Robotique Mobile
04 - Software architectures - ROS

David Filliat

Alexandre Chapoutot

Goran Frehse

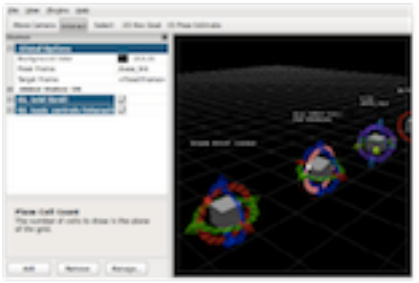
prenom.nom@ensta-paris.fr

Robot Operating System

- Software architecture for robots (middleware)
- Different from control architecture

=

+



Plumbing

Tools

+

+

Capabilities

Ecosystem

ROS Philosophy

- Peer to peer
  - Individual programs communicate over defined API (ROS messages, services, etc.).
- Distributed
  - Programs can be run on multiple computers and communicate over the network.
- Multi-lingual
  - ROS modules can be written in any language for which a client library exists (C++, Python, MATLAB, Java, etc.).
- Software reuse
  - No need to reinvent the wheel every time.
- Free and open-source
  - Most ROS software is open-source and free to use.

Development

- Started at STANFORD ~2005
- Carried by Willow Garage 2007 - 2013
- Now by Open Source Robotics Foundation

Installation

- Mainly on Linux Ubuntu
- Current 'long support' version ROS Melodic + ubuntu 18.04
- Quite easy to install

sudo apt install ros-melodic-desktop-full

Robots

- Hundreds of robots : http://wiki.ros.org/Robots

ROS 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 Jade Turtle	May 23rd, 2015			May, 2017

- For ROS Melodic

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

Packages

- Unit (directory) grouping elements (source code, scripts, launch files, libraries …) related to one function
- Defined with Manifest “package.xml”

  Version / purpose / dependencies …
- Building tool : catkin (using cmake)

```
catkin_create_pkg beginner_tutorials std_msgs rospy roscpp
. ~/catkin_ws/devel/setup.bash
catkin_make
```

Meta file system

- roscd, rosls
- rospack Ex : `rospack find [package_name]`

Node

- The unit of execution in ROS applications

Messages

- Data structures containing data of various kind (float, string, images ….)
- Existing list of std messages (position, cmd_vel …) and message ontology (geometry, sensors, navigation ...)

Topic

- publish/subscribe communication channel transmitting messages

Service

- Request/response communication channel

Actions

- Similar to services, for long / preemptible tasks

ROS Nodes

- **Single-purposed executable in ROS applications** : e.g. sensor driver(s), actuator driver(s), mapper, planner, UI, etc.
- Individually compiled, executed, and managed
- Organized in *packages*
- Same node can be launched several times but with a different name.

Run a node with

```
> rosrun package_name node_type
```

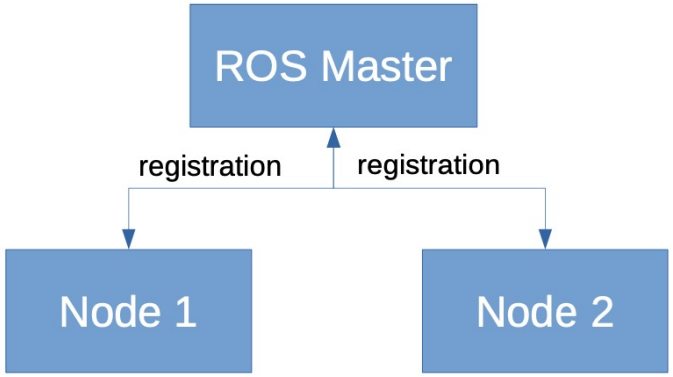
See active nodes with

```
> rosnode list
```

Retrieve information about a node with

```
> rosnode info node_type
```

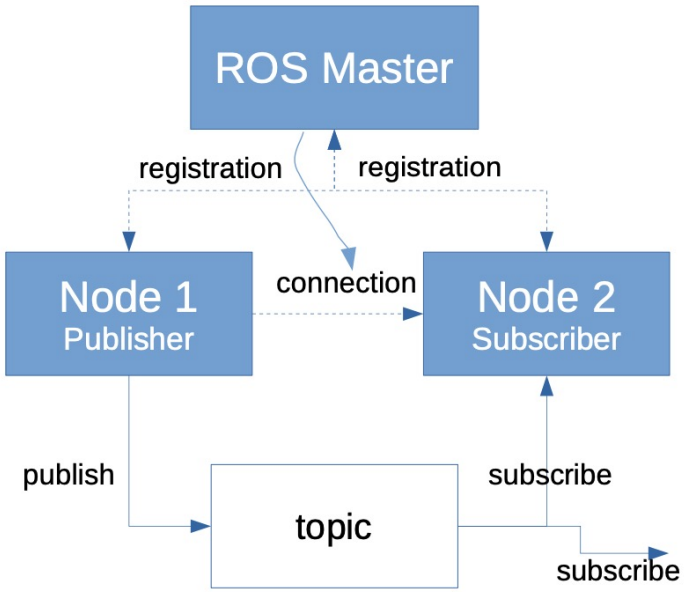
- Manages the communication between nodes (processes)
- Every node registers at startup with the master
- Host a parameter server



Start a master with

```
> roscore
```

- 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
- Topic is a name for a stream of messages
- A node doesn't care if no node has subscribed to his topic.



List active topics with

```
> rostopic list
```

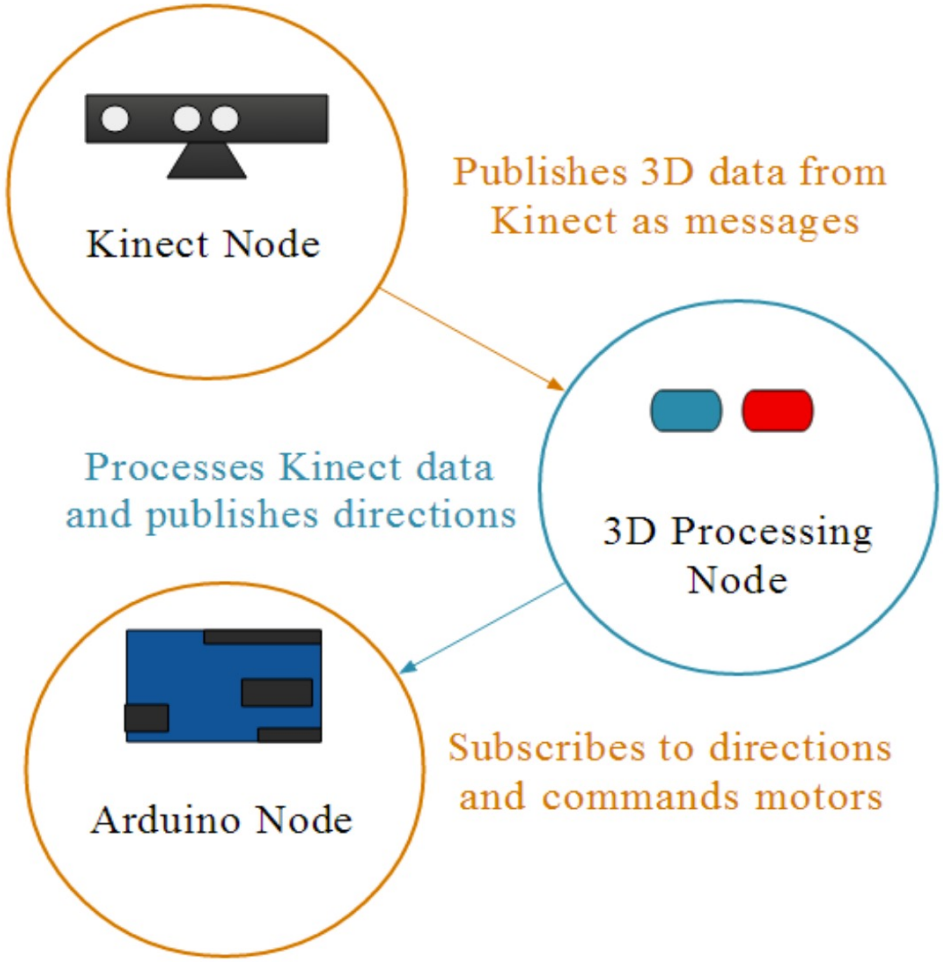
Subscribe and print the contents of a topic with

