robot platforms, different operating systems and programming languages but similar software and hardware modules for most of them.

Motivation

- Numerous different robotics labs, each with their own robot platforms, different operating systems and programming languages but similar software and hardware modules for most of them.
- Each lab reinventing the wheel for their platforms.

Motivation

- Numerous different robotics labs, each with their own robot platforms, different operating systems and programming languages but similar software and hardware modules for most of them.
- Each lab reinventing the wheel for their platforms.
- **Idea**: provide a unified software framework for everyone to work with. Requirements:

Motivation

- Numerous different robotics labs, each with their own robot platforms, different operating systems and programming languages but similar software and hardware modules for most of them.
- Each lab reinventing the wheel for their platforms.
- **Idea**: provide a unified software framework for everyone to work with. Requirements:
  - Support for different programming languages

Motivation

- Numerous different robotics labs, each with their own robot platforms, different operating systems and programming languages but similar software and hardware modules for most of them.
- Each lab reinventing the wheel for their platforms.
- **Idea**: provide a unified software framework for everyone to work with. Requirements:
  - Support for different programming languages
  - Different operating systems

Motivation

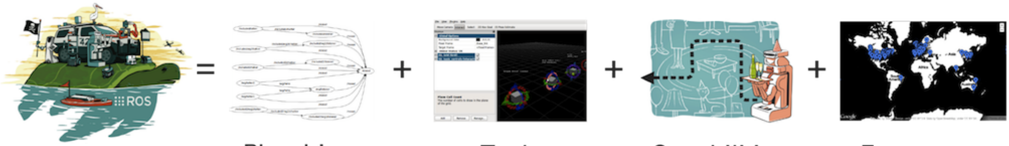
- Numerous different robotics labs, each with their own robot platforms, different operating systems and programming languages but similar software and hardware modules for most of them.
- Each lab reinventing the wheel for their platforms.
- **Idea**: provide a unified software framework for everyone to work with. Requirements:
  - Support for different programming languages
  - Different operating systems
  - Distributed processing over multiple computers / robots

Motivation

- Numerous different robotics labs, each with their own robot platforms, different operating systems and programming languages but similar software and hardware modules for most of them.
- Each lab reinventing the wheel for their platforms.
- **Idea**: provide a unified software framework for everyone to work with. Requirements:
  - Support for different programming languages
  - Different operating systems
  - Distributed processing over multiple computers / robots
  - Easy software sharing mechanisms

Robot Operating System

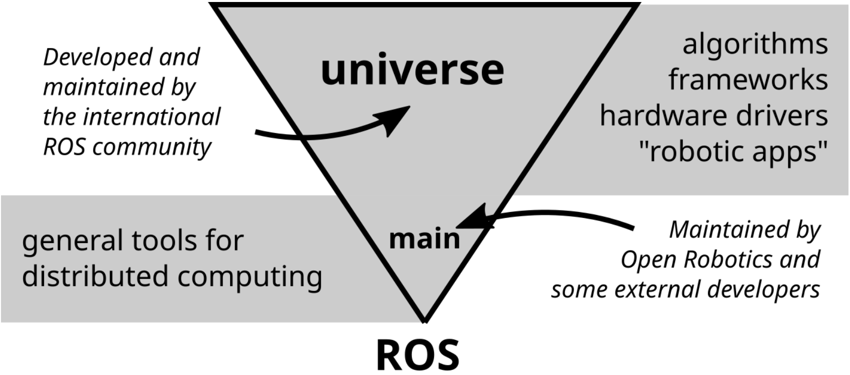
At 2007 Willow Garage, a company founded by an early Google employee Scott Hassan at 2006 in the Silicon Valley, starts working on their Personal Robotics project and ROS.



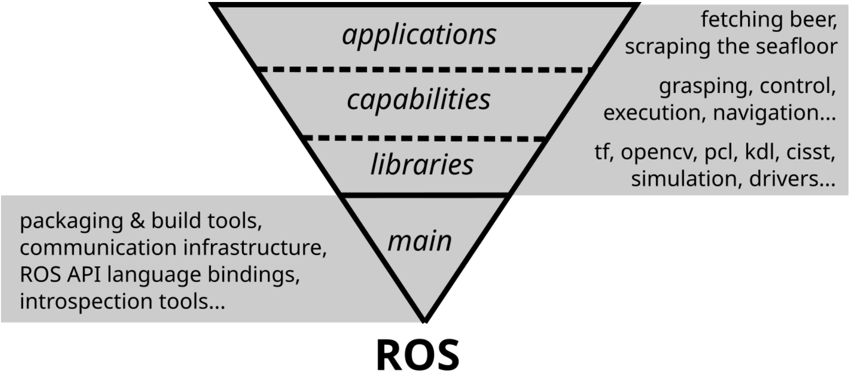
Plumbing | Tools | Capabilities | Ecosystem

