Introduction to ROS

Lab 1 - Autonomous Robots

Laboratories Sessions

- **L1: Introduction to ROS –18th and 21th September**
- **L2: Low-level control –28th and 29th September**
- L3: Drones 1 –19th and 20th of October
- **L4: Navigation stack –2nd and 3rd of November**
- **L5: Path planning –16th and 17th of November**
- L6: Drones 2 –30th of November and 1st of December

Evaluation

- Practical Assesment (60%)
    - Developed in groups of 2 but graded individually

    - Laboratories (25%)
        - Online tests (Moodle), during next lab sessions (15%) [Lab 2-6]
        - Submit solution and code before the beginning of the next Lab session (10%) [Lab 2-6]
            - Test on real platforms [Lab 2,4,5] during the Lab session
            - Evaluation based on the code quality and organization

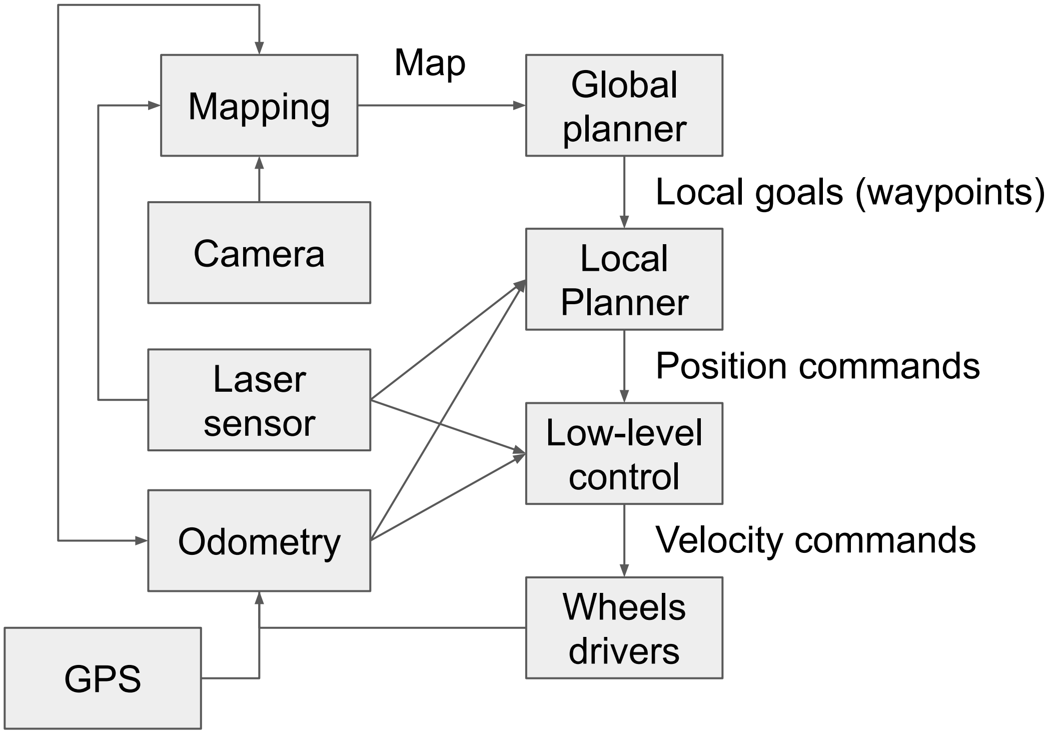
    - Practical work (35%)
        - Integration of a full navigation robotic system and/or implementation of a state of the art algorithm
        - Presentations during january: 8 mins (4+4) plus questions

~~What~~ ROS?