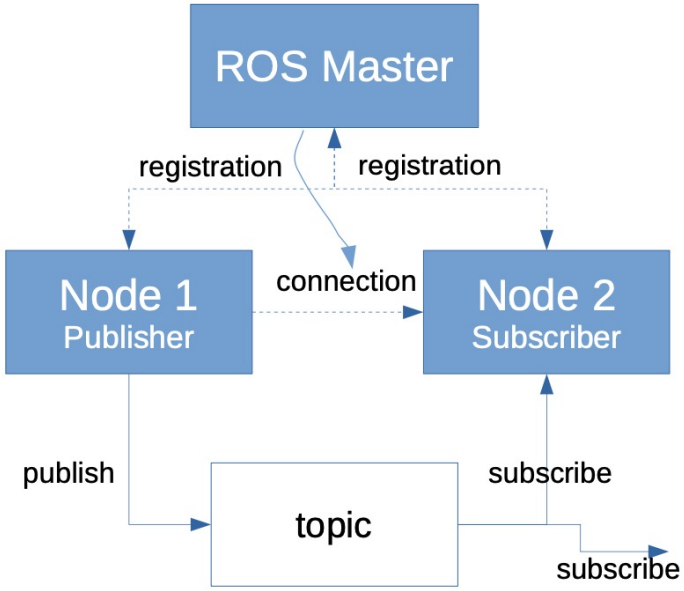
```
> rostopic echo /topic_name
```

Show information about a topic with

```
> rostopic info /topic_name
```

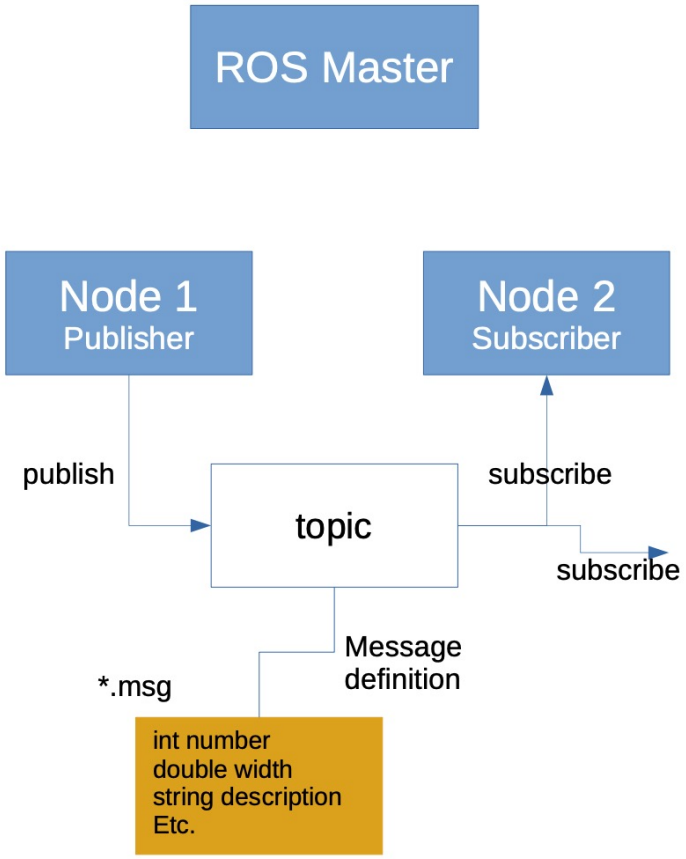
ROS Topic Example



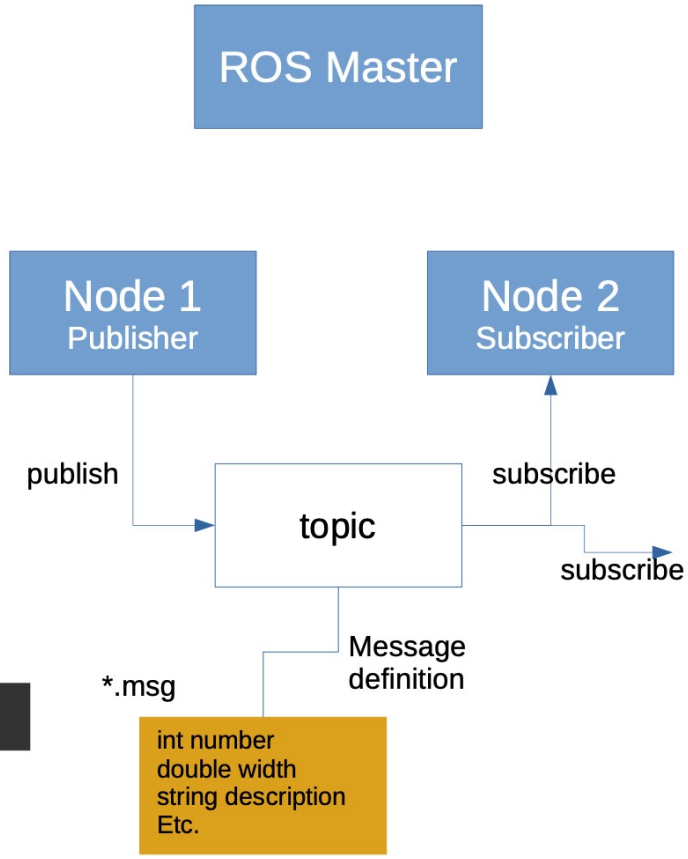


ROS Messages

- Data structure defining the *type* of a topic
- Data structures containing data of various kind (float, string, images, booleans ....)
- Existing list of standard messages (position, cmd_vel ...)
- message ontology (geometry, sensors, navigation ...) :
  - std_msgs/xxx : standard messages
  - geometry_msgs/xxx : messages about geometry
  - Etc.
- Messages can be organized as a nested structure of messages



