Robotics and Intelligent Systems Lab

Lecture 1

Francesco Maurelli

Fall 2022

1. Organization
2. Introduction and History
3. Installing ROS
4. ROS Packages & Catkin
5. ROS Master, Nodes & Topics

Organization

Content

RIS Lab 1 will focus on robotics middleware, namely the Robot Operating System (ROS)

What You will learn

- Understand the key concepts of ROS
- Understand and create your own ROS programs
- Describe robotics software architecture
- Create new packages and functionalities in a robotics simulator
- Run a simulated robot using ROS Gazebo package

Topics

- Introduction & History
- Basic Concepts (File System, packages, package system)
- Catkin workspace & build system
- ROS nodes, topics, messages, services
- Creating your own nodes
- ROS bags & logging: Recording and playing back data
- Debugging strategies
- Data visualization
- Robot simulation in ROS: Gazebo

Organization

- Lectures
  - Lecture sessions
  - Hands-on work
- Schedule:
  - current: Fridays 09:45 - 11:00
  - proposal: Wednesdays 12:45 - 15:30, for half of the semester

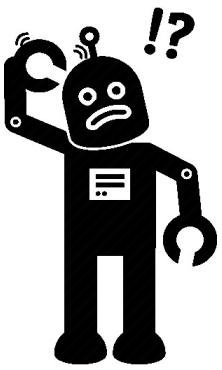
Organization

- Evaluation:
  - Project
  - Work will be split Groups of 3
  - Each team needs to submit a report (Latex preferred) with the written code
  - Each group need to specify the amount of work done my each member (in %)

Introduction and History

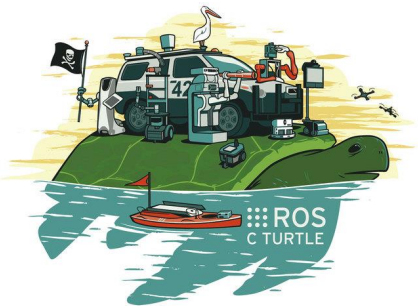
Robotic Software Development before ROS

- Detached development among research groups
- Lack of coding standards
- Limited code re-usability
- Very little documentation
- Reinventing the wheel with every new robot
  - device drivers
  - robot interfaces
  - communication protocols
  - etc.



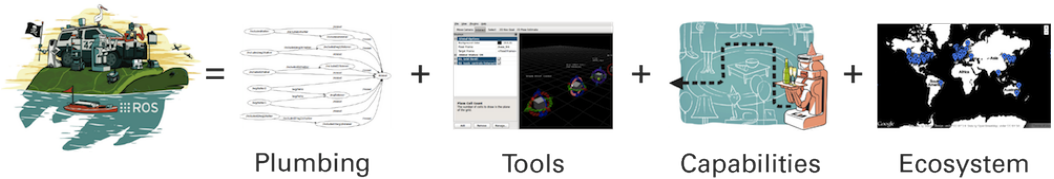
What is ROS?

- Short for **R**obot **O**perating **S**ystem (it is not)
- Can be called a meta-OS for sharing similarities with a real OS
- Created by Williow Garage and maintained by Open Source Robotic Foundation (OSRF)
- An open-source Robotics Software framework mainly for programming complex robots
- Tool for quickly prototyping a robot software, without writing everything from scratch



ros.org

The ROS Equation

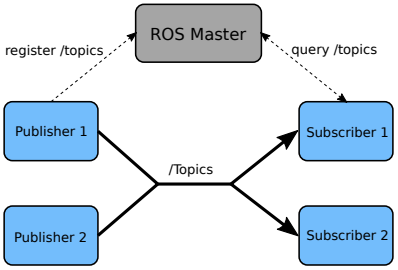


ros.org

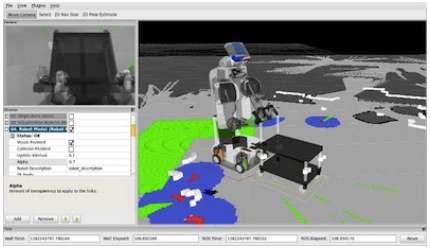
- Middleware (Plumbing)
  - Process communication and data management
- Tools
  - rviz, rqt, rosbag ...
- Capabilities (Robotics libraries)
  - Planning , control, navigation
- Ecosystem
  - Community maintained packages

Plumbing (Middleware)

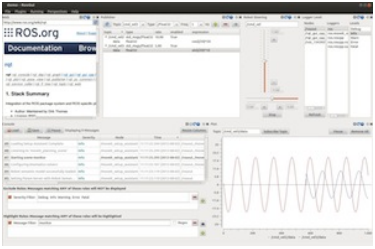
- A communication middleware which helps two or more programs to communicate with each other
- Provides APIs for users to send & receive various types of data between multiple programs
- Creates a network of programs (nodes) that are communicating with each other.
- This is called a ROS computational graph



