École Nationale Supérieure
de **Techniques Avancées**

Architecture for robotics ROB314

Emmanuel Battesti - 19/01/2024

Course objectives

- Learn Robotic Middleware: ROS
- Discover the mechatronic aspects of robotics
- Developing a "complex" robotics project
  - A personalized project on real robot
  - Integration of functions: perception/navigation/control
  - Development of new functions
- Gain experience in computer science
  - Languages Python, C++
  - OS Linux, Git
  - Development
- Use things learned in other robotics courses
  - Navigation, Vision, etc.

Course Program

- A *ROS presentation* course (3 sessions)

  → https://perso.ensta-paris.fr/~battesti/rob314.htm

- A *mechatronic in robotic* course (3 sessions)
- A project:
  - Create teams of 1 to 3 people
  - Definition of a project and choice of a robot
  - Robot handling and project development (6 sessions)
  - Last session: presentation of projects
  - A report will be due one week later

Contact information

- Lecturers
  - **Emmanuel Battesti**, robotics engineer at U2IS ENSTA (emmanuel.battesti@ensta-paris.fr)
  - **Thibault Toralba**, robotics engineer at U2IS ENSTA (thibault.toralba@ensta-paris.fr)
- Leader
  - **David Filliat,** professor at U2IS ENSTA

    (david.filliat@ensta-paris.fr)

Robots available: Turtlebot

- Turtlebot 2.0
  - Differential mobile base, 6kg, 0.6 m/s
  - Netbook ROS (navigation, mapping…)
  - RGBD sensors (Kinect or Xtion),
  - Encoders,
  - Gyrometers, bumpers
  - laser telemeter, that can be added

Robots available: Drone DJI Tello

- 1 front camera
- 1 camera under the drone for stabilization
- Several preprogrammed modes
- 13 min time of flight

Others robots

- Robots built in U2IS
- 1 or 2 Husky
- Older robots:
  - 1 Nao v4
  - 1 Pepper robot
  - Old AR-Drones 1.0

École Nationale Supérieure
de **Techniques Avancées**

Introduction to ROS ROB314

Emmanuel Battesti

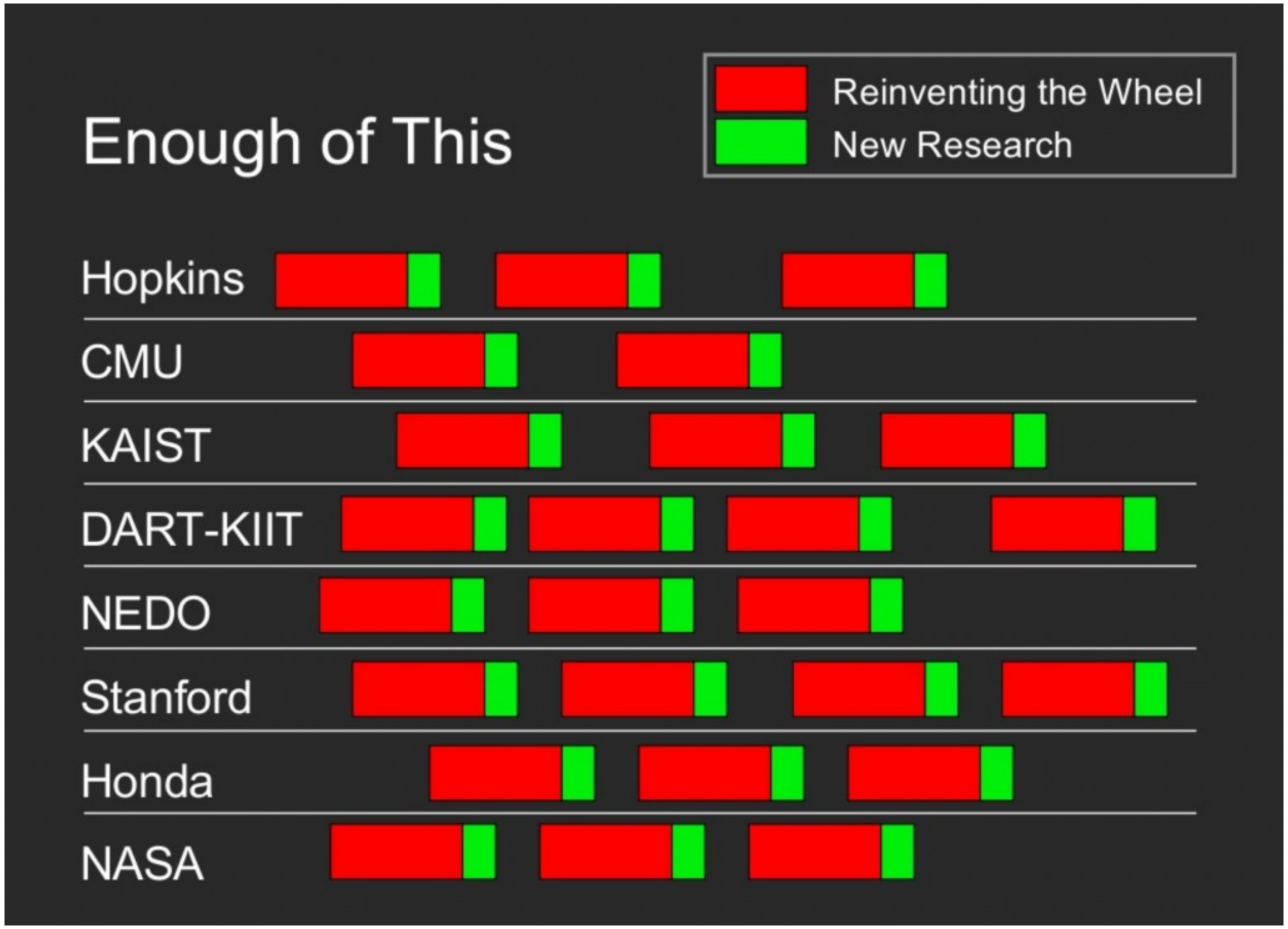
Course Summary

- Why does ROS exist?
- How does ROS work?
- How to use ROS with your own code ?

What tools are needed in robotics?

- **Distributed computation:**
  - robots should be able to work with **remote** software or hardware,
  - robots should be able to work with **humans** via software interface.
  - Small stand-alone parts of software should be able to **cooperate**.
  - ➔ need communication mechanisms
- **Software reuse:** growing collections of algorithms
  - Need **standard packages**
  - Need **standard communication** and **standard interface**
  - **Community**: a place where we can discuss and share some codes
- **Rapid testing:**
  - Use **simulators** instead of real robots ➔ easier
  - But also **record** and play back of real data sensor