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



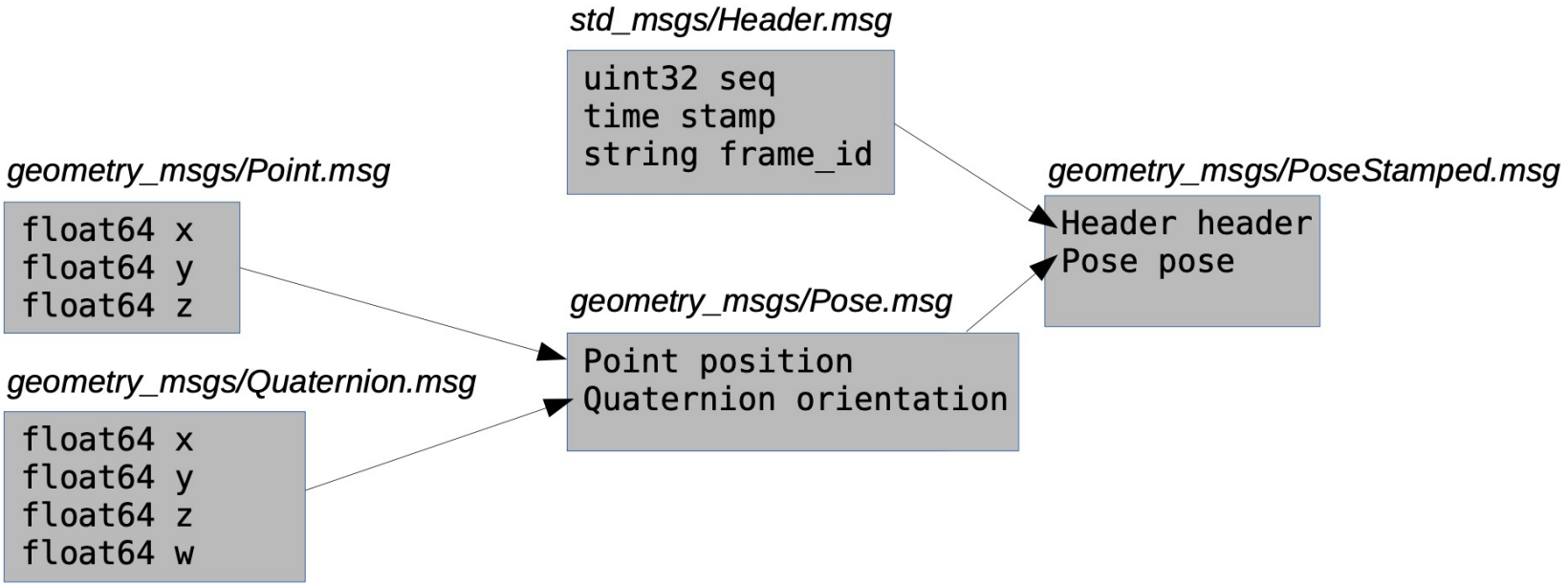
See the type of a topic

```
> rostopic type /topic_name
```