Tools



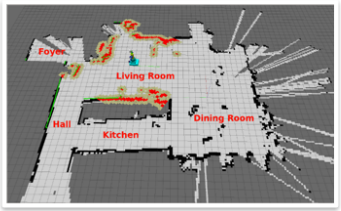
(a) Rviz



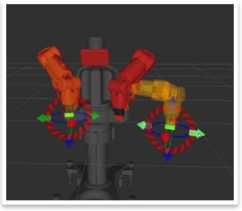
(b) Rqt

- Graphical and command-line tools
- Debug and monitor various kinds of data
- Visualize sensor data, maps, etc.
- Examples: rviz, rqt_graph, rqt_plot, rosbag, ...

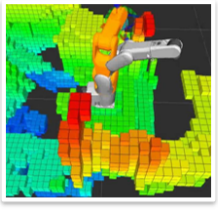
Capabilities



Navigation



Manipulation



Perception

- Built in software libraries providing various capabilities for robots such as navigation, perception, manipulation, etc.
- Examples: OpenCV, Point Cloud Library (PCL), navigation, 3d_navigation, MoveIT, Gazebo, etc.

Ecosystem

- Open-source framework powered by thousands of developers across the globe
- Documentation & support (http://wiki.ros.org/, http://answers.ros.org)

Evolution of ROS

- Stanford (<2008)
  - Avoid reinventing the wheel in robotics development
- Willow Garage (2008-2013)
  - Building libraries and a community
- OSRF/Open Robotics (2014+)
  - 2000+ packages, worldwide adoption
- ROS2 (2016)
  - New middleware with deep changes

ros.org

Distributions

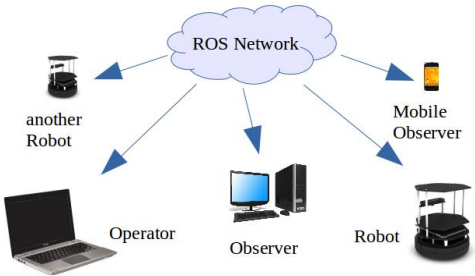
ROS Groovy Galapagos	December 31, 2012			July, 2014
ROS Fuerte Turtle	April 23, 2012			--
ROS Electric Emys	August 30, 2011			--
ROS Diamondback	March 2, 2011			--
ROS C Turtle	August 2, 2010			--
ROS Box Turtle	March 2, 2010			--

Distro	Release date	Poster	*Tuturtle*, turtle in tutorial	EOL date
ROS Noetic Ninjemys (**Recommended**)	May 23rd, 2020			May, 2025 (Focal EOL)
ROS Melodic Morenia	May 23rd, 2018			May, 2023 (Bionic EOL)
ROS Lunar Loggerhead	May 23rd, 2017			May, 2019
ROS Kinetic Kame	May 23rd, 2016			April, 2021 (Xenial EOL)
ROS Jade Turtle	May 23rd, 2015			May, 2017
ROS Indigo Igloo	July 22nd, 2014			April, 2019 (Trusty EOL)
ROS Hydro Medusa	September 4th, 2013			May, 2015

Operating Systems

ROS currently only runs on Unix-based platforms. Software for ROS is primarily tested on Ubuntu and Debian systems, though the ROS community has been contributing support for Windows, Mac OS X, Fedora, Gentoo, Arch Linux and other Linux platforms.

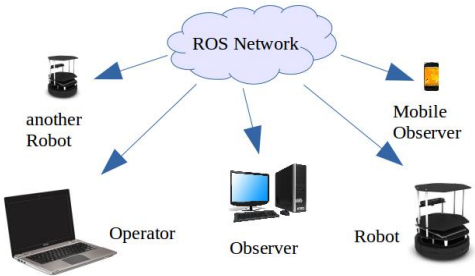
Why ROS?



wiki.ros.org

- **Peer-to-peer:** Individual programs communicate over

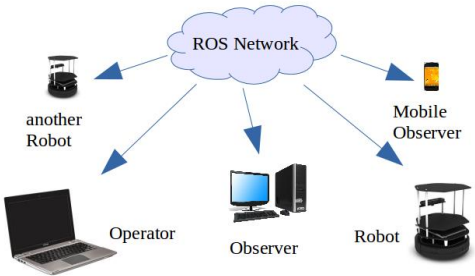
Why ROS?



wiki.ros.org

- **Peer-to-peer:** Individual programs communicate over
- **Distributed:** programs can run on multiple machines

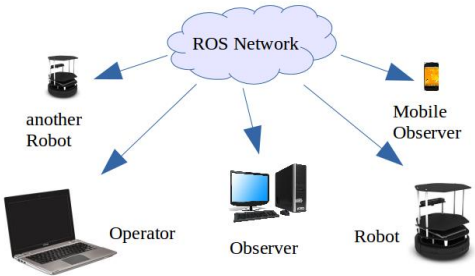
Why ROS?



wiki.ros.org

- **Peer-to-peer:** Individual programs communicate over
- **Distributed:** programs can run on multiple machines
- **Multi-lingual:** C++, Python, Matlab, Java, etc..