2013 the Open Source Robotics Foundation (OSRF) takes over
2014 NASA announces first robot to run ROS: Robonaut 2
2017 OSRF changes name to Open Robotics
2018 Microsoft ports to Windows, AWS releases RoboMaker

Robot Operating System [2]

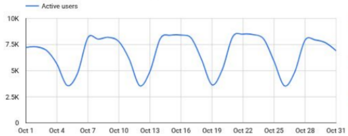


Robot Operating System [3]



ROS Community Report - Docs users

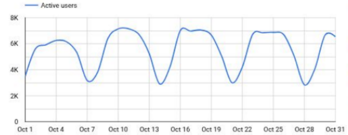
docs.ros.org October 2023 vs. October 2024



	G4 DATE	Active users
1.	Oct 1, 2024	7,223
2.	Oct 2, 2024	7,287
3.	Oct 3, 2024	6,936
4.	Oct 4, 2024	5,645
5.	Oct 5, 2024	3,588
6.	Oct 6, 2024	4,747
7.	Oct 7, 2024	8,160
	Grand total	115,506

1 - 31 / 31

Active users
115,506



	G4 DATE	Active users
1.	Oct 1, 2023	3,487
2.	Oct 2, 2023	5,612
3.	Oct 3, 2023	5,909
4.	Oct 4, 2023	6,251
5.	Oct 5, 2023	6,154
6.	Oct 6, 2023	5,319
7.	Oct 7, 2023	3,174
	Grand total	93,340

1 - 31 / 31

Active users
93,340

docs.ros.org

ROS 2 documentation users increased by **+23.75%**

ROS™

ROS Community Report - Distributions dowloaded

Distros by Year /

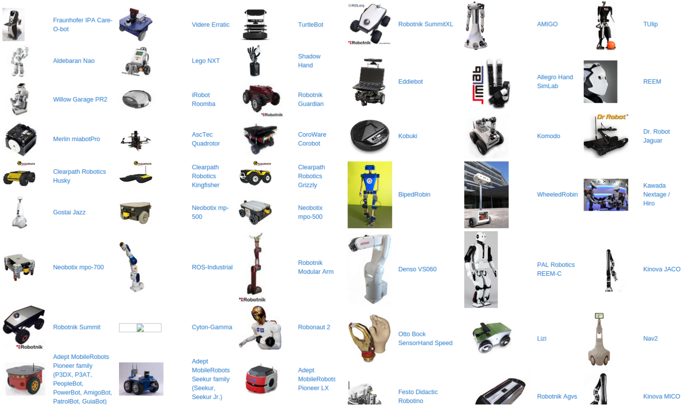
Distro	October 2024	October 2023	YoY Change
Melodic	2.32%	5.66%	-3.34
Noetic	22.18%	30.51%	-8.33
Foxy	3.58%	6.37%	-2.79
Galactic	1.25%	2.33%	-1.08
Humble	39.38%	32.79%	+6.59
Iron	5.26%	4.97%	+0.29
Jazzy	9.36%	N/A	+9.36
Rolling	4.90%	4.82%	+0.08

Package downloads, by distro, as a percentage of all downloads from the ROS servers in October of 2023, and 2024.

ROS™

ROS Wiki - Robots

Just a few example robots supporting ROS:



ROS Build System

Packages and Metapackages

- *Packages* are a named collection of software that is built and treated as an atomic dependency in the ROS build system.
- *Metapackages* are dummy “virtual” packages that reference one or more related packages which are loosely grouped together

Similar to Debian packages.
Actually released through the Debian packaging system.