Problems in robotics before ROS



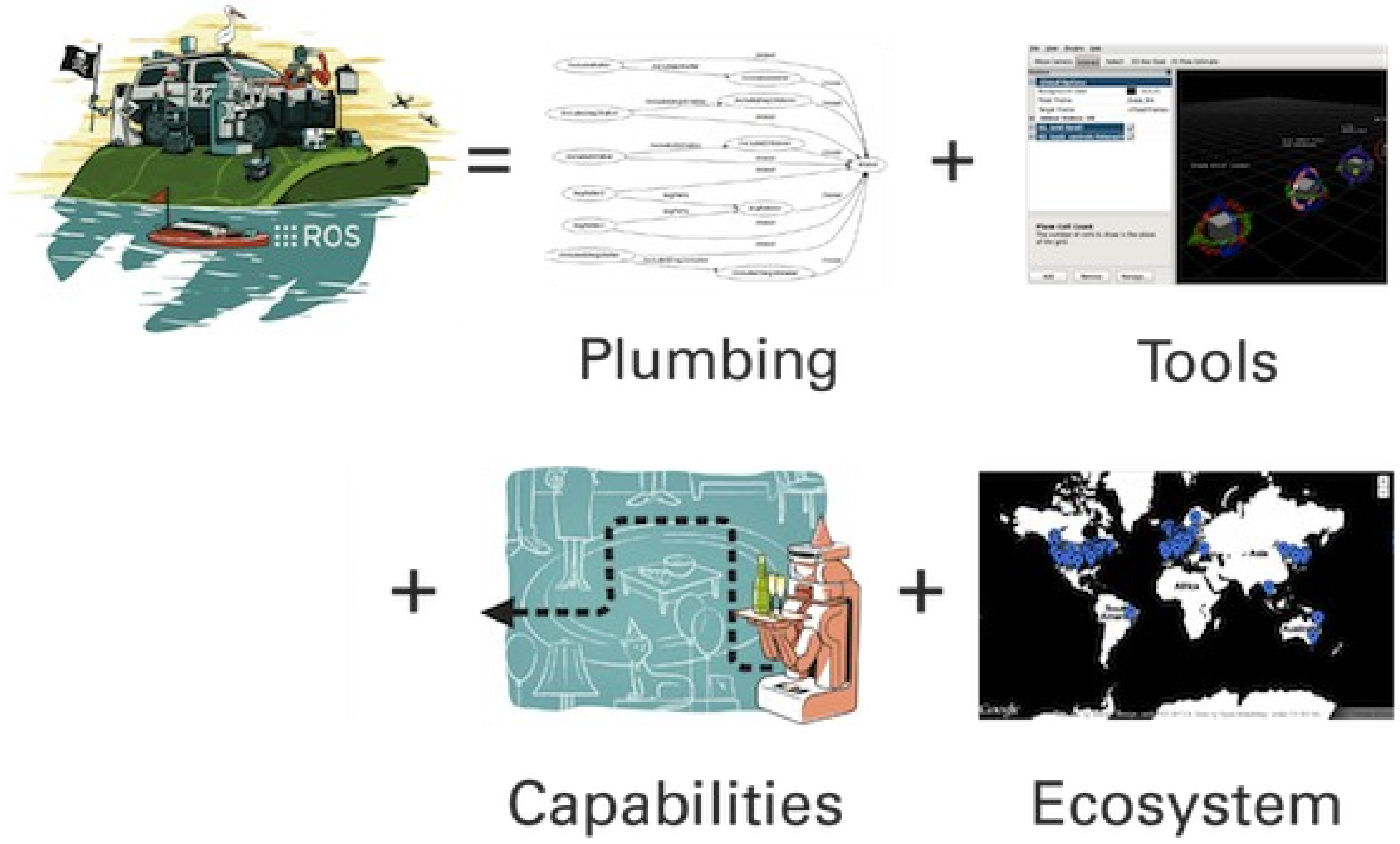
What is ROS?

- **ROS** means **R**obot **O**perating **S**ystem
- ROS is **open-source** software
- Software tools that help you **build 'easily' robot applications**.
- And that work across a **wide variety of robotic platforms**.

What does ROS bring?

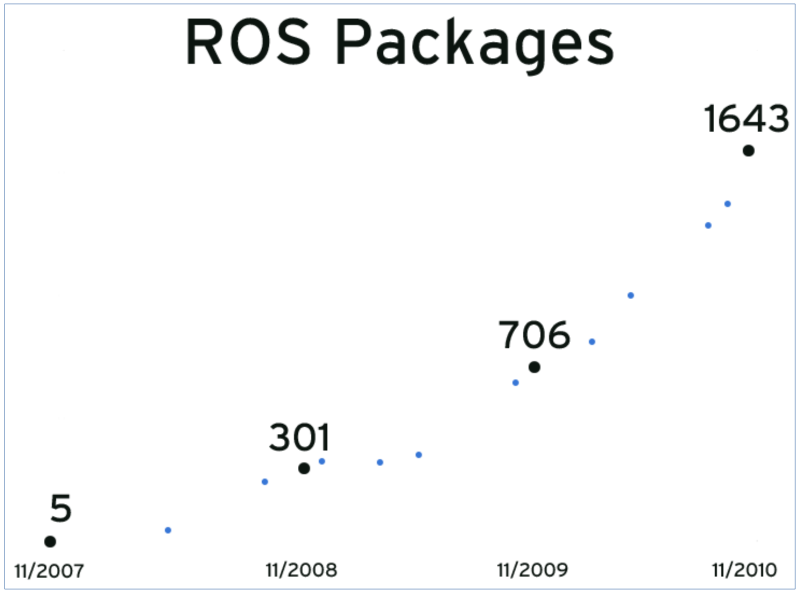
- Great tools:
  - **Communication tools ➔** standard messages and communication library (topics, services, parameters)
  - **Distributed computation ➔** a central server called *master*
  - An **OS-like structure** to organize (packages, nodes) and **command tools** to compile and navigate easily (catkin, roscd, rosls,..).
  - **Testing ➔** simulators (*gazebo*), visualizations (*rviz*), data logging, replaying (ros*bag*)
- Lot of help and useable algorithms
  - **Ecosystem ➔** a large community (wiki) and a lot of standalone libraries are wrapped for ROS (ex: *OpenCV*)
  - **Capabilities ➔** a lot of packages are available: control, planning, perception, mapping, manipulation, etc.

ROS Overview



History of ROS

- Started at *Stanford University*, ~2005
  - **personal** project of Keenan Wyrobek and Eric Berger, two phd students
  - They observe that roboticists **waste time** in areas that do not interest them and that they do not master.
  - First prototype with the robot PR1
- Carried by *Willow Garage,* 2008 – 2013
  - Willow Garage was a robotic research center
  - 2010: **first distribution**
  - Ros became popular
  - 2011: release of turtlebot robot
- Now by *Open Robotics Foundation,* since 2013
- Creation of *ROS 2.0* in 2015, and first release in 2017
  - the distribution is completely rethought
  - Oriented towards industry: real-time, security, etc.
- Robots
  - Hundreds of robots: https://robots.ros.org/
  - For research, this has become a standard.



ROS philosophy

- **Peer to Peer**
  - ROS systems consist of numerous small computer programs which connect to each other and continuously exchange messages
- **Tools-based**
  - There are many small, generic programs that perform tasks such as visualization, logging, plotting data streams, etc.
- **Multi-Lingual**
  - ROS software modules can be written in any language for which a client library has been written. Currently client libraries exist for **C++**, **Python**, LISP, Java, JavaScript, MATLAB, Ruby, and more.
- **Thin**
  - The ROS conventions encourage contributors to create stand-alone libraries and then wrap those libraries so they send and receive messages to/from other ROS modules.
- **Free and open source**

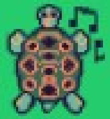
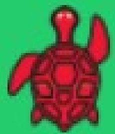
ROS is not...

- **ROS is not a programming language:** could use C++, Python, Java, Lisp
- **ROS is not only a library** (see above)
- **ROS is not an integrated development environment:** could be used with most popular IDEs.

ROS Requirement

- Mainly on Ubuntu
- 1 ROS version ⇔ 1 Ubuntu version
  - ‘Long Term Support’ version ROS Melodic Morenia + Ubuntu 18.04
  - ‘Long Term Support’ version ROS Noetic Ninjemys + Ubuntu 20.04
- The different versions of ROS are not always compatible between them.
- Quite big but easy to install
- Avoid virtual machines to work with real robots
- Multi-lingual
  - ROS modules can be written in any language for which a client library exists (C++, Python, MATLAB, Java, etc.).

ROS Distribution Releases

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 Melodic Installation

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

```
sudo apt-key adv --keyserver 'hkp://keyserver.ubuntu.com:80' --recv-key
C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654
```

```
sudo apt-get --yes update
Sudo apt-get --yes install ros-melodic-desktop-full python-rosinstall
python-rosinstall-generator python-wstool build-essential python-rosdep
rosdep init
echo "source /opt/ros/melodic/setup.bash" >> ~/.bashrc
```

Course Summary

- Why does ROS exist?
- <u>How does ROS work?</u>
- How to use ROS with your own code ?

ROS Packages 1/2

- **All ROS software** is organized into *package*
- A *package* is one tool or a set of tools on a **particular theme**
- A package usually contains one or more **nodes (i.e. ROS executables)**.
- A package can very well contain only a **library**.
- Sometimes, **known libraries** are packaged for ROS (like Open-CV or PCL).
- **Package**:
  - source code and/or executables (nodes),
  - scripts,
  - config files,
  - dataset,
  - messages or/and services…

ROS Packages 2/2

- Where can we find the packages?
  - Most ROS packages are hosted in **GitHub**.
  - Can be part of a **metapackage**: a collection of related packages (for example *ros_base* ou *ros_control*)
  - We can create our **own package**.
  - The main packages can be installed as **Ubuntu packages** (*sudo apt install ros-melodic-xxx*)

- Listing and locating packages: *rospack list*
- Locating a single package: *rospack find package-name*
- Linux-like command: *roscd, rosls*…

ROS Nodes 1/2

- **Node = single-purposed executable in ROS applications**: e.g. sensor driver(s), actuator driver(s), mapper, planner, UI, image viewer, logger, etc.
- **Individually** compiled, executed, and managed:
  - **One process** by node. So, if one of them craches, the others nodes will not.
  - Reduce code complexity
  - Easier to test
- Nodes are combined into a **graph** and **communicate with each other** using ROS topics, services, actions, etc.
- Organized in ***packages***
- Nodes are **language agnostic**: for example, python node can communicate with c++ node.