Publish a message to a topic

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

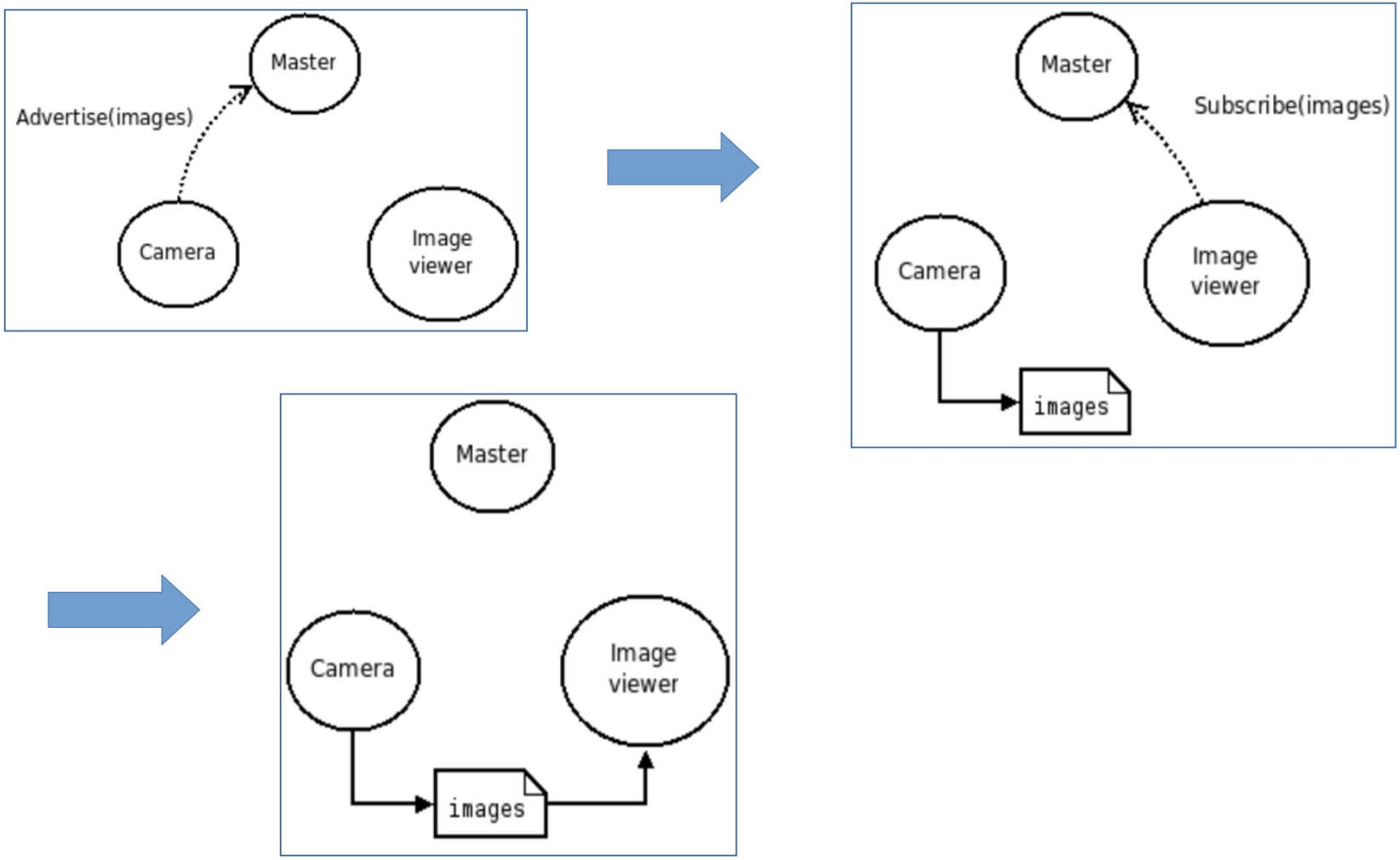
- Pose Stamped Example



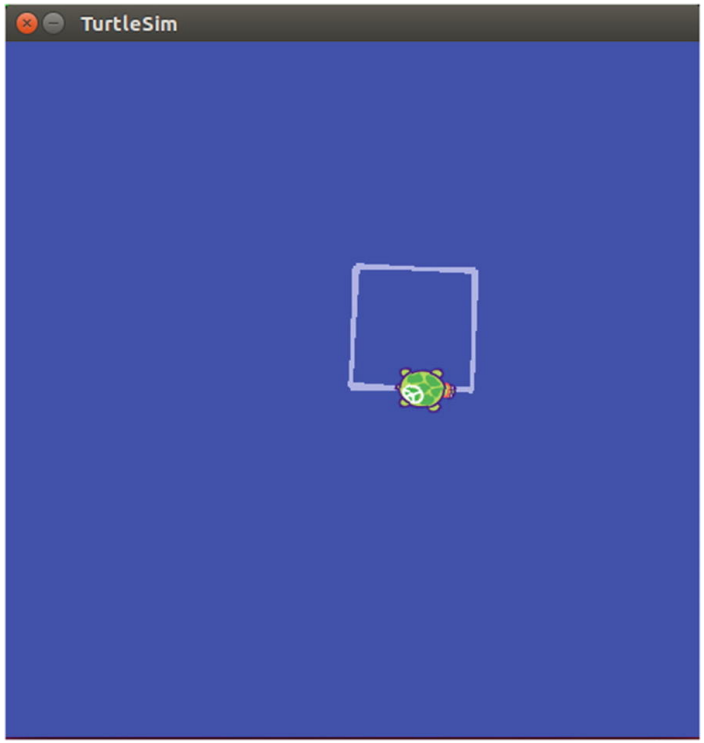

```
> rosmsg show geometry_msgs/PoseStamped
```

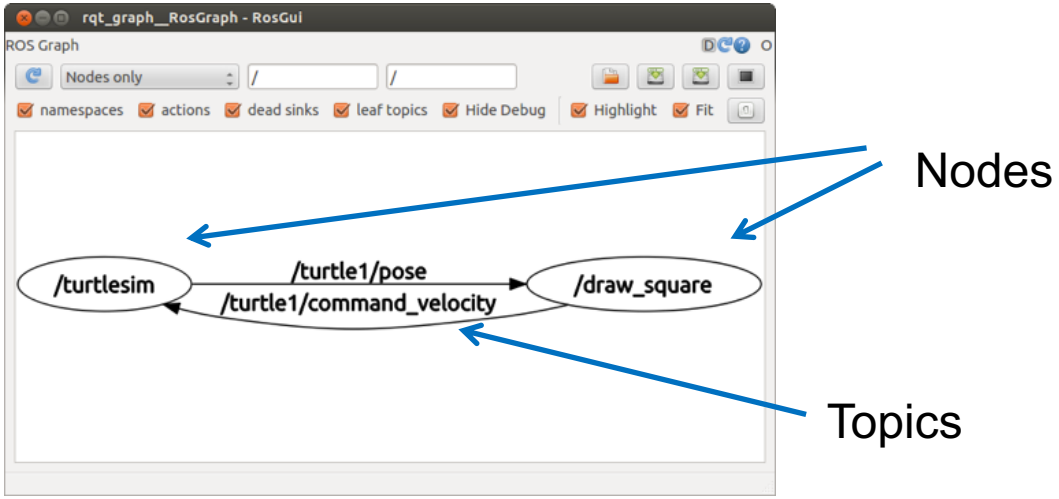
```
> rosmsg show geometry_msgs/Pose
```

ROS Master in details



Example application



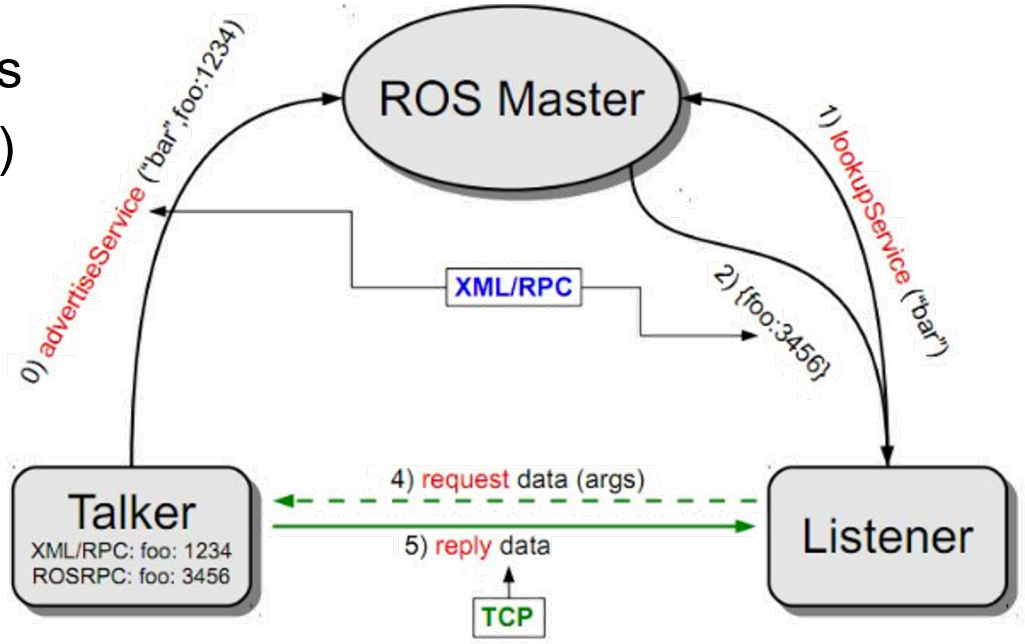


Roscore (ROS Master)

- Main ros process, handle connections between nodes
- Host a parameter server

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



Rospy, RosCPP, …

- The libraries to interact with ros network in various languages

Publisher python example

```python
#!/usr/bin/env python
# license removed for brevity
import rospy
from std_msgs.msg import String

def talker():
    pub = rospy.Publisher('chatter', String, queue_size=10)
    rospy.init_node('talker', anonymous=True)
    rate = rospy.Rate(10) # 10hz
    while not rospy.is_shutdown():
        hello_str = "hello world %s" % rospy.get_time()
        rospy.loginfo(hello_str)
        pub.publish(hello_str)
        rate.sleep()

if __name__ == '__main__':
    try:
        talker()
    except rospy.ROSInterruptException:
        pass
```

Subscriber python example

```python
#!/usr/bin/env python
import rospy
from std_msgs.msg import String

def callback(data):
    rospy.loginfo(rospy.get_caller_id() + "I heard %s", data.data)

def listener():

    # In ROS, nodes are uniquely named. If two nodes with the same
    # node are launched, the previous one is kicked off. The
    # anonymous=True flag means that rospy will choose a unique
    # name for our 'listener' node so that multiple listeners can
    # run simultaneously.
    rospy.init_node('listener', anonymous=True)

    rospy.Subscriber("chatter", String, callback)

    # spin() simply keeps python from exiting until this node is stopped
    rospy.spin()

if __name__ == '__main__':
    listener()
```

rosrun

– Run an executable in a package

rosrun package executable

roslaunch

– Run a set of nodes defined in a « launch file »

roslaunch package fulldemo.launch

```
<launch>
   <!-- a basic listener node -->
  <node name="listener-1" pkg="rospy_tutorials" type="listener" />
  <!-- pass args to the listener node -->
  <node name="listener-2" pkg="rospy_tutorials" type="listener" args="-foo
arg2" />
  <!-- a respawn-able listener node -->
  <node name="listener-3" pkg="rospy_tutorials" type="listener"
respawn="true" />
</launch>
```