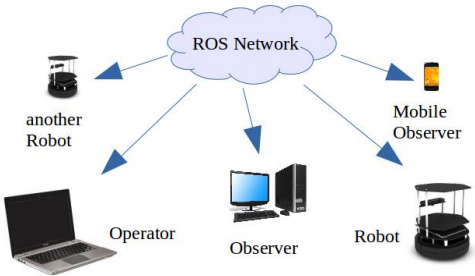
Why ROS?



wiki.ros.org

- **Peer-to-peer:** Individual programs communicate over
- **Distributed:** programs can run on multiple machines
- **Multi-lingual:** C++, Python, Matlab, Java, etc..
- **Light-weight:** Libraries are wrapped around with a ROS layer

Why ROS?



wiki.ros.org

- **Peer-to-peer:** Individual programs communicate over
- **Distributed:** programs can run on multiple machines
- **Multi-lingual:** C++, Python, Matlab, Java, etc..
- **Light-weight:** Libraries are wrapped around with a ROS layer
- **Free and open-source:** most ROS libraries are free to use

Installing ROS

Install ROS I

1. Configure your Ubuntu repositories: allow *restricted*, *universe*, and *multiverse*.
2. Setup your computer to accept software from packages.ros.org

```
$ sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" >
/etc/apt/sources.list.d/ros-latest.list'
```

3. Set up your keys

```
$ sudo apt install curl
```

```
$ curl -s https://raw.githubusercontent.com/ros/rosdistro/master/ros.asc | sudo apt-key add -
```

Install ROS II

4. Update your packages

```
$ sudo apt update
```

Install ROS III

Melodic Morenia / Ubuntu 18.04

- Desktop-Full Install: (Recommended)

```
$ sudo apt install ros-melodic-desktop-full
```

- Source the environment in every bash terminal you use ROS in

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

- Optional: automatically source environment when a shell is launched

```
$ echo "source /opt/ros/melodic/setup.bash" >>
~/.bashrc
```

Noetic Ninjemys / Ubuntu 20.04

- Desktop-Full Install: (Recommended)

```
$ sudo apt install ros-noetic-desktop-full
```

- Source the environment in every bash terminal you use ROS in

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

- Optional: automatically source environment when a shell is launched

```
$ echo "source /opt/ros/noetic/setup.bash" >>
~/.bashrc
```

Install ROS IV

Melodic Morenia / Ubuntu 18.04

- Dependencies for building packages

```
$ sudo apt install python-rosdep
python-rosinstall python-rosinstall-generator
python-wstool build-essential
```

- Initialize rosdep

```
$ sudo rosdep init
$ rosdep update
```

Noetic Ninjemys / Ubuntu 20.04

- Dependencies for building packages

```
sudo apt install python3-rosdep
python3-rosinstall
python3-rosinstall-generator python3-wstool
build-essential
```

- Initialize rosdep

```
$ sudo rosdep init
$ rosdep update
```

ROS Filesystem

```
> ls /opt/ros/noetic
bin     etc      lib                local_setup.sh   setup.bash   _setup_util.py  share
env.sh  include  local_setup.bash   local_setup.zsh  setup.sh     setup.zsh
```

- Default installation path
  - `/opt/ros/<version-codename>/ (kinetic/melodic/noetic...)`
  - Contains core and officially supported libraries
- Setup scripts
  - setup.bash – Environment setup file for Bash shell
  - setup.sh – Environment setup file for Bourne shell
  - setup.zsh – Environment setup file for zshell

Navigating ROS Filesystem

Overview of Filesystem Concepts

- **Packages:** Packages are the software organization unit of ROS code. Each package can contain libraries, executables, scripts, or other artifacts.
- **Manifests (package.xml):** A manifest is a description of a package. It serves to define dependencies between packages and to capture meta information about the package like version, maintainer, license, etc...

Navigating ROS Filesystem

Overview of Filesystem Concepts

- **Packages:** Packages are the software organization unit of ROS code. Each package can contain libraries, executables, scripts, or other artifacts.
- **Manifests (package.xml):** A manifest is a description of a package. It serves to define dependencies between packages and to capture meta information about the package like version, maintainer, license, etc...