The following examples are inspired by the official documentation:
`https://docs.ros.org/en/jazzy/Tutorials/Beginner-Client-Libraries.html`

ROS Workspace build with colcon

`$ source /opt/ros/jazzy/setup.bash`

Packages are stored in ROS workspaces. Create a workspace:

`$ mkdir -p  /ros_ws/src`

`$ cd  /ros_ws/src`

- Get an example package:

  `$ git clone https://github.com/ros/ros_tutorials.git -b jazzy`

- Install dependencies with *rosdep*:

  `$ cd ..`

  `$ rosdep install -i --from-path src --rosdistro jazzy -y`

- Build the workspace:

  `$ colcon build`

  `$ ls` → `build install log src`

- Source your workspace:

  `$ ros2 pkg prefix turtlesim` → /opt/ros/jazzy

  `$ source install/local_setup.bash` → *local* overlay on top of *jazzy/setup.bash*

  `$ ros2 pkg prefix turtlesim` → /home/<user>/ros_ws/install/turtlesim

  `$ ros2 run turtlesim turtlesim_node` → runs the local code

ROS Workspace for a CMAKE package

Workspaces have a specific structure

```
├── build (contains binaries/executables)
├── install (to link libraries)
├── log
└── src
    └── ros_tutorials (CMAKE package)
        ├── .git
        └── turtlesim
            ├── package.xml
            ├── CMakeLists.txt
            ├── include
            │   └── turtlesim
            │       └── ...
            ├── src
            │   └── ...
            └── ...
```

ROS Workspace Create new Package

```
$ cd  /ros_ws/src
```

- Create a package with **ament_python**:
  ```
  $ ros2 pkg create --build-type ament_python --license Apache-2.0 --node-name my_node my_package
  ```
- Compiling a package:
  ```
  $ cd ..  && colcon build
  ```
- Update ROS filesystem for new package:
  ```
  $ source install/local_setup.bash
  ```
- Run the new node
  ```
  $ ros2 run my_package my_node
  ```

Naming convention: underscores (no CamelCase, no-dashes)!

→ Multiple workspaces chained together.
→ source *local_setup.bash* for additive overlay or *setup.bash* for global overwrite

ROS Workspace for a PYTHON package

Workspaces have a specific structure

```
├── build
├── install
├── log
└── src
    ├── ros_tutorials (CMAKE)
    │       └── turtlesim
    │               ├── package.xml (meta information)
    │               ├── CMakeLists.txt (install/execute instructions)
    │               ├── include (public header files)
    │               └── src (source code)
    └── my_package (PYTHON)
            ├── package.xml (meta information)
            ├── setup.cfg (package executables)
            ├── setup.py (install instructions)
            ├── my_package (for __init__.py)
            └── resource/my_package (marker file)
```

Package.xml - Define dependencies for rosdep

my_package/package.xml

```
<?xml version="1.0"?>
<?xml-model href="http://download.ros.org/schema/package_format3.xsd" schematypens="
    http://www.w3.org/2001/XMLSchema"?>
<package format="3">
        <name>my_package</name>
        <version>0.0.0</version>
        <description>TODO: Package description</description>
        <maintainer email="aniedz@cs.uni-bremen.de">arthur</maintainer>
        <license>Apache-2.0</license>
        <test_depend>ament_copyright</test_depend>
        <test_depend>ament_flake8</test_depend>
        <test_depend>ament_pep257</test_depend>
        <test_depend>python3-pytest</test_depend>
        <export>
                <build_type>ament_python</build_type>
        </export>
</package>
```

setup.py - Define entry points in the package

my_package/setup.py

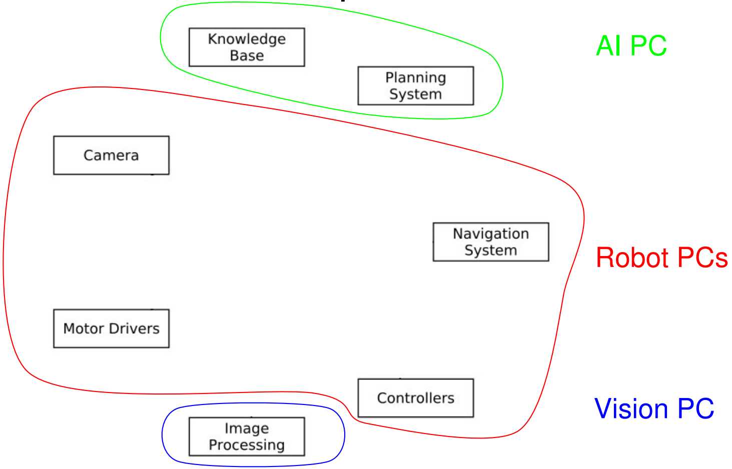
```python
from setuptools import find_packages, setup

package_name = 'my_package'
setup(
    ...
    entry_points={
        'console_scripts': [
        'my_node = my_package.my_node:main'
        ],
    },
)
```