ROS Nodes 2/2

- Nodes of the **same type** can be started multiple times but with **different names**.

- Run a node with:

```
> rosrun package_name node_type
```

See active nodes with:

```
> rosnode list
```

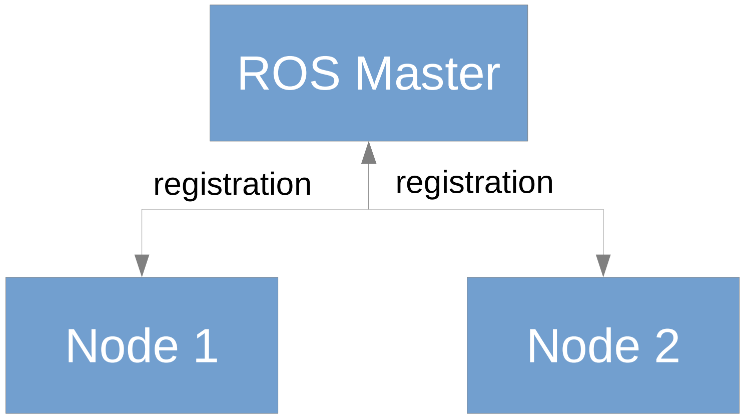
Retrieve information about a node with:

```
> rosnode info node_type
```

Warning!
node_type ≠ node_name

ROS Master

- Every node **registers** at startup with the master
- Manages the **communication** between nodes (processes)
- Host a **parameter server**

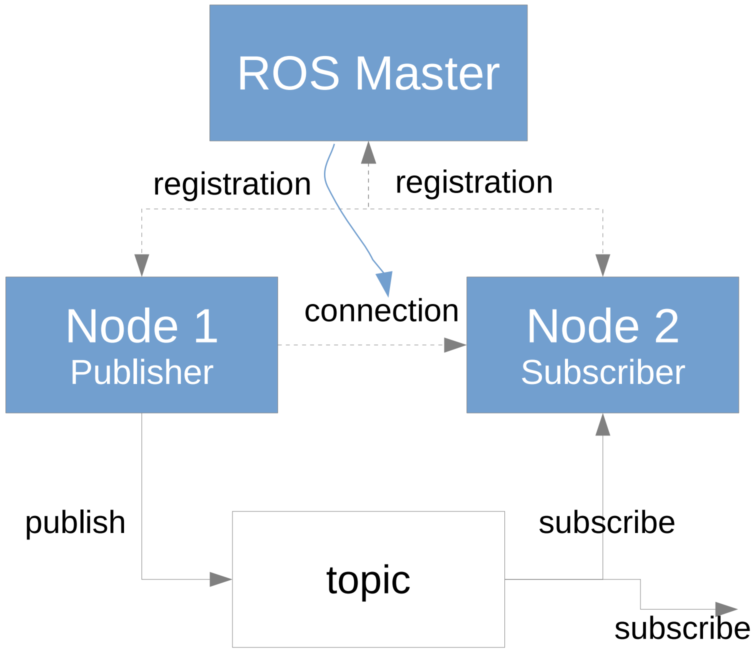


Start a master with

```
> roscore
```

ROS Topics

- Topic is a name for a « **stream of messages** »
- Nodes **communicate** over *topics*
  - Nodes can ***publish*** or ***subscribe*** to a topic
  - Typically, 1 publisher and n subscribers
  - But can possibly have many publishers and many subscribers
- A node doesn't care if no node has subscribed to his topic.
- The topics are created inside the nodes.



Subscribe and print the contents of a topic with

```
> rostopic echo /topic_name
```