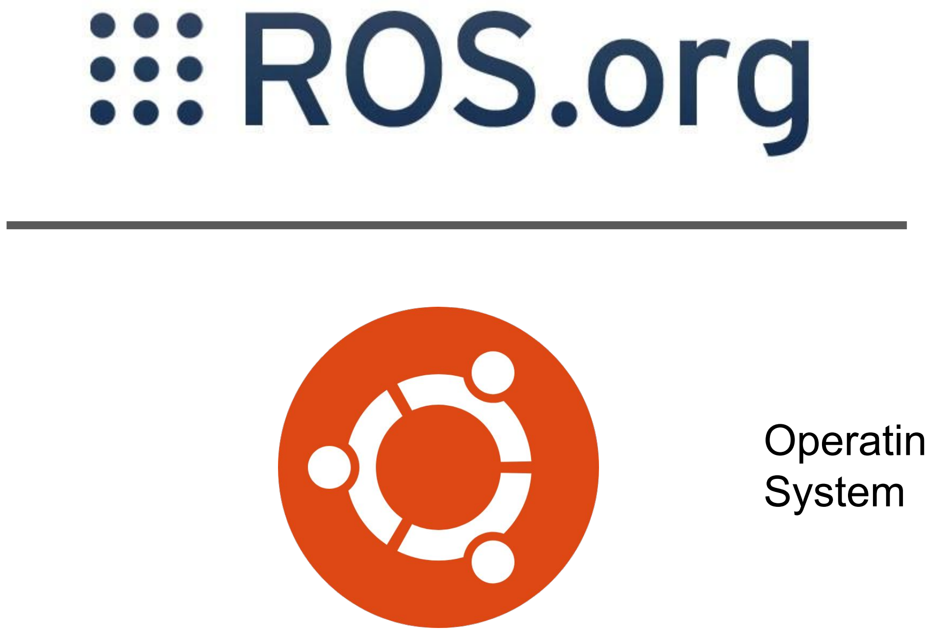
Why ROS?

ROS: Robot Operating System

- Run in **parallel**
- **Communicate** between them and between **platforms** (robots/computers)
- Useful **tools**:
  - Visualization
  - User interface
  - Simulation
- Already **created** software:
  - Planning
  - Control
  - Perception
  - Mapping
- **Easy** to install and use, standard and shareable (**community** driven).



ROS: Robot Operating System

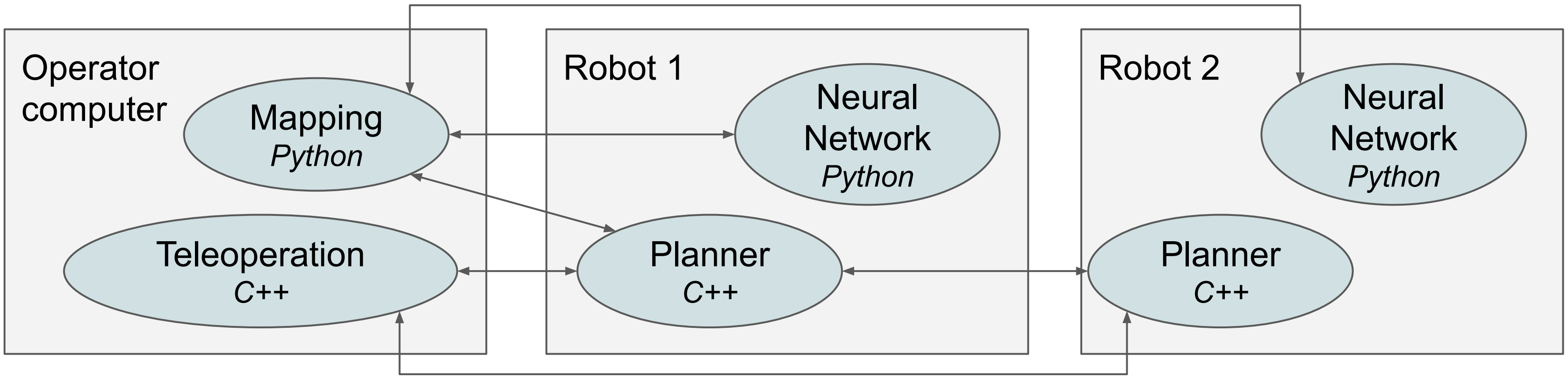


Framework (?)