Add subscriber and listener as executables:
https://docs.ros.org/en/jazzy/Tutorials/Beginner-Client-Libraries/Writing-A-Simple-Py-Publisher-And-Subscriber.html

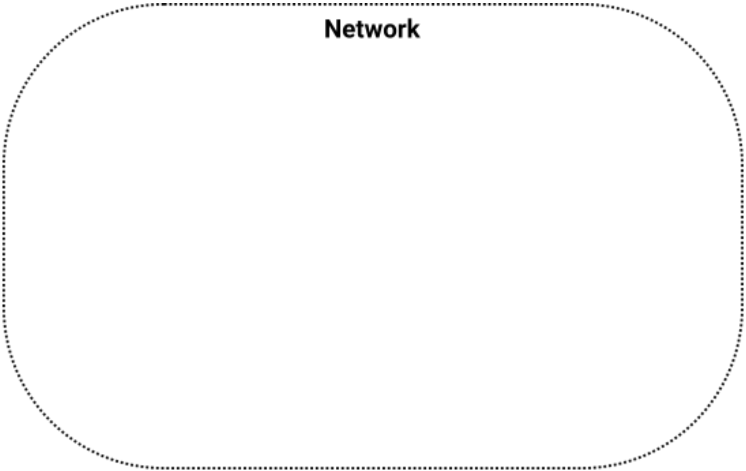
ROS Communication Layer

Robotic software components

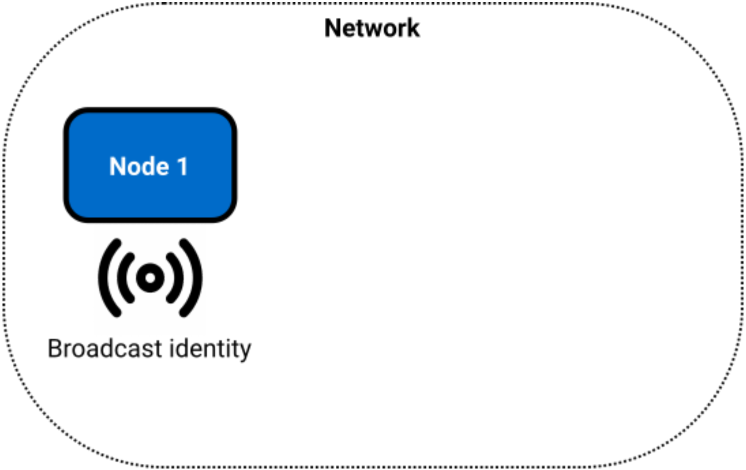


→ Processes distributed all over the place.