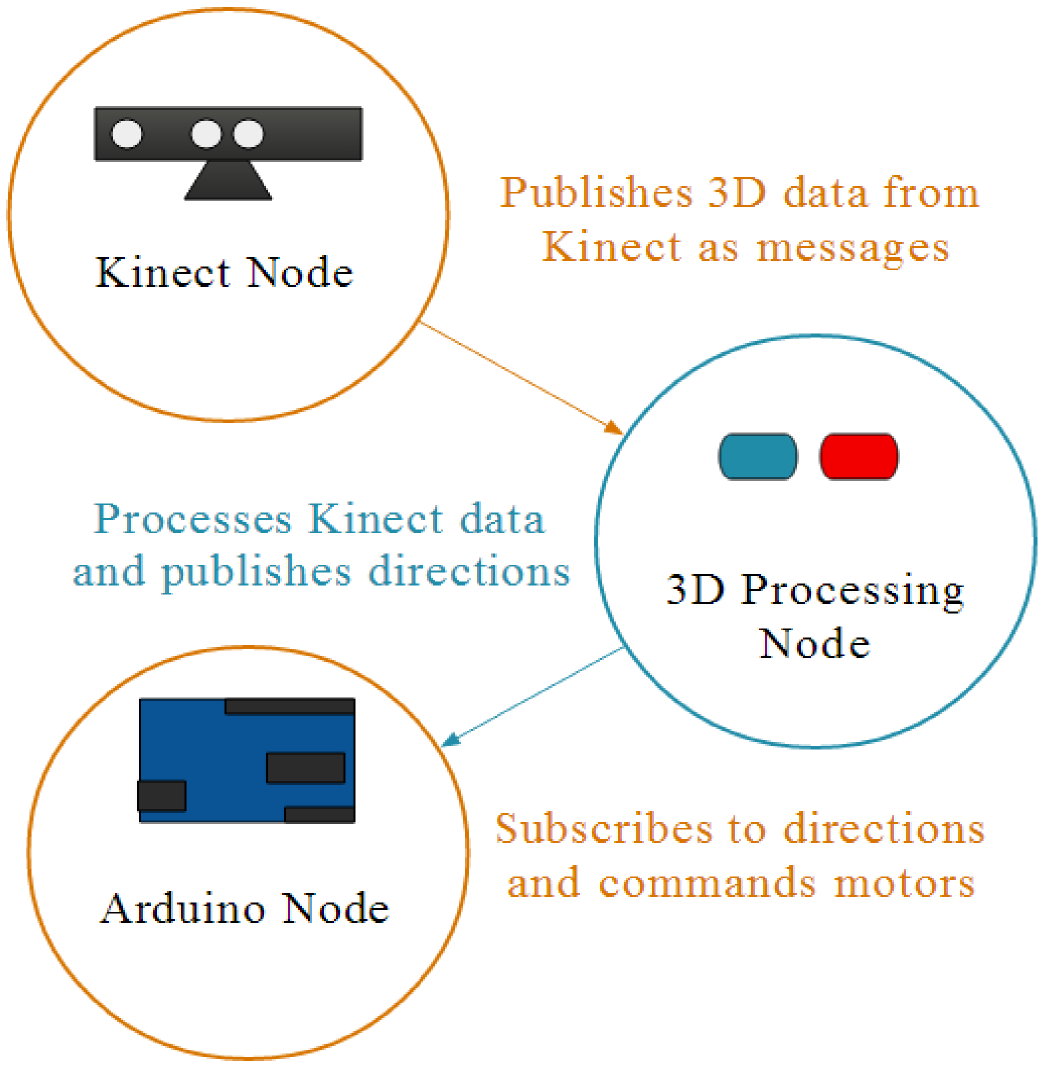
List active topics with

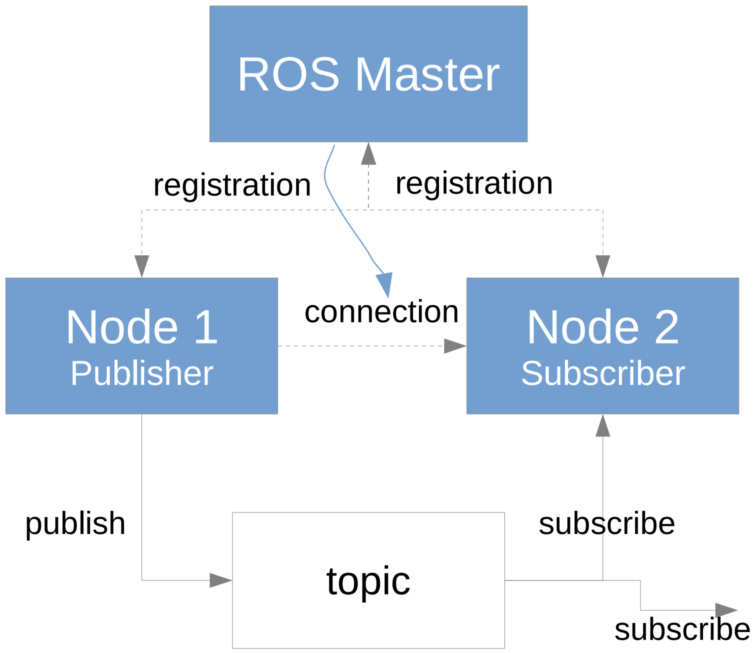
```
> rostopic list
```

Show information about a topic with

```
> rostopic info /topic_name
```

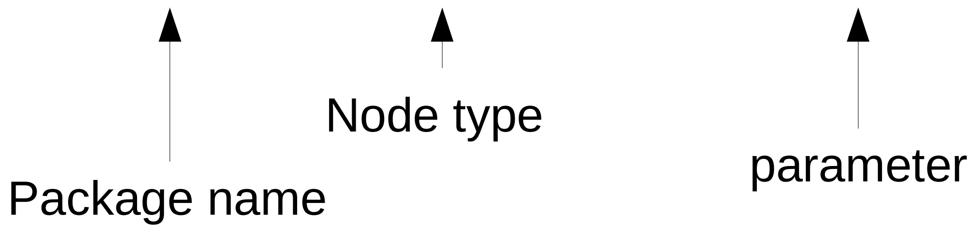
ROS Topics Example


Kinect Node
Publishes 3D data from Kinect as messages
Processes Kinect data and publishes directions
3D Processing Node
Arduino Node
Subscribes to directions and commands motors



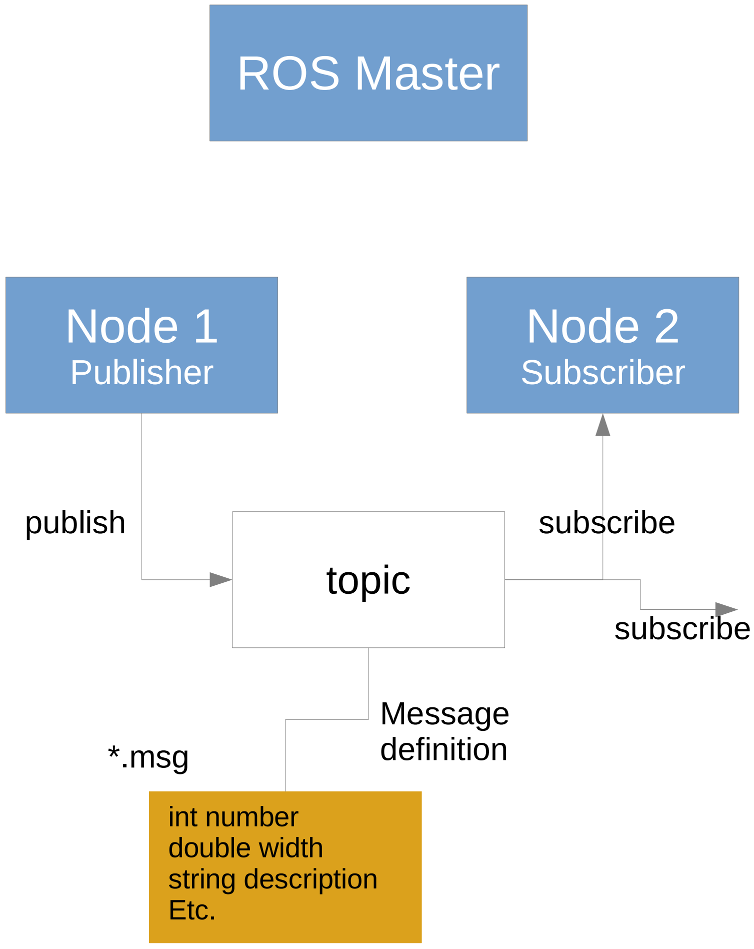
First example

- *sudo apt install ros-melodic-usb-cam*
- Three terminals :
  - In each terminal : *source /opt/ros/melodic/setup.bash*
  - Term 1 : *roscore*
  - Term 2 : *rosrun usb_cam usb_cam_node _pixel_format_:=yuyv*
  - Term 3 : *rosrun image_view image_view image:=/usb_cam/image_raw*



ROS Messages 1/2

- Message = data structure defining the ***type*** of a topic
- Data structures containing data of **various kinds** : float, string, images, booleans, etc.
- Existing list of **standard** messages : position, cmd_vel (command velocity), etc.
- Messages are sorted by **theme**: geometry, sensors, navigation, etc.:
  - std_msgs/xxx: standard messages
  - geometry_msgs/xxx: messages about geometry
  - Etc.
- Messages can be organized as a **nested structure** of messages



ROS Messages 2/2

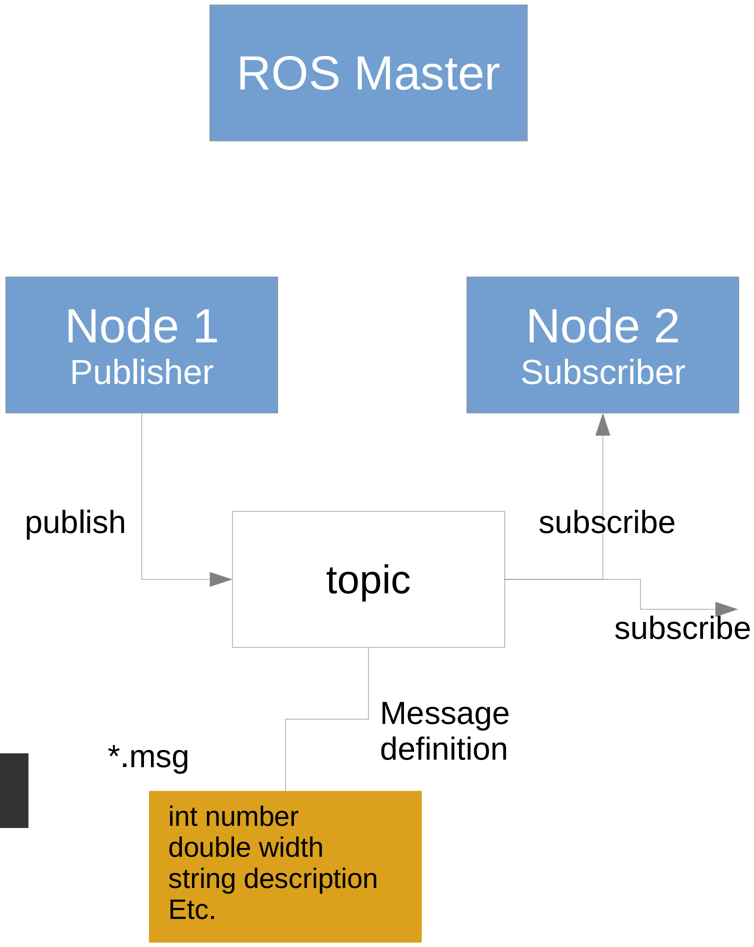
- Defined in ***.msg** files
- You can create new ones.
- But to use all the tools, it is **better to use the standard messages**.