rostopic

- Display/send/stats on topics

```
rostopic echo
rostopic bw
rostopic hz
rostopic pub
```

rosservice

- similar for services

rosnode

- list / info / kill / clean nodes

```
rosnode list
…
```

Workspace: catkin

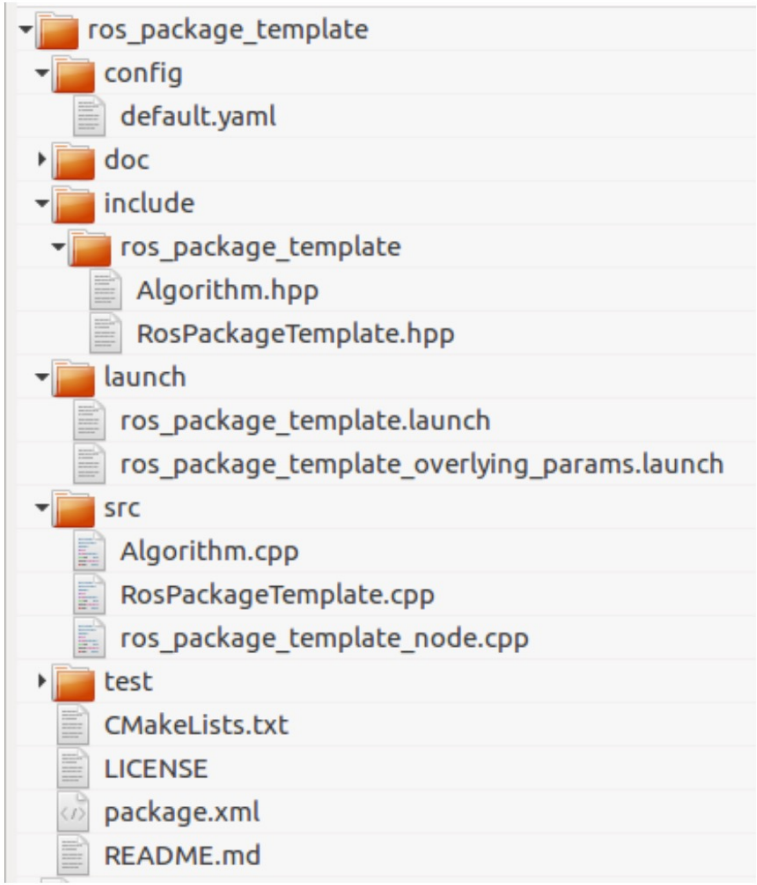
- Catkin is the ROS build system to generate executables, libraries, and interfaces
- A *catkin workspace* is the place in which one or more catkin packages can be built.
- Create and build a *catkin workspace*:

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

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

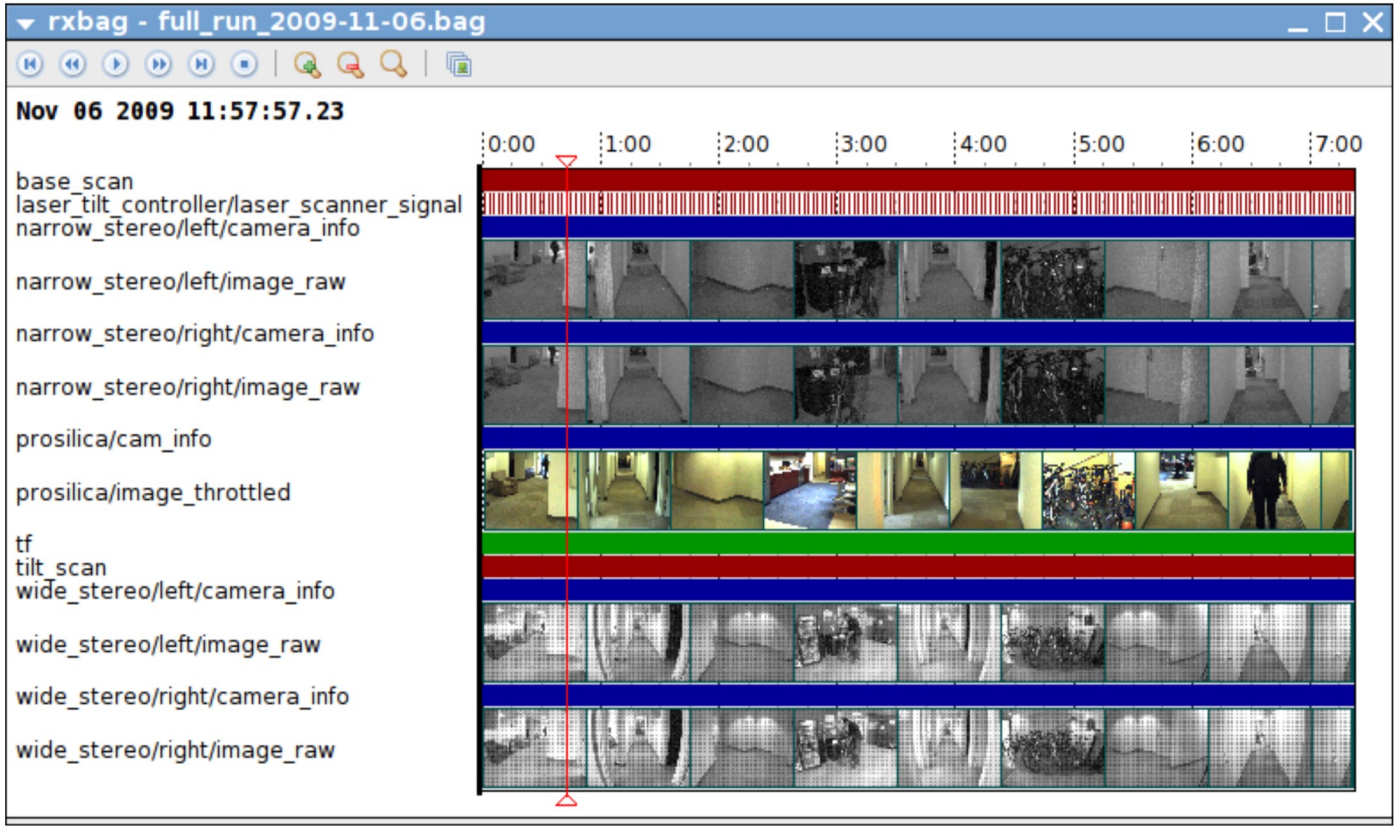
- In your catkin workspace, you have 3 folders *build*, *devel* and *src*
  - ***src*** : *Work here*
    - The source space contains the source code. This is where you can clone, create, and edit source code for the packages you want to build.
  - ***build*** *: Don't touch*
    - *The build space is where CMake is invoked to build the packages in the source space. Cache information and other intermediate files are kept here.*
  - ***devel*** *: Don't touch*
    - *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

- Your own package or dowloaded package should place in the `~/catkin_ws/src` folder
- The pre-installed package are in */opt/ros/melodic/*
- Technically, a package directory is a directory which contains a file *package.xml* describing the package
- A package directory follow a common structure.