- *rospack* allows you to get information about packages.

```
$ rospack find [package_name]
```

```
▶ rospack find roscpp
/opt/ros/noetic/share/roscpp

~
▶ rospack find rospy
/opt/ros/noetic/share/rospy
```

Navigating ROS Filesystem

- *roscd* allows to change directory (cd) directly to a package or a stack.

```
$ roscd <package-or-stack>[/subdir]
```

```
▶ roscd roscpp

noetic/share/roscpp
▶ pwd
/opt/ros/noetic/share/roscpp
```

```
▶ roscd roscpp/
roscpp/cmake      roscpp/msg       roscpp/rosbuild  roscpp/srv
```

Navigating ROS Filesystem

- *roscd* allows to change directory (cd) directly to a package or a stack.

```
$ roscd <package-or-stack>[/subdir]
```

```
▶ roscd roscpp

noetic/share/roscpp
▶ pwd
/opt/ros/noetic/share/roscpp
```

```
▶ roscd roscpp/
roscpp/cmake      roscpp/msg       roscpp/rosbuild  roscpp/srv
```

Note: roscd will only find ROS packages that are within the directories listed in your `ROS_PACKAGE_PATH`. To see what is in your `ROS_PACKAGE_PATH`, type:

```
▶ echo $ROS_PACKAGE_PATH
/opt/ros/noetic/share
```

Navigating ROS Filesystem

- *rosls* allows to ls directly in a package by name rather than by absolute path.

```
$ rosls <package-or-stack>[/subdir]
```

```
▶ rosls roscpp
cmake  msg  package.xml  rosbuild  srv
```

Navigating ROS Filesystem

- *rosls* allows to ls directly in a package by name rather than by absolute path.

```
$ rosls <package-or-stack>[/subdir]
```

```
▶ rosls roscpp
cmake  msg  package.xml  rosbuild  srv
```

Summary

- rospack = ros + pack(age)
- roscd = ros + cd
- rosls = ros + ls

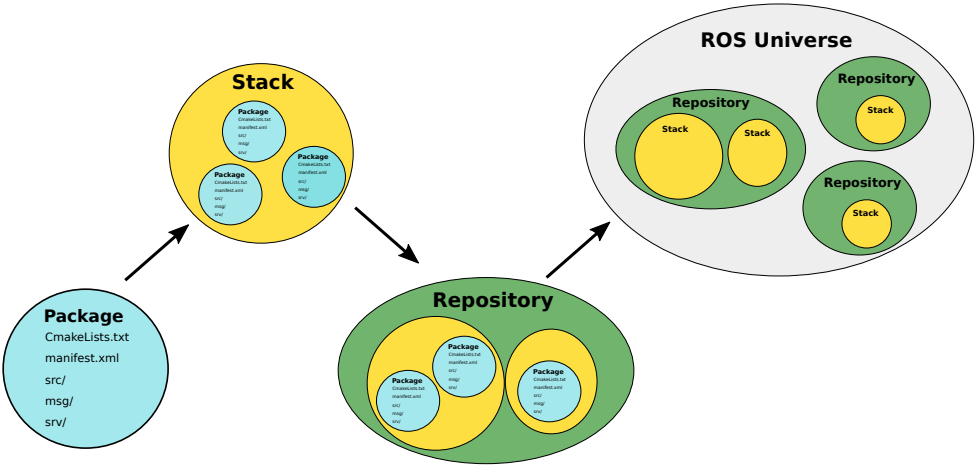
ROS Packages & Catkin

ROS Package

- Software in ROS is organized in packages
- A package can contain one or more nodes, documentation, and provides a ROS interface
- For a package to be considered a catkin package it must meet a few requirements:
  - The package must contain a catkin compliant package.xml file (meta information about the package).
  - The package must contain a CMakeLists.txt which uses catkin.
  - Each package must have its own folder

```
my_package/
  CMakeLists.txt
  package.xml
```

ROS Package System



Catkin Workspaces

- A ROS package is a directory inside a **catkin workspace** that has a package.xml file in it
- A **catkin workspace** is a set of directories in which a set of related ROS code/packages live (catkin ~ ROS build system: CMake + Python scripts)
- Keep user work or developed packages independent from the ROS default packages
- It's possible to have multiple workspaces, but work can performed on only one-at-a-time

```
workspace_folder/        -- WORKSPACE
  src/                   -- SOURCE SPACE
    CMakeLists.txt       -- 'Toplevel' CMake file, provided by catkin
    package_1/
      CMakeLists.txt     -- CMakeLists.txt file for package_1
      package.xml        -- Package manifest for package_1
    ...
    package_n/
      CMakeLists.txt     -- CMakeLists.txt file for package_n
      package.xml        -- Package manifest for package_n
```

Catkin Workspace Structure

Don't touch

build

The build space
Cmake is invoked here to build packages in *src/*. Keeps cache information and intermediate files.