Operating System

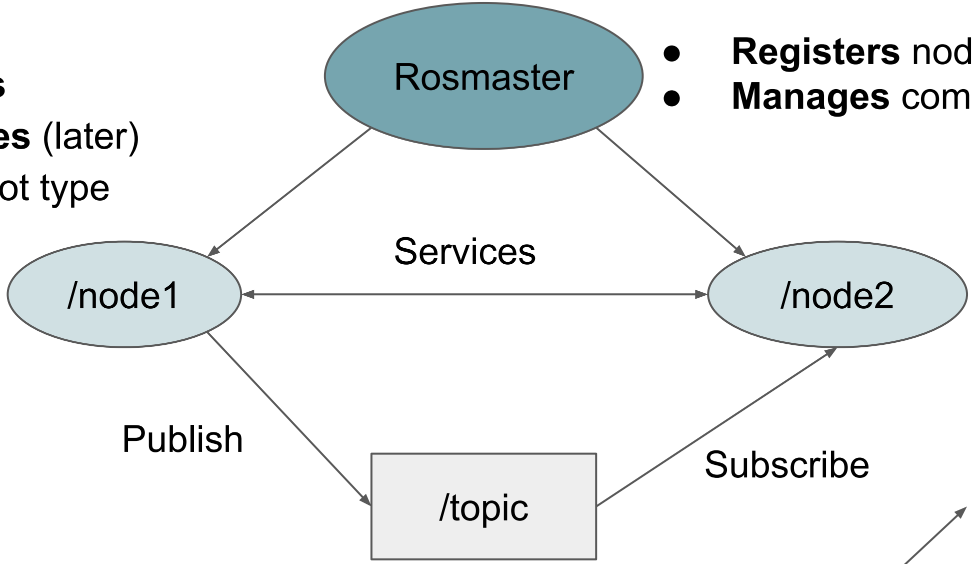
ROS features

- **Peer to peer**: individual programs running in **parallel** and **communicating** over API (ROS messages, services, etc).
- **Distributed**: multiple computers over network sharing communication
- **Multi-lingual**: **C++**, Python, MATLAB, Java, etc
- **Light-weight**: easy integration and low resources (constrained platforms: phones, drones, etc)
- **Free and open-source**: community, research and university driven projects



Nodes and topics

- Executable **programs**
- Organized in **packages** (later)
- Unique names **BUT** not type

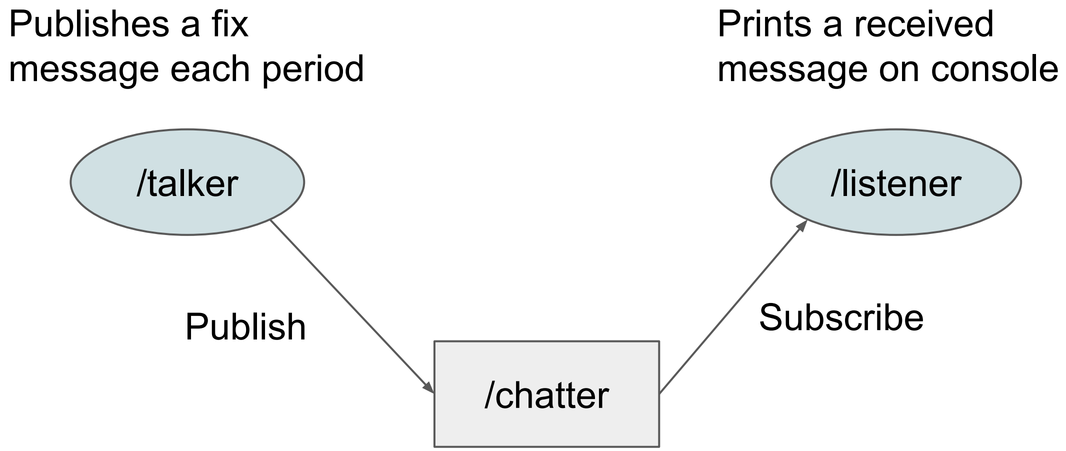


- **Registers** nodes, topics, etc
- **Manages** communication routing

/std_msgs/String
/geometry_msgs/Twist
/my_msgs/Color

- Of one **message type**
- Messages are **not retained** by default
- Unique names
- Nodes can publish **AND** subscribe to as **many topics as they want**