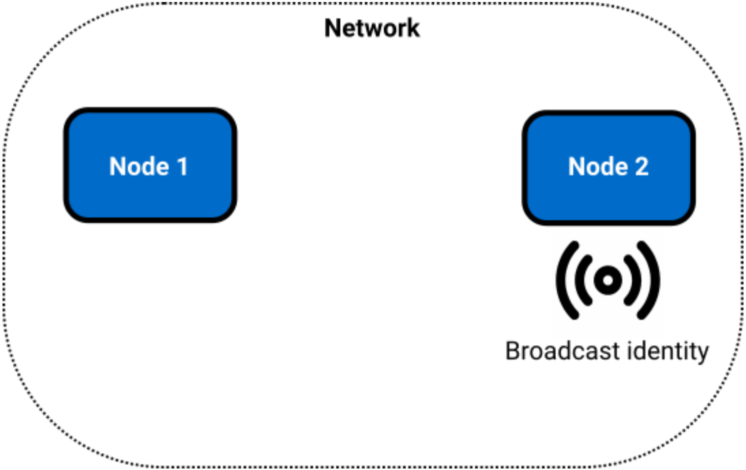
Node Discovery



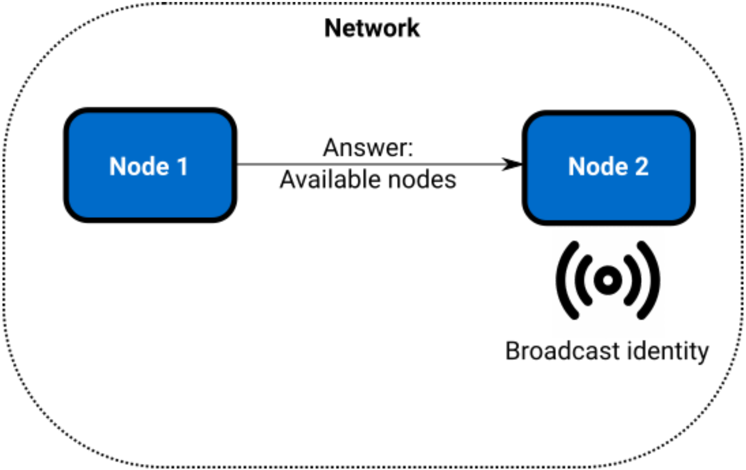
Node Discovery



Node Discovery



Node Discovery



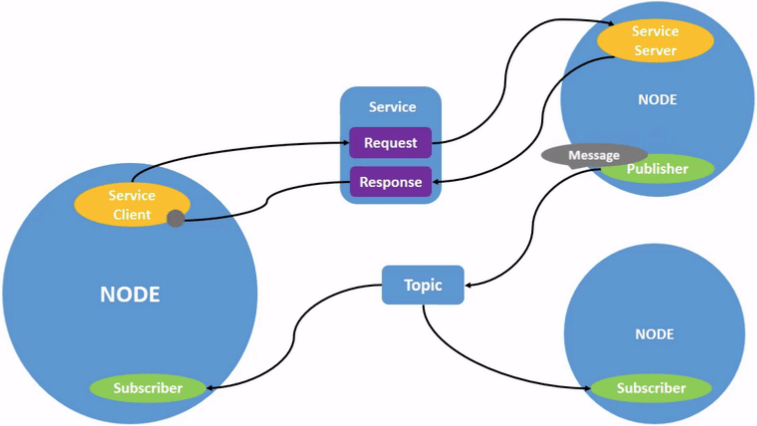
Terminology

- **Nodes** are processes that produce and consume data
- **Parameters** are persistent data stored on parameter server, e.g. configuration and initialization settings

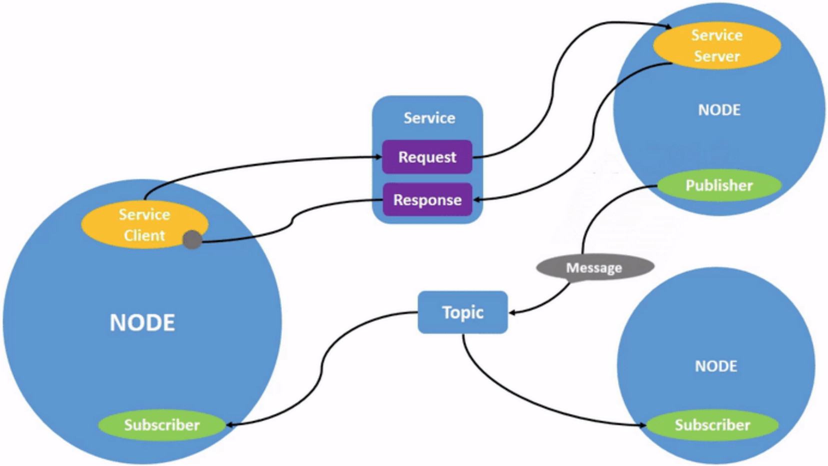
Node communication means:

- **Topics**: asynchronous many-to-many "streams-like"
  - Strongly-typed (ROS .msg spec), see `$ ros2 interface list`
  - Can have one or more *publishers*
  - Can have one or more *subscribers*
- **Services**: synchronous blocking one-to-many "function-call-like"
  - Strongly-typed (ROS .srv spec)
  - Can have only one *server*
  - Can have one or more *clients*
- **Actions**: asynchronous non-blocking one-to-many "function-call-like"
  - Built on top of topics but can be canceled