Don't touch

devel

The development space
Keeps built packages/targets prior to being installed

Work here

src

The source space
Keeps all source code of packages. This directory is used to create source code of packages you want to build

Creating a Catkin Workspace

- Pre-requisite: install catkin_tools

```
$ sudo apt-get install python3-catkin-tools
```

Creating a Catkin Workspace

- Pre-requisite: install catkin_tools

```
$ sudo apt-get install python3-catkin-tools
```

- Open a new shell and source the ROS environemt

```
$ source /opt/ros/<version-codename>/setup.bash
```

Creating a Catkin Workspace

- Pre-requisite: install catkin_tools

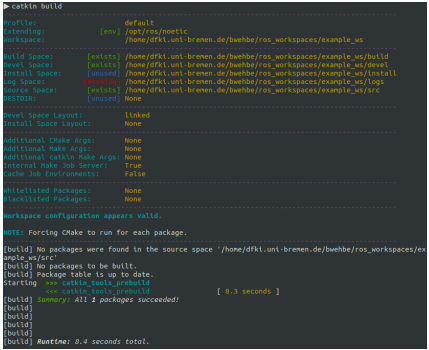
```
$ sudo apt-get install python3-catkin-tools
```

- Open a new shell and source the ROS environemt

```
$ source /opt/ros/<version-codename>/setup.bash
```

- Create and build a catkin workspace

```
$ mkdir -p ~/catkin_ws/src
$ cd ~/catkin_ws/
$ catkin build
$ source devel/setup.bash
```



- use *catkin build* instead of *catkin_make*

Creating a ROS Package

- Change to the source space directory of the catkin workspace you created

```
$ cd ~/catkin_ws/src
```

Creating a ROS Package

- Change to the source space directory of the catkin workspace you created

```
$ cd ~/catkin_ws/src
```

- The tool *catkin_create_pkg* can be used given the name of the package and the desired dependencies

```
$ catkin_create_pkg <package_name> [depend1] [depend2] [depend3] #do not run
```

```
$ catkin_create_pkg beginner_tutorials std_msgs rospy roscpp
```

Creating a ROS Package

- Change to the source space directory of the catkin workspace you created

```
$ cd ~/catkin_ws/src
```

- The tool *catkin_create_pkg* can be used given the name of the package and the desired dependencies

```
$ catkin_create_pkg <package_name> [depend1] [depend2] [depend3] #do not run
```

```
$ catkin_create_pkg beginner_tutorials std_msgs rospy roscpp
```

```
▶ ls beginner_tutorials
CMakeLists.txt  include  package.xml  src
```

Building a ROS Package

- Go back to the catkin workspace and build the new package

```
$ cd ~/catkin_ws
$ catkin build
```

- After the workspace has been built it has created a similar structure in the devel subfolder. To add the workspace to your ROS environment you need to source the generated setup file:

```
$ source devel/setup.bash
```

Building a ROS Package

- Go back to the catkin workspace and build the new package

```
$ cd ~/catkin_ws
$ catkin build
```

- After the workspace has been built it has created a similar structure in the devel subfolder. To add the workspace to your ROS environment you need to source the generated setup file:

```
$ source devel/setup.bash
```

- To view the first-order dependencies of a package, *rospack depends1* can be used

```
$ rospack depends1 beginner_tutorials
roscpp
rospy
std_msgs
```

Building a ROS Package

- You can also find the dependencies in the package.xml file

```
$ roscd beginner_tutorials
$ cat package.xml
```

```
<package format="2">
...
  <buildtool_depend>catkin</buildtool_depend>
  <build_depend>roscpp</build_depend>
  <build_depend>rospy</build_depend>
  <build_depend>std_msgs</build_depend>
...
</package>
```

Building a ROS Package

- You can also find the dependencies in the package.xml file

```
$ roscd beginner_tutorials
$ cat package.xml
```

```
<package format="2">
...
  <buildtool_depend>catkin</buildtool_depend>
  <build_depend>roscpp</build_depend>
  <build_depend>rospy</build_depend>
  <build_depend>std_msgs</build_depend>
...
</package>
```

- Finding indirect dependencies

```
$ rospack depends1 rospy
$ rospack depends beginner_tutorials
```

Customizing your Package

- Let's open the package.xml file again
  - Description tag

```
<description>The beginner_tutorials package</description>
```

  - Maintainer tag

```
<!-- <maintainer email="jane.doe@example.com">Jane Doe</maintainer> -->
<maintainer email="user@todo.todo">user</maintainer>
```