See the type of a topic

```
> rostopic type /topic_name
```

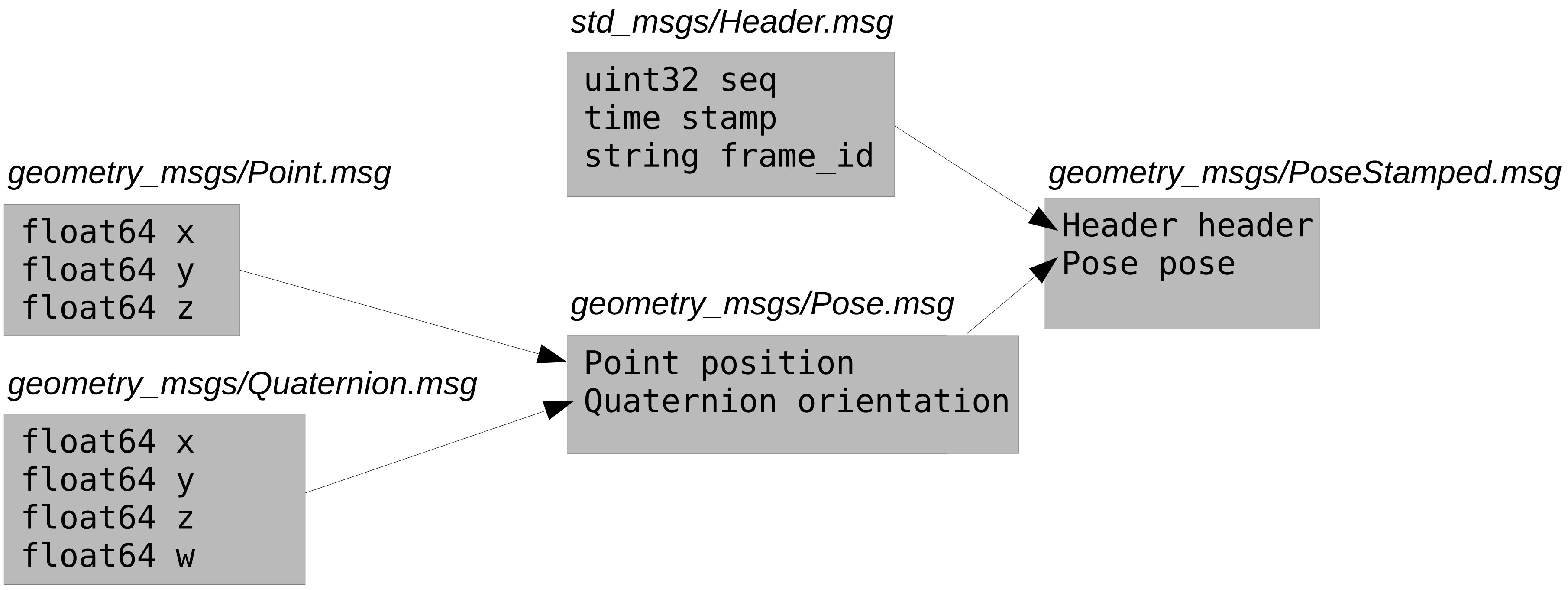
Publish a message to a topic

```
> rostopic pub /topic_name msg_type data
```



ROS Messages example

- Pose Stamped Example



```
> rosmsg show geometry_msgs/PoseStamped
```

```
> rosmsg show geometry_msgs/Pose
```

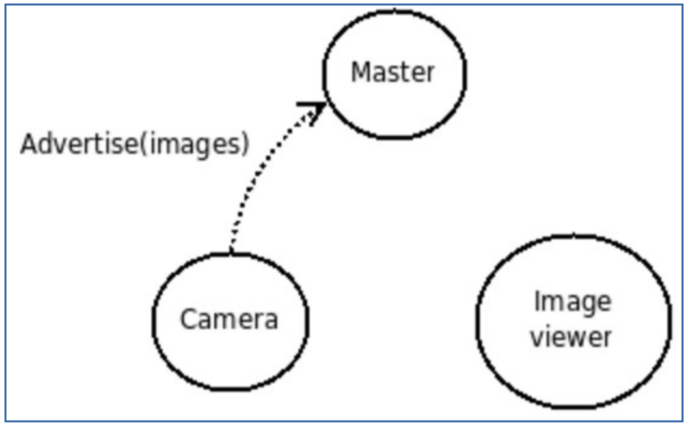
ROS Messages example

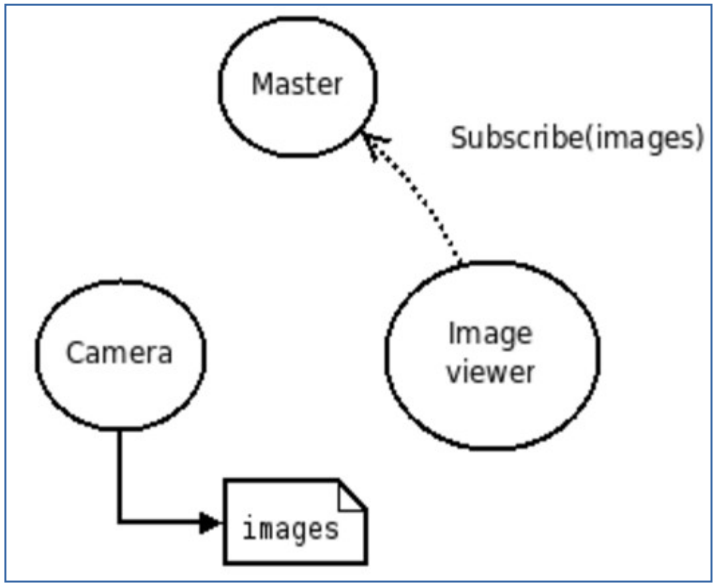
- Image Example

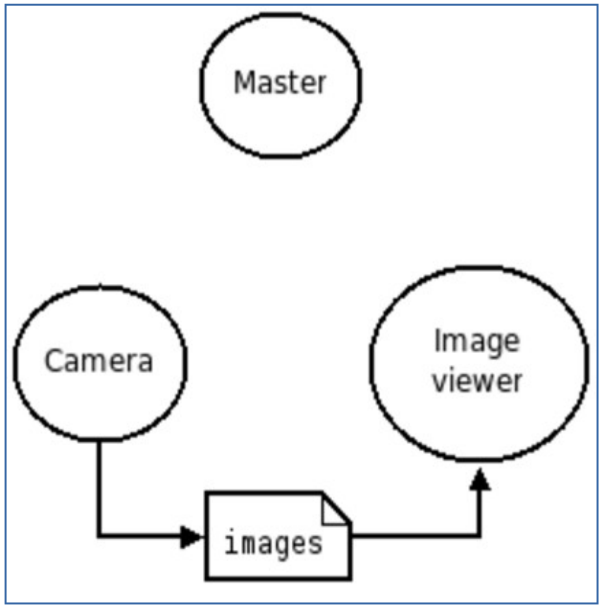
Text file: sensor_msgs/Image.msg

```
Header header        # Header timestamp should be acquisition time of image
                     # Header frame_id should be optical frame of camera
                     # origin of frame should be optical center of camera
                     # +x should point to the right in the image
                     # +y should point down in the image
                     # +z should point into to plane of the image
                     # If the frame_id here and the frame_id of the CameraInfo
                     # message associated with the image conflict the behavior is undefined

uint32 height        # image height, that is, number of rows
uint32 width         # image width, that is, number of columns

# The legal values for encoding are in file src/image_encodings.cpp
# If you want to standardize a new string format, join
# ros-users@lists.sourceforge.net and send an email proposing a new encoding.

string encoding      # Encoding of pixels -- channel meaning, ordering, size
                     # taken from the list of strings in include/sensor_msgs/image_encodings.h

uint8 is_bigendian   # is this data bigendian?
uint32 step          # Full row length in bytes
uint8[] data         # actual matrix data, size is (step * rows)
```

ROS Master in details







ROS Services

- **Request/response communication** between nodes is realized with ***services***
  - The *service* ***server*** advertises the service
  - The *service* ***client*** accesses this service
- A client node using a service will wait until the response: **blocking** behavior
- Similar in structure to messages, services are defined in ***.srv** files
- List available services with

```
> rosservice list
```

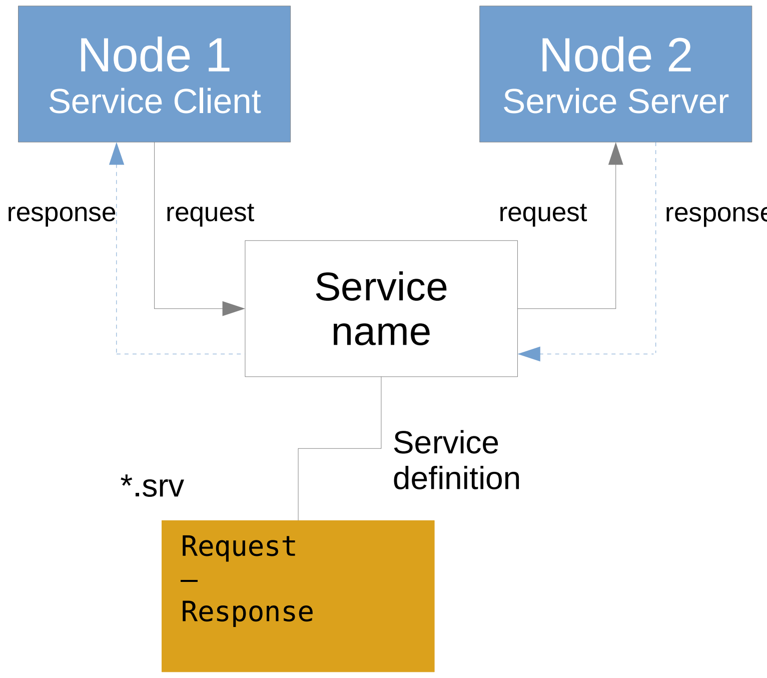
- Show the type of a service

```
> rosservice type /service_name
```

