

Robotique Mobile

04 - Software architectures - ROS

David Filliat