  - Package URL

```
<url type="website">http://wiki.ros.org/beginner_tutorials</url>
```

  - License tag (more on licenses: https://opensource.org/licenses/alphabetical)

```
<!-- BSD, MIT, Boost Software License, GPLv2, GPLv3, LGPLv2.1, LGPLv3 -->
<license>TODO</license>
```

Customizing your Package

- Dependencies tag:
  - `buildtool_depend`: for build tool packages

```
<buildtool_depend>catkin</buildtool_depend>
```

  - `build_export_depend`: if you want to build packages on top of this package

```
<buildtool_depend>package-name</buildtool_depend>
```

  - `build_depend`: for packages you need at compile time

```
<build_depend>package-name</build_depend>
```

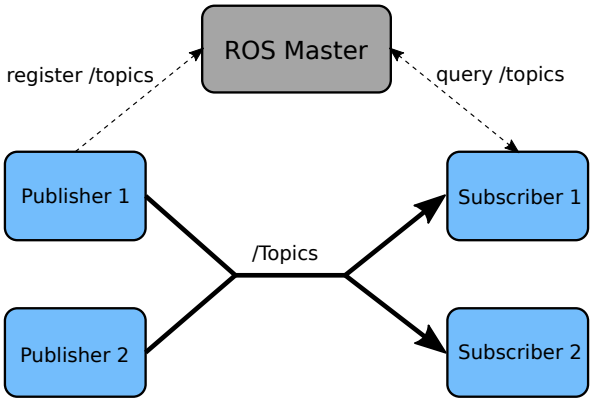
  - `exec_depend`: for packages you need at run time

```
<exec_depend>package-name</exec_depend>
```

ROS Master, Nodes & Topics

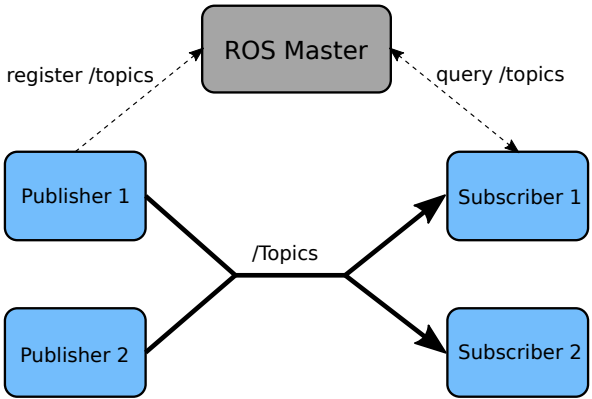
Plumbing (Middleware)

- A communication middleware which helps two or more programs to communicate with eauch other
- Provides APIs for users to send & receive various types of data between multiple programs
- Creates a network of programs (nodes) that are communicating with each other.
- This is called a ROS computational graph



ROS Master I

- Allows all ROS software (nodes) to find and talk to each other
- Every node registers the details of messages, services or actions
- **Publishing:** sending data
- **Subscribing:** receiving data



ROS Master II

- Starting ROS Master

```
$ roscore
```

```
▶ roscore
... logging to /home/dfki.uni-bremen.de/bwehbe/.ros/log/764bc10a-0b06-11ec-8bfc-e1d513279c14/roslaunc
h-EUREX-LAP2-U-302840.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://EUREX-LAP2-U:34511/
ros_comm version 1.15.11


SUMMARY
========

PARAMETERS
 * /rosdistro: noetic
 * /rosversion: 1.15.11

NODES

auto-starting new master
process[master]: started with pid [302855]
ROS_MASTER_URI=http://EUREX-LAP2-U:11311/

setting /run_id to 764bc10a-0b06-11ec-8bfc-e1d513279c14
process[rosout-1]: started with pid [302872]
started core service [/rosout]
```

ROS Master II

- Starting ROS Master

```
$ roscore
```

- Notice the PARAMETERS field, roscore enables sharing values between nodes.
- Typically used for data that does not change over the course of a run

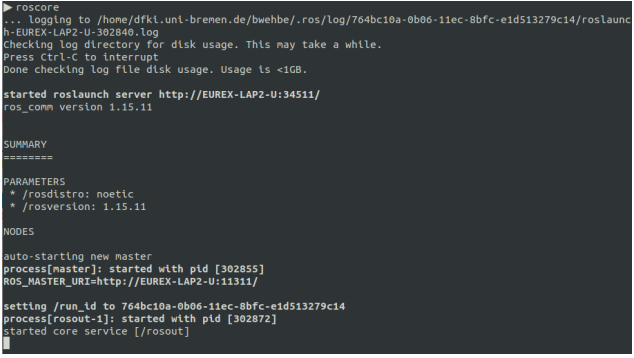
```
▶ roscore
... logging to /home/dfki.uni-bremen.de/bwehbe/.ros/log/764bc10a-0b06-11ec-8bfc-e1d513279c14/roslaunc
h-EUREX-LAP2-U-302840.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://EUREX-LAP2-U:34511/
ros_comm version 1.15.11


SUMMARY
========

PARAMETERS
 * /rosdistro: noetic
 * /rosversion: 1.15.11

NODES

auto-starting new master
process[master]: started with pid [302855]
ROS_MASTER_URI=http://EUREX-LAP2-U:11311/

setting /run_id to 764bc10a-0b06-11ec-8bfc-e1d513279c14
process[rosout-1]: started with pid [302872]
started core service [/rosout]
```