- Call a service with the request contents

```
> rosservice call /service_name args
```

Example:
map_server/static_map –
retrieves the current grid map used by the robot for navigation



ROS Services: Examples

nav_msgs/GetPlan.srv

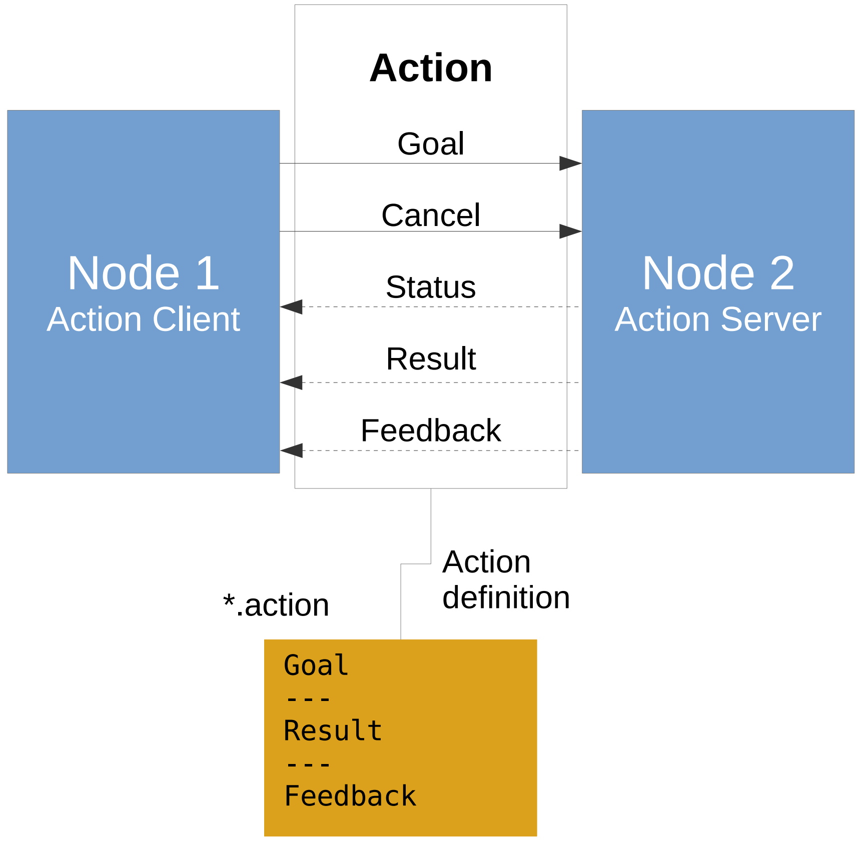
```
# Get a plan from the current
# position to the goal Pose

# The start pose for the plan
geometry_msgs/PoseStamped start

# The final pose of the goal position
geometry_msgs/PoseStamped goal

# If the goal is obstructed, how
# many meters the planner can
# relax the constraint in x
# and y before failing.
float32 tolerance
---
nav_msgs/Path plan
```

std_srvs/Trigger.srv

```
---
# indicate successful
# run of triggered service
bool success

# informational, e.g.
# for error messages
string message
```

ROS Actions (actionlib)

- Similar to service calls, but provide possibility to
  - Cancel the task (preempt)
  - Receive feedback on the progress
- Best way to implement interfaces to long, goal-oriented behaviors
- Non-blocking behavior
- Similar in structure to services, actions are defined in **.action* files
- Internally, actions are implemented with a set of topics



```
Goal
---
Result
---
Feedback
```

Topics, Services, and Actions Comparison

- Topics
  - **Description**: continuous data streams
  - **Application**: one-way continuous data flow
  - **Examples**: sensor data, robot state
- Services
  - **Description**: blocking call for processing a request
  - **Application**: short triggers or calculations
  - **Examples**: trigger change, request state, compute quantity
- Actions
  - **Description**: non-blocking, preemptable goal-oriented tasks
  - **Application**: task executions and robot actions
  - **Examples**: navigation, grasping, motion execution

ROS Architecture

- Each node is a different process
- Inter process communication
  - Direct communications between node
  - through TCP/IP or UDP
  - Easy on multiple computers (set ROS_MASTER_URI)
  - Shared memory (nodelet) on single computer: avoid to copy and use of lot of memory.
- Rospy, Roscpp, …
  - The libraries to interact with ROS network in various languages

Exercice 1 – chatter/listener

- Live demonstration
- topics covered:
  - Launch roscore
  - Launch node *talker* and *listener* of package *roscpp_tutorials*
  - Use tools to analyze
  - Publish a message

Exercice 2 – Turtlesim

- Live demonstration
- topics covered:
  - Launch roscore
  - Launch node *turtlesim_node* and *turtle_teleop_key* of package *turtlesim*
  - Use tools to analyze
  - Publish a message to control the turtle

Course Summary

- Why does ROS exist?
- How does ROS work?
- How to use ROS with your own code ?

Workspace: catkin 1/2

- The ROS packages used in your future project will come from :
  - **Pre-installed packages**, locate in */opt/ros/melodic/*
  - **Newly installed packages**, locate in */opt/ros/melodic/*
  - **downloaded** package, usually from github
  - Your **own** self-coded package
- The last two must be **compiled** before being used !
- ***Catkin*** is the name of the **ROS build system** to generate executables, libraries, and interfaces
- A ***catkin workspace*** is the place in which one or more catkin packages can be built.

Workspace: catkin 2/2

- The first time, to create a *catkin workspace*:

```
> mkdir -p ~/catkin_ws/src
> cd ~/catkin_ws/src
> catkin_init_workspace
```

- The first build in your *catkin workspace*:

```
> cd ~/catkin_ws/
> catkin_make
```

- ⇒ Create the environment to develop new packages
- ⇒ 3 folders *build*, *devel* and *src*