rosbag

- Allows to record / replay messages
- Possibility to filter / change frequency …
- Rqt_bag for display



rosparam

– Get/set params in the roscore

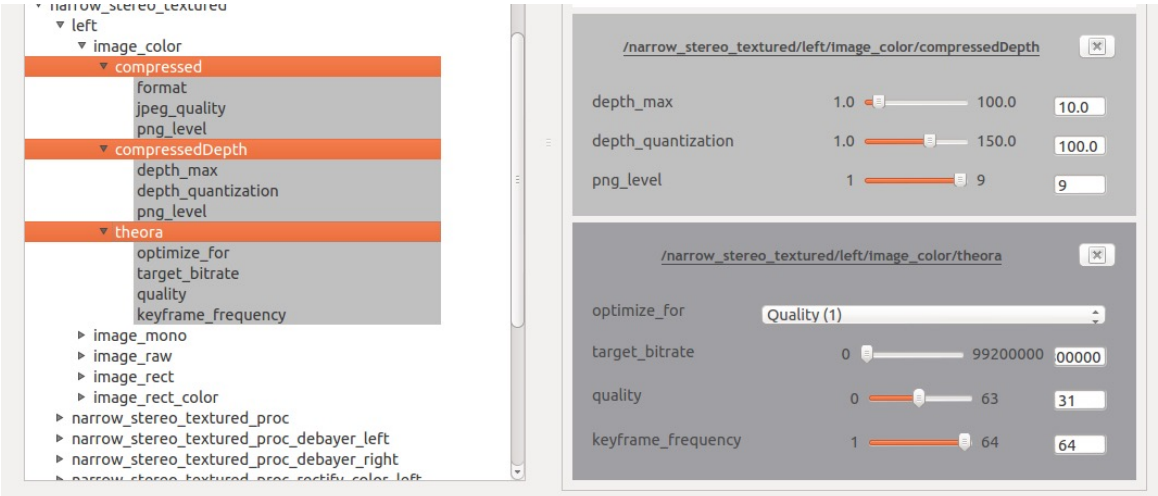
```
rosparam set
rosparam get
```

– Accessible through roscpp/rospy…

```
global_name = rospy.get_param("/global_name")
rospy.set_param('a_string', 'baz')
```

dynamic reconfigure

– package to dynamically change node params

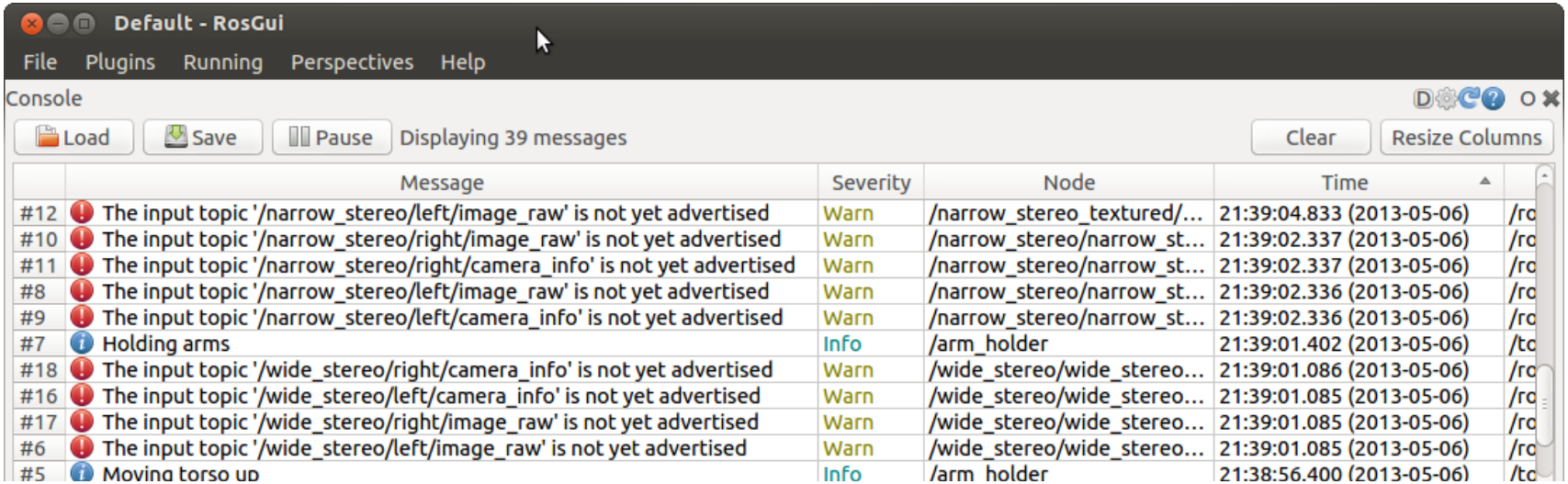


rosconsole

- Logging capability in roscpp (also in rospy)

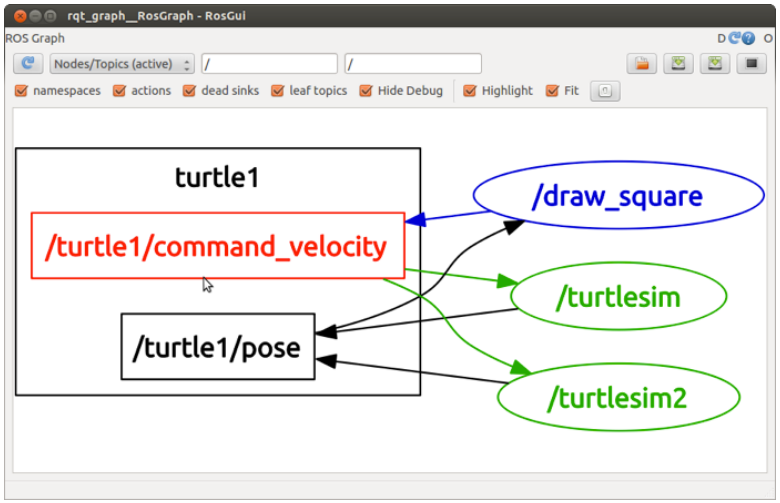
```
#include <ros/console.h>
ROS_DEBUG("Hello %s", "World");
ROS_DEBUG_STREAM("Hello " << "World");
```

- Several verbosity level (DEBUG, INFO, WARN …)
- Messages send to /rosout
- Use rqt_console for visualisation/filtering