ROS Master II

- Starting ROS Master

```
$ roscore
```

- Notice the PARAMETERS field, roscore enables sharing values between nodes.
- Typically used for data that does not change over the course of a run
- */rosout*: is a node used for logging



ROS Parameters

- Use rosparam to interact with the parameter server from the command-line.

```
▶ rosparam list
/rosdistro
/roslaunch/uris/host_eurex_lap2_u__34511
/rosversion
/run_id

▶ rosparam get /rosdistro
'noetic

  '
```

ROS Parameters

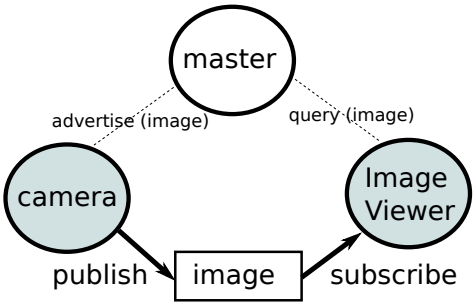
- Use rosparam to interact with the parameter server from the command-line.

```
▶ rosparam list
/rosdistro
/roslaunch/uris/host_eurex_lap2_u__34511
/rosversion
/run_id

▶ rosparam get /rosdistro
'noetic

  '
```

```
▶ rosparam --help
rosparam is a command-line tool for getting, setting, and deleting parameters from the ROS Parameter
Server.

Commands:
        rosparam set    set parameter
        rosparam get    get parameter
        rosparam load   load parameters from file
        rosparam dump   dump parameters to file
        rosparam delete delete parameter
        rosparam list   list parameter names
```

ROS Node

- Single-purposed executable programs
  - for example, camera or motor driver, controller, mapper, etc.
- Each node is individually compiled, executed, and managed
- Nodes are managed in packages
- Nodes use a ROS client library to connect with other nodes
  - **rospy** is the client library for python
  - **roscpp** is the client library for C++
- Nodes can *publish* or *subscribe* to a **Topic**
- Nodes can also *provide* or *use* a **Service** or an **Action**



Inspecting ROS Nodes

- `rosnode` displays information about the ROS nodes that are currently running

```
$ rosnode list
/rosout
```

- only one node is running */rosout*

Inspecting ROS Nodes

- `rosnode` displays information about the ROS nodes that are currently running

```
$ rosnode list
/rosout
```

- only one node is running */rosout*
- `rosnode info` command returns information about a specific node.

```
▶ rosnode info /rosout
--------------------------------------------------------------------------------
Node [/rosout]
Publications:
 * /rosout_agg [rosgraph_msgs/Log]

Subscriptions:
 * /rosout [unknown type]

Services:
 * /rosout/get_loggers
 * /rosout/set_logger_level


contacting node http://EUREX-LAP2-U:43641/ ...
Pid: 302872
```

Inspecting ROS Nodes

- Other `rosnode` commands are

```
$ rosnode [ping, kill] <node_name>
$ rosnode [machine, cleanup]
```

Inspecting ROS Nodes

- Other `rosnode` commands are

```
$ rosnode [ping, kill] <node_name>
$ rosnode [machine, cleanup]
```

- `ping`: test connectivity to node
- `kill`: kill a running node
- `machine`: list nodes running on a particular machine or list machines
- `cleanup`: purge registration information of unreachable nodes

Running ROS Nodes I

- `rosrun` allows you to use the package name to directly run a node within a package

```
$ rosrun [package_name] [node_name]
```

```
▶ rosrun turtlesim turtlesim_node
[ INFO] [1630492547.098660870]: Starting turtlesim with node name /turtlesim
[ INFO] [1630492547.108846571]: Spawning turtle [turtle1] at x=[5,544445], y=[5,544445], theta=[0,000
000]
```