Workspace: folders

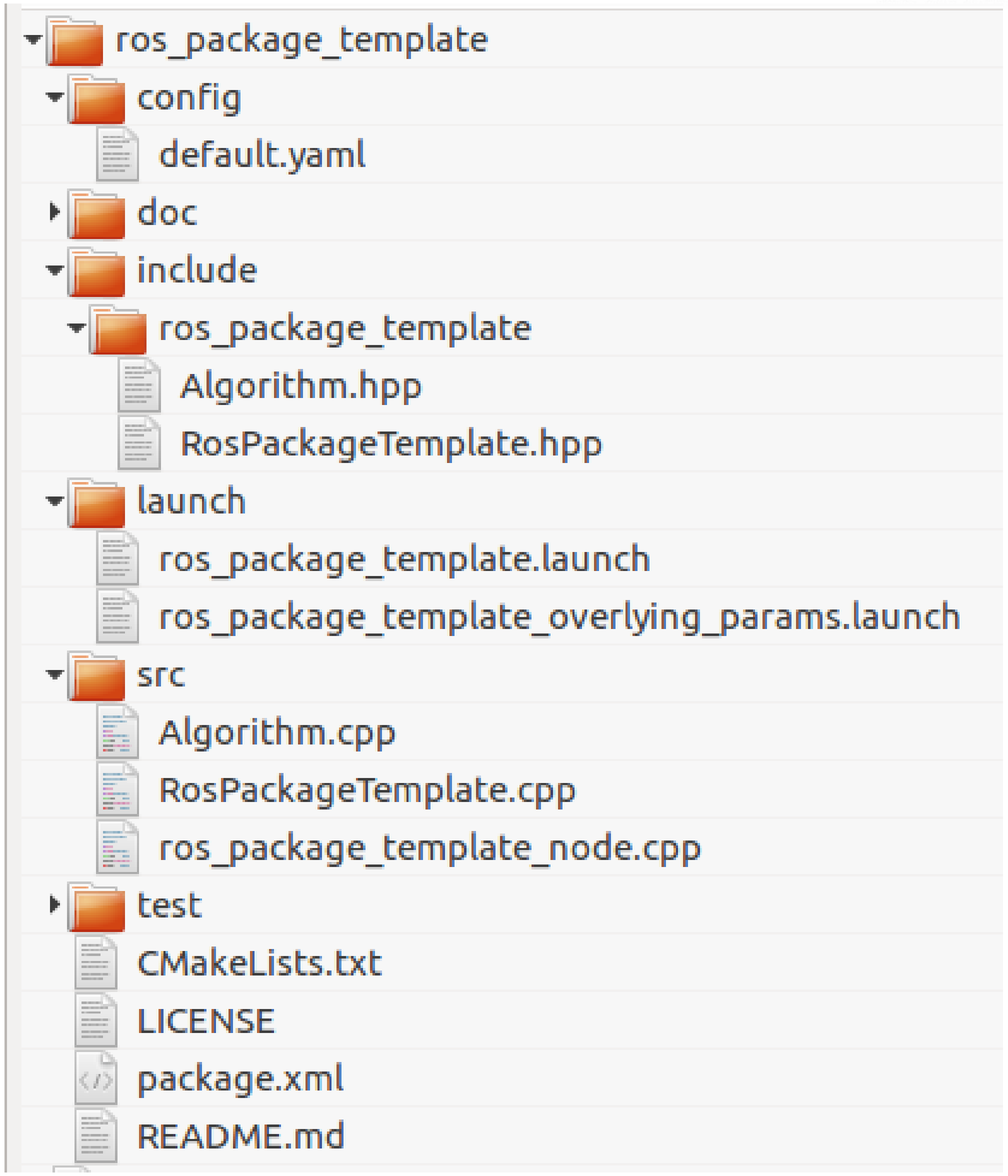
- In your catkin workspace, you have 3 folders *build*, *devel* and *src*
  - ***src***: *Work here*
    - The source space contains the source code. This is where you can clone, create, and edit source code for the packages you want to build, i.e. the ones you have created or the ones you have downloaded
  - ***build****: usually, it should not be touched.*
    - *The build space is where CMake is invoked to build the packages in the source space. Cache information and other intermediate files are kept here.*
  - ***devel****: usually, it should not be touched.*
    - *The development (devel) space is where built targets are placed (prior to being installed).*
- If necessary, clean the entire build and devel space simply by deleting *build* and *devel* folder

Workspace: package 1/2

- The pre-installed packages are in ***/opt/ros/melodic/***
- Your **own** package or **downloaded** package should be placed in the ***`~/catkin_ws/src`* folder**
- Technically, a package directory is a directory which contains a file ***package.xml*** describing the package.
- If you rename *package.xml*, the package becomes invisible for ROS.

Workspace: package 2/2

- A package directory follows a common structure:
  - Package.xml
  - CmakeLists.txt
  - src / include
  - Etc.



```
ros_package_template
  config
    default.yaml
  doc
  include
    ros_package_template
      Algorithm.hpp
      RosPackageTemplate.hpp
  launch
    ros_package_template.launch
    ros_package_template_overlying_params.launch
  src
    Algorithm.cpp
    RosPackageTemplate.cpp
    ros_package_template_node.cpp
  test
  CMakeLists.txt
  LICENSE
  package.xml
  README.md
```

Workspace: setup.bash

- Default workspace is loaded with:

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

- Overlay your catkin workspace with:

```
> source ~/catkin_ws/devel/setup.bash
```

- Check your workspace with

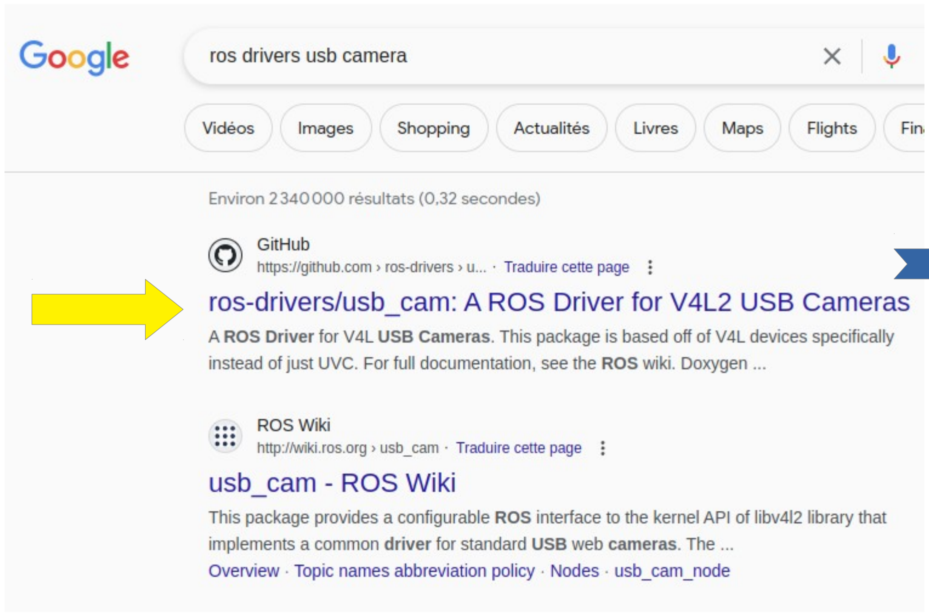
```
> echo $ROS_PACKAGE_PATH
```

- Each time we want to open a terminal to run a ROS command, we have to execute this *setup.bash* file
- Good **idea**: put the two first commands at the end of the *.bashrc* file.

Workspace: add a new package from source

- Open a terminal and go in your workspace

```
> cd ~/catkin_ws/src/
```

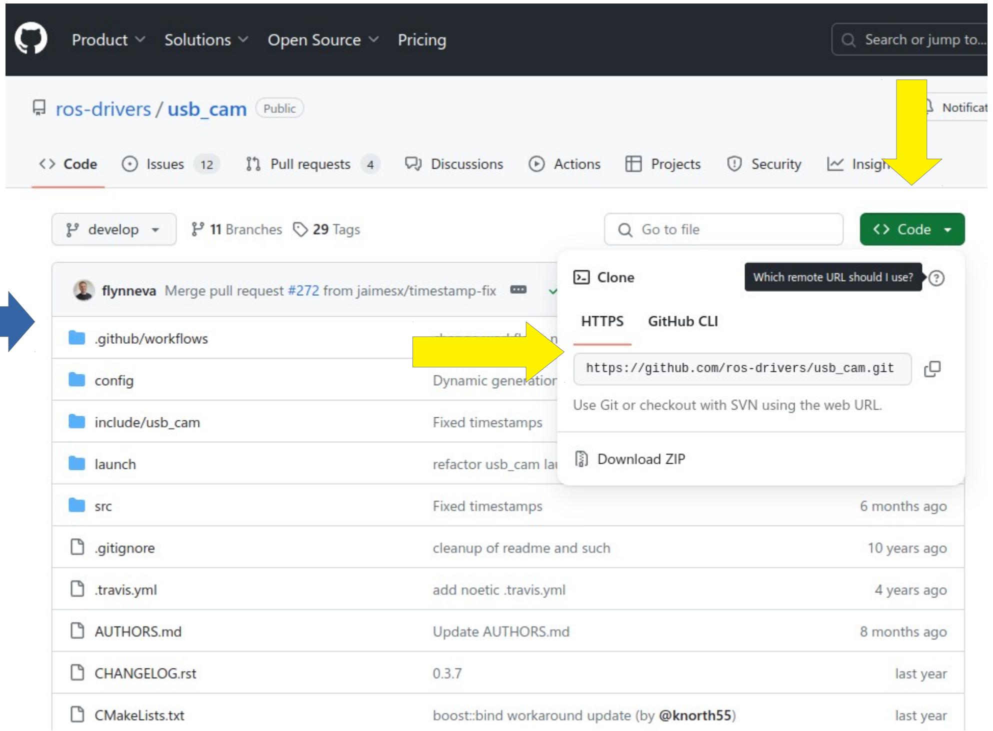


ros drivers usb camera

Environ 2 340 000 résultats (0,32 secondes)

GitHub
https://github.com › ros-drivers › u... · Traduire cette page
ros-drivers/usb_cam: A ROS Driver for V4L2 USB Cameras
A **ROS Driver** for V4L **USB Cameras**. This package is based off of V4L devices specifically instead of just UVC. For full documentation, see the **ROS** wiki. Doxygen ...