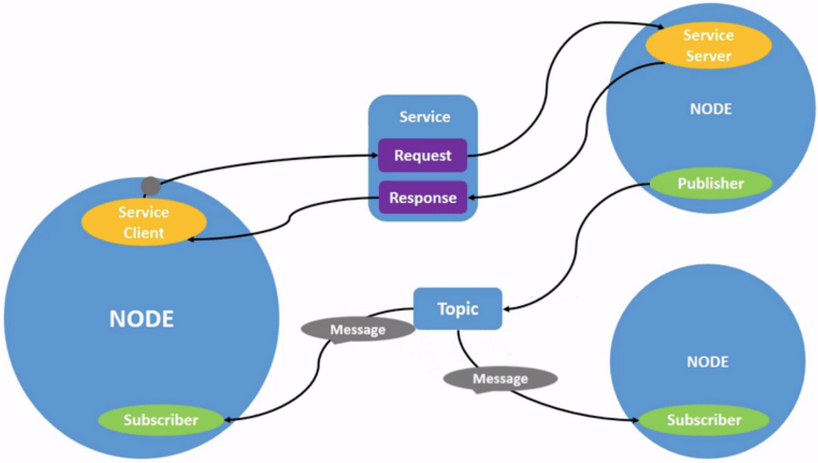
Node communication



Node communication [2]



Node communication [3]



Tools

- `ros2 node`: gives the user information about a node

  `$ ros2 node -h`

  `info, list`

- `ros2 topic`: gives publishers, subscribes to the topic, datarate, the actual data

  `bw, delay, echo, find, hz, info, list, pub, type`

- `ros2 service`: enables a user to call a ROS Service from the command line

  `call, echo, find, info, list, type`

- `ros2 interface`: gives information about message types

  `list, package, packages, proto, show`

- `ros2 wtf`: diagnoses problems with a ROS network
- more with `ros2 -h`

ROS Graph

- Start the turtle simulation:

```
$ ros2 run turtlesim turtlesim_node
```

- Start teleoperation:

```
$ ros2 run turtlesim turtle_teleop_key
```

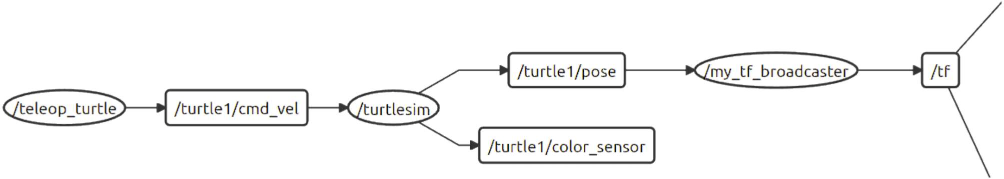
- Publish turtle1 position to /tf:

```
$ ros2 run turtle_tf2_py turtle_tf2_broadcaster --ros-args -p turtlename:=turtle1
```

See also `https://design.ros2.org/articles/ros_command_line_arguments.html`

- Examining the ROS Graph:

```
$ rqt_graph
```



Launch Files

See the docs:
https://docs.ros.org/en/jazzy/Tutorials/Intermediate/Launch/Creating-Launch-Files.html
Config files for launching nodes:

- as XML, YAML or Python
- automatically set parameters and start nodes with a single file
- hierarchically compose collections of launch files
- automatically re-spawn nodes if they crash
- change node names, namespaces, topics, and other resource names
- without recompiling
- easily distribute nodes across multiple machines

Launch Files [2]

launch/turtlesim_mimic_launch.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<launch>
    <node pkg="turtlesim" exec="turtlesim_node" name="sim" namespace="turtlesim1" />
    <node pkg="turtlesim" exec="turtlesim_node" name="sim" namespace="turtlesim2" />
    <node pkg="turtlesim" exec="mimic" name="mimic">

    <remap from="/input/pose" to="/turtlesim1/turtle1/pose" />
    <remap from="/output/cmd_vel" to="/turtlesim2/turtle1/cmd_vel" />
  </node>
</launch>
```

Launch Files [2]

launch/turtlesim_mimic_launch.yaml

```yaml
%YAML 1.2
---
launch:
    - node:
        pkg: "turtlesim"
        exec: "mimic"
        name: "mimic"
        remap:
            - from: "/input/pose"
              to: "/turtlesim1/turtle1/pose"
            - from: "/output/cmd\_vel"
              to: "/turtlesim2/turtle1/cmd\_vel"
    - node:
        pkg: "turtlesim"
        exec: "turtlesim_node"
        name: "sim"
        namespace: "turtlesim1"
    - node:
```

Launch Files [2]

launch/turtlesim_mimic_launch.py