Running ROS Nodes I

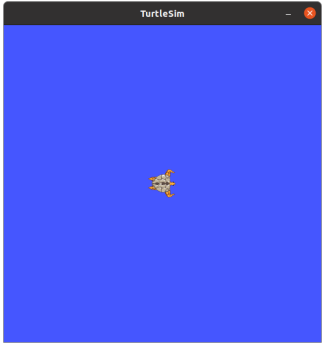
- `rosrun` allows you to use the package name to directly run a node within a package

```
$ rosrun [package_name] [node_name]
```

```
▶ rosrun turtlesim turtlesim_node
[ INFO] [1630492547.098660870]: Starting turtlesim with node name /turtlesim
[ INFO] [1630492547.108846571]: Spawning turtle [turtle1] at x=[5,544445], y=[5,544445], theta=[0,000
000]
```

- List the running nodes again

```
$ rosnode list
/rosout
/turtlesim
```



Running ROS Nodes II

- Names of nodes can be reassigned from command-line

```
$ rosrun turtlesim turtlesim_node __name:=my_turtle
```

```
$ rosnode list
/my_turtle
/rosout
```

Running ROS Nodes II

- Names of nodes can be reassigned from command-line

```
$ rosrun turtlesim turtlesim_node __name:=my_turtle
```

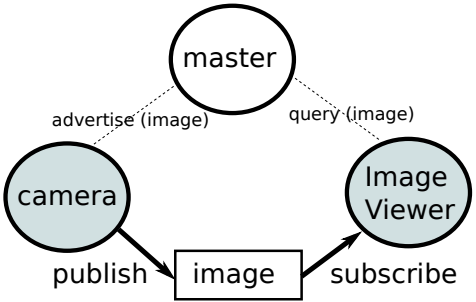
```
$ rosnode list
/my_turtle
/rosout
```

- A node could be pinged to check if a node is running

```
▶ rosnode ping /my_turtle
rosnode: node is [/my_turtle]
pinging /my_turtle with a timeout of 3.0s
xmlrpc reply from http://EUREX-LAP2-U:38565/     time=85.242271ms
xmlrpc reply from http://EUREX-LAP2-U:38565/     time=0.648022ms
xmlrpc reply from http://EUREX-LAP2-U:38565/     time=0.813246ms
xmlrpc reply from http://EUREX-LAP2-U:38565/     time=0.793219ms
```

ROS Topics I

- Topics are channels for communication between nodes
- A node sends a message over a topic by publishing it
- Other nodes may subscribe to the topic to receive the messages published to it
- A topic is associated with a single message type
- Sensor data is often published to specific topics (e.g. image)



ROS Topics II

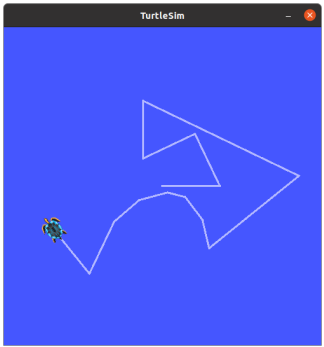
- Make sure turtlesim is running

```
$ rosrun turtlesim turtlesim_node
```

- Run `turtle_teleop_key` to control the turtle

```
$ rosrun turtlesim turtle_teleop_key
```

- Move the turtle with the arrow keys



ROS Topics III

- To check the active topics (-v for verbose)

```
▶ rostopic list -v

Published topics:
 * /rosout_agg [rosgraph_msgs/Log] 1 publisher
 * /rosout [rosgraph_msgs/Log] 2 publishers
 * /turtle1/pose [turtlesim/Pose] 1 publisher
 * /turtle1/color_sensor [turtlesim/Color] 1 publisher
 * /turtle1/cmd_vel [geometry_msgs/Twist] 1 publisher

Subscribed topics:
 * /rosout [rosgraph_msgs/Log] 1 subscriber
 * /turtle1/cmd_vel [geometry_msgs/Twist] 1 subscriber
```

- Show information about a topic

```
▶ rostopic info /turtle1/cmd_vel
Type: geometry_msgs/Twist

Publishers:
 * /teleop_turtle (http://EUREX-LAP2-U:43555/)

Subscribers:
 * /turtlesim (http://EUREX-LAP2-U:43707/)
```

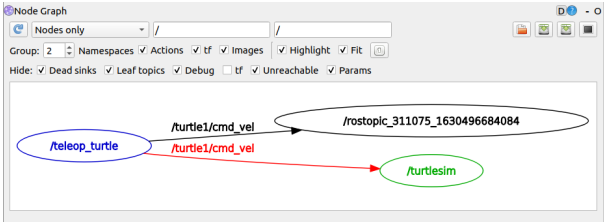
ROS Topics IV

- Subscribe and print content of a topic (then move the turtle)

```
rostopic echo /turtle1/cmd_vel
linear:
  x: 2.0
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 0.0
---
```

- In a new terminal

```
$ rosrun rqt_graph rqt_graph
```



Questions?