ROS Wiki
http://wiki.ros.org › usb_cam · Traduire cette page
usb_cam - ROS Wiki
This package provides a configurable **ROS** interface to the kernel API of libv4l2 library that implements a common **driver** for standard **USB** web **cameras**. The ...
Overview · Topic names abbreviation policy · Nodes · usb_cam_node



ros-drivers / usb_cam Public

11 Branches 29 Tags

flynneva Merge pull request #272 from jaimesx/timestamp-fix

https://github.com/ros-drivers/usb_cam.git

Use Git or checkout with SVN using the web URL.

.github/workflows		
config	Dynamic generation	
include/usb_cam	Fixed timestamps	
launch	refactor usb_cam lau	
src	Fixed timestamps	6 months ago
.gitignore	cleanup of readme and such	10 years ago
.travis.yml	add noetic .travis.yml	4 years ago
AUTHORS.md	Update AUTHORS.md	8 months ago
CHANGELOG.rst	0.3.7	last year
CMakeLists.txt	boost::bind workaround update (by @knorth55)	last year

- Clone the Git repository of the package, for example:

```
> git clone https://github.com/ros-drivers/usb_cam.git
```

Workspace: compile the new package

- Go to your catkin workspace

```
> cd ~/catkin_ws
```

- (Here we specifically need to install before *libv4l* for *usb_cam*)

```
> sudo apt install libv4l-dev
```

- Build the package with

```
> catkin_make
```

- Re-source your workspace setup

```
> source devel/setup.bash
```

- Launch the node with *roslaunch*, for example:

```
> roslaunch usb_cam test_img_view.launch
```

- *Ctrl+C* to stop the program

ROS Launch

- *launch* is a tool for launching multiple nodes (as well as setting parameters)
- Are written in XML as *.*launch* files
- If not yet running, launch automatically starts a roscore
- A launch file can be executed in two ways:
  - Browse to the folder and start a launch file with

```
> roslaunch file_name.launch
```

  - Start a launch file from a package with

```
> roslaunch package_name file_name.launch
```

ROS Launch: file structure

```
<launch>

  <node name="my_usb_cam"    pkg="usb_cam"    type="usb_cam_node" output="screen"/>

  <node name="my_image_view" pkg="image_view" type="image_view"   output="screen"/>

</launch>
```

- **launch**: Root element of the launch file
- **node**: Each <*node*> tag specifies a node to be launched
- **name**: Name of the node (free to choose). Two nodes with the same type should have different names.
- **pkg**: Package containing the node
- **type**: Type of the node, there must be a corresponding executable with the same name
- **output**: Specifies where to output log messages (screen: console, log: log file)

ROS Launch: arguments

- Create reusable launch files with <*arg*> tag, which works like a parameter (default optional)

```
<arg name="arg_name" default="default_value"/>
```

- Use arguments in launch file with

```
$(arg arg_name)
```

- When launching, arguments can be set with

```
> roslaunch launch_file.launch arg_name:=value
```

- More info: http://wiki.ros.org/roslaunch/XML/arg

usb_cam.launch

```
<launch>
  <arg name="show_camera" default="false" />

  <node name="usb_cam" pkg="usb_cam" type="usb_cam_node" output="screen" >
      <rosparam command="load" file="$(find usb_cam)/config/usb_cam.yml"/>
  </node>
  <node if="$(arg show_camera)" name="image_view" pkg="image_view" type="image_view"
        respawn="false" output="screen">
    <remap from="image" to="/usb_cam/image_raw"/>
    <param name="autosize" value="true" />
  </node>
</launch>
```

ROS Launch: Including Other Launch Files

- Include other launch files with <*include*> tag to organize large projects

```
<include file="package_name"/>
```

- Find the system path to other packages with

```
$(find package_name)
```

- Pass arguments to the included file set with

```
<arg name="arg_name" value="value"/>
```

- More info: http://wiki.ros.org/roslaunch/XML/include

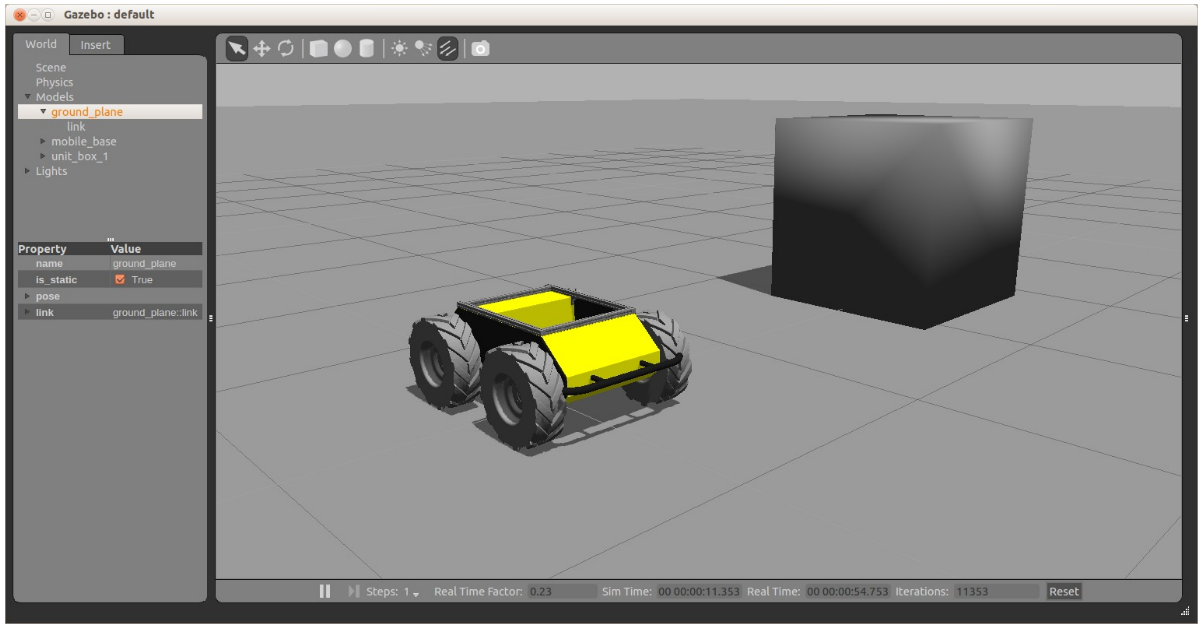
test_img_view.launch

```
<launch>
  <arg name="show_image" default="true" />

  <include file="$(find usb_cam)/launch/usb_cam.launch">
    <arg name="show_camera" value="$(arg show_image)" />
  </include>

</launch>
```

Gazebo Simulator

- Simulate 3d rigid-body dynamics
- Simulate a variety of sensors including noise
- 3d visualization and user interaction
- Includes a database of many robots and
- environments (Gazebo worlds)
- Provides a ROS interface
- Extensible with plugins

```
> rosrun gazebo_ros gazebo
```



Property	Value
name	ground_plane
is_static	True
pose	
link	ground_plane::link

```
Steps: 1   Real Time Factor: 0.23   Sim Time: 00 00:00:11.353   Real Time: 00 00:00:54.753   Iterations: 11353
```

Exercice 3 – play with husky

- Topics covered:
  - Gazebo
  - ROS architecture
  - ROS master, nodes, and topics
  - Console commands
  - Catkin workspace and build system
  - Launch-files

Further References

- **ROS Wiki:**
  - http://wiki.ros.org/
- **Installation:**
  - http://wiki.ros.org/ROS/Installation
- **Tutorials:**
  - http://wiki.ros.org/ROS/Tutorials
- **Available packages:**
  - http://www.ros.org/browse/
- **ROS Cheat Sheet:**
  - https://www.clearpathrobotics.com/ros-robot-operating-system-cheat-sheet/
  - https://kapeli.com/cheat_sheets/ROS.docset/Contents/Resources/Documents/index
- **ROS Best Practices:**
  - https://github.com/leggedrobotics/ros_best_practices/wiki
- **ROS Package Template:**
  - https://github.com/leggedrobotics/ros_best_practices/tree/master/ros_package_template