```python
from launch import LaunchDescription
from launch_ros.actions import Node

def generate_launch_description():
    return LaunchDescription([
        Node(
            package='turtlesim',
            executable='mimic',
            name='mimic',
            remappings=[
                ('/input/pose', '/turtlesim1/turtle1/pose'),
                ('/output/cmd_vel', '/turtlesim2/turtle1/cmd_vel'),
            ]
        ),
        Node(
            package='turtlesim',
            namespace='turtlesim1',
            executable='turtlesim_node',
```

ROS API

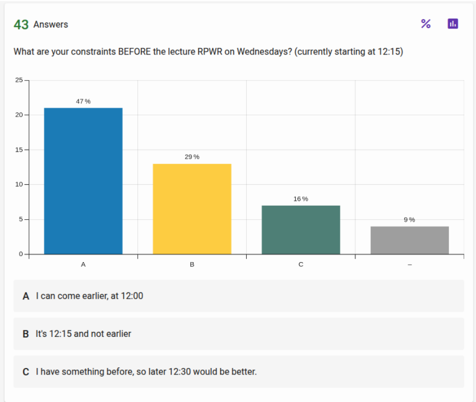
ROS API provides the programmer with means to

- start ROS node processes
- generate messages
- publish and subscribe to topics
- start service servers
- send service requests
- provide and query action services
- find ROS packages
- ...

ROS APIs: `rclcpp`, `rclpy`

Overview

Lecture Time - Begin Class

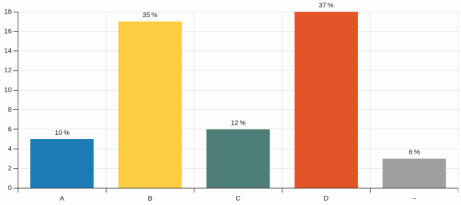


1/2 can start earlier.

Lecture Time - Lunch Break

43 Answers

How is it in the middle, in between Lecture and Assignments?



A I need a long break of 1 hour

B I can do lunch in 45 min

C 30min is enough for me

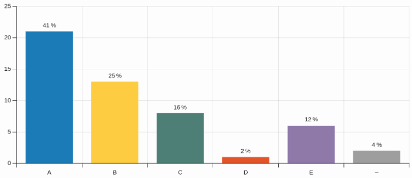
D Let's continue after 15 min break

1/3 each want lunch or continue quickly

Lecture Time - Assignments After Lecture

43 Answers

What are your constraints AFTER the lecture RPWR on Wednesdays? (currently closing at 15:45)



A I can stay for the assignments after the lecture and longer

B I 'd like to finish timely at 15:45

C Closing earlier is better, at 15:00/15:30

D I can stay just a bit until 14:30

E I can not stay for the assignments after the lecture, so I leave at 13:45 max.

2/3 can stay until 15:45, 1/6 leaves earlier, 1/6 can't stay

Results - Less Lecture, more Tutorium

Lecture: 12:15 to 13:30
Lunchbreak
Tutorium: 14:15 to 15:45

- 1/2 can start earlier
  → not enough to start earlier.
- 1/3 each want lunch or continue quickly
  → having lunch is more important that saving time.
- 2/3 can stay until 15:45, 1/6 leaves earlier, 1/6 can't stay
  → I assume the 1/3 want to go to SECORO, which should be fine now (?)

Assignments RPWR

- Assignment 2 still ongoing:
  `https://github.com/artnie/rpwr-assignments`
- Due: 06.05.25, 23:59 German time
- 12 points for this assignment

Assignments RPWR - Supplementary Material

- For reference and further information of tf2, follow the official ROS documentation:
  `https://docs.ros.org/en/jazzy/Tutorials/Intermediate/Tf2/Introduction-To-Tf2.html`
- Additional details on Markers can be found here (C++):
  `https://docs.ros.org/en/jazzy/Tutorials/Intermediate/RViz/Marker-Display-types/Marker-Display-types.html`
- The inspiration for the Balloon code, publishing Markers with the Stretch robot:
  `https://docs.hello-robot.com/0.3/ros2/example_4/`
- For more on ROS, follow the tutorials here
  `https://docs.ros.org/en/jazzy/Tutorials.html`

Schedule

- Lunchbreak until 14:15
- SECORO Tutorium here in TAB ECO 0.30 ’Knowledge’
- Next RPWR class: 07.05.25, 12:15

Evaluation

Thank you for your attention!

Special thanks to Lorenz Mösenlechner and Jan Winkler for providing illustrations!