Simple example



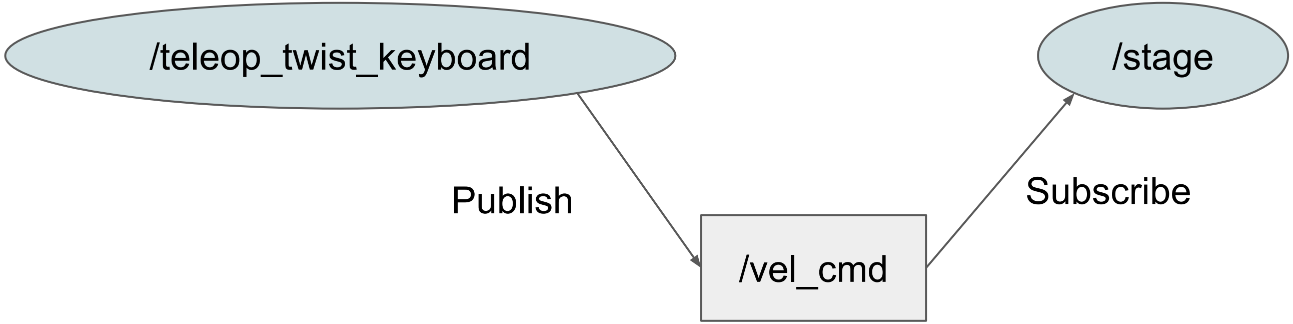
Publishes a fix message each period

Prints a received message on console

Another example

Receives keyboard inputs
and publishes velocities

Receives velocities and
moves simulation



/teleop_twist_keyboard

Publish

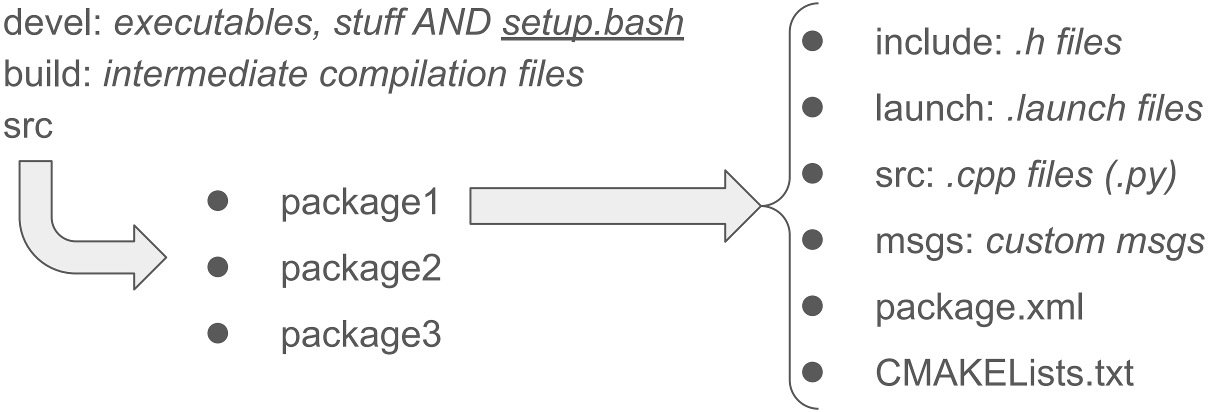
/vel_cmd

Subscribe

/stage

Workspace and build system

- Package based:
    - ROS system install (`/opt/ros/noetic/setup.bash`):
        - ROS base code
        - Basic and standalone packages (installed with *apt install*)
    - catkin_ws (my_ws):
        - Additional packages and modified standalone packages
        - Structure:
            - devel: *executables, stuff AND <u>setup.bash</u>*
            - build: *intermediate compilation files*
            - src



- package1
- package2
- package3

- include: *.h files*
- launch: *.launch files*
- src: *.cpp files (.py)*
- msgs: *custom msgs*
- package.xml
- CMAKELists.txt

Workspace and build system

- **Sourcing**: add to the terminal the information on where to **look for ROS executables** (ROS commands and ROS packages, nodes, launchfiles, etc)
  - ROS system installation and new workspaces
  - Add to .bashrc (*careful with naming conflicts*)
  - Execute command: source /path/to/catkin_ws/bin/setup.bash
- ROS uses ***improved*** CMake: manages **detection**, **compilation** and generates necessary **files** and **links**. **CATKIN**

Terminal commands

- Terminal shortcuts:
  - Tab: autocomplete
  - ctrl+c: stop node/program
  - ctrl+r: search in command history

- **Linux:**
  - **ls**: show files and directories in current directory
  - **cd**: change directory
  - **echo**: prints something on the terminal (followed by “>> filename” introduces the thing in a file)
  - **source**: source
- **Catkin:**
  - **catkin build <package_name>**: builds the package name or all packages
  - **catkin clean**: removes devel and build (cleans the workspace)
- **ROS:**
  - **roscore**: executes a rosmaster (has to remain ***on***). Rosmaster is also launched with *roslaunch*
  - **roscd <pkg_name>**: goes to the directory of the package
  - **rosrun <pkg_name> <node_name> <args>**: executes a node in the terminal and blocks the terminal
  - **roslaunch <pkg_name> <launchfile_name>**: executes a launchfile with several nodes AND the rosmaster
  - **rosnode list**: shows the list of running nodes
  - **rosnode info**: shows info (publishing/subscribed topics, name, type, package, etc)
  - **rostopic list**: shows the list of topics
  - **rostopic info**: shows info (publishing/subscribed nodes, message type, etc)
  - **rostopic echo**: prints the messages that arrived to the topic
  - **rostopic hz**: shows the publishing frequency

Care with Rosmaster conflicts!