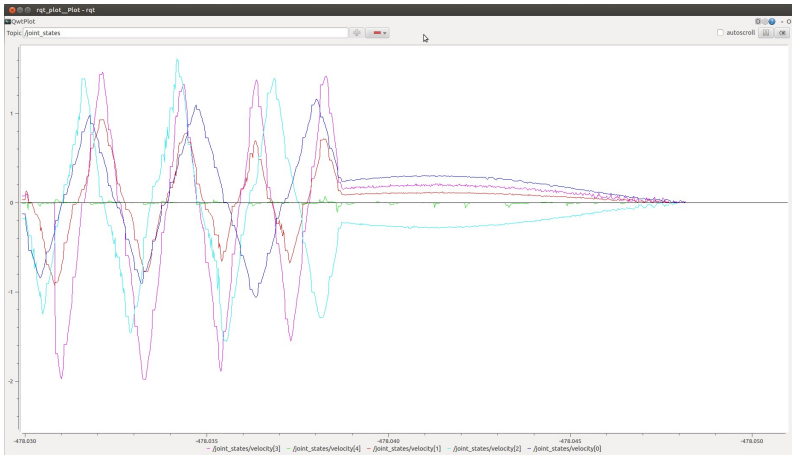


RQT

- Graphical User Interface with pugins
- Rqt_graph

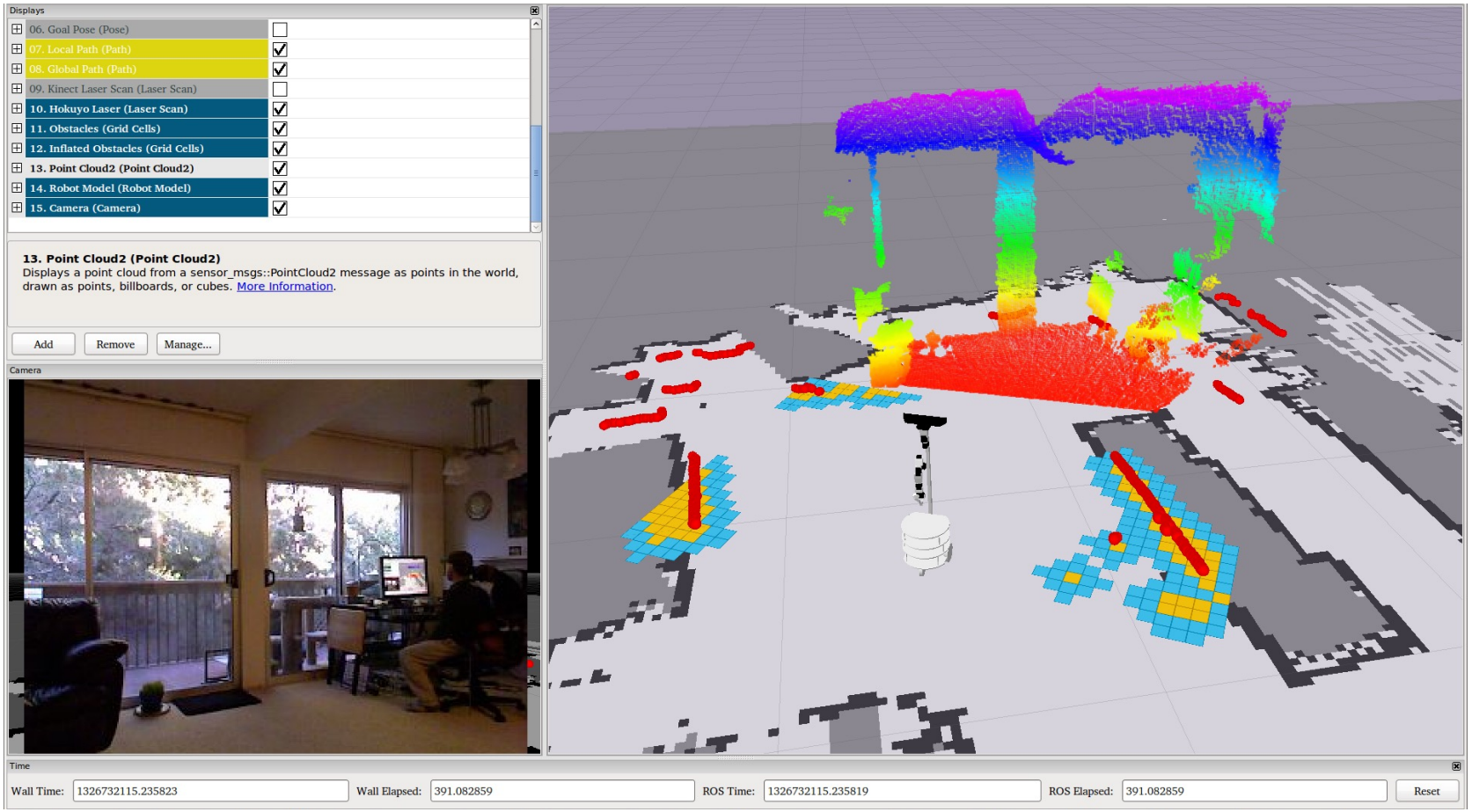


- Rqt_plot



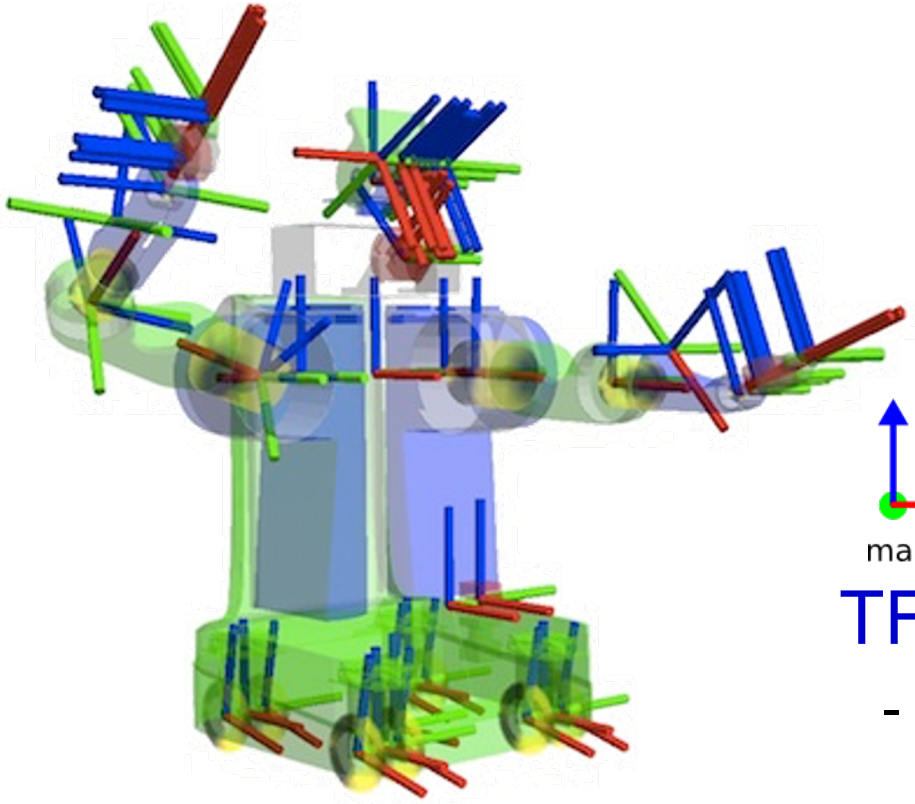
RVIZ

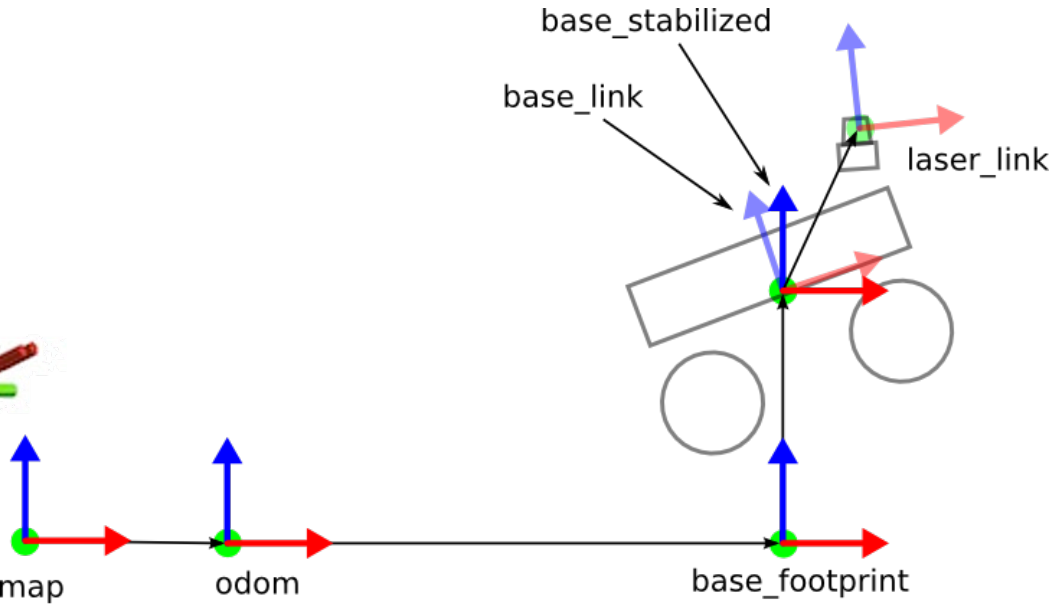
- Visualization of Std messages
- Extensible through plugins



URDF

- Define a robot structure / 3D model in xml
- Used for display
- Used to change frame of reference
- Used in simulators …





TF

- Library to compute coordinate transform

ROS available functionalities

Hardware abstraction / std_msg

- Many sensor abstractions
- Common interface to many robots

Interaction with libraries

- Opencv/PCL bridge
- MoveIt

Navigation

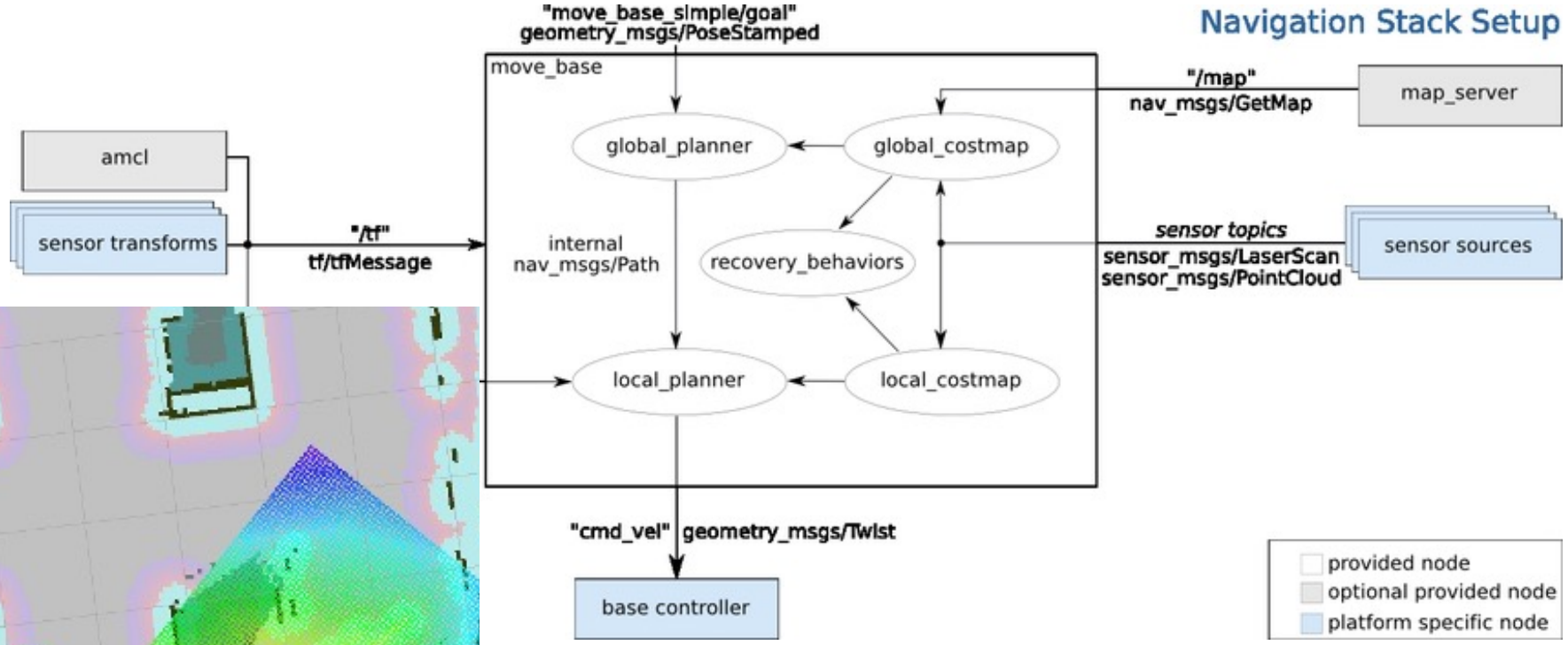
- SLAM (gmapping, hector …)
- Localisation
- Planning / Obstacle avoidance / Recovery behavior

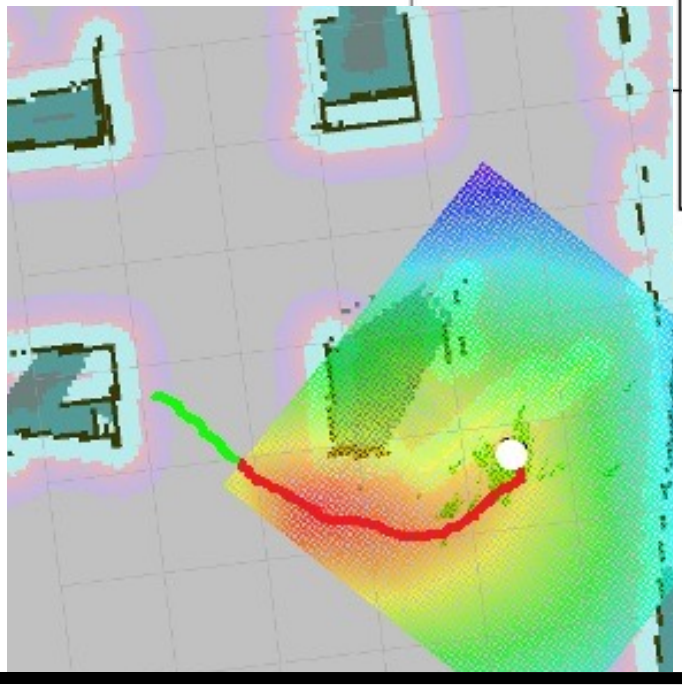
Manipulation

- Object detection / recognition
- Planner

Ex : Navigation Stack

– Used to guide a robot in a known map





Software architectures - Summary

- Software architectures for robotics provide:
  - Ways to distribute computation and exchange data between modules
  - Tools to visualise/process/save/replay… these data
  - Ready to use functionalities (navigation, object picking …)
  - A community of researchers/industrials/hardware

- Robot Operating System ROS is one such architecture, currently the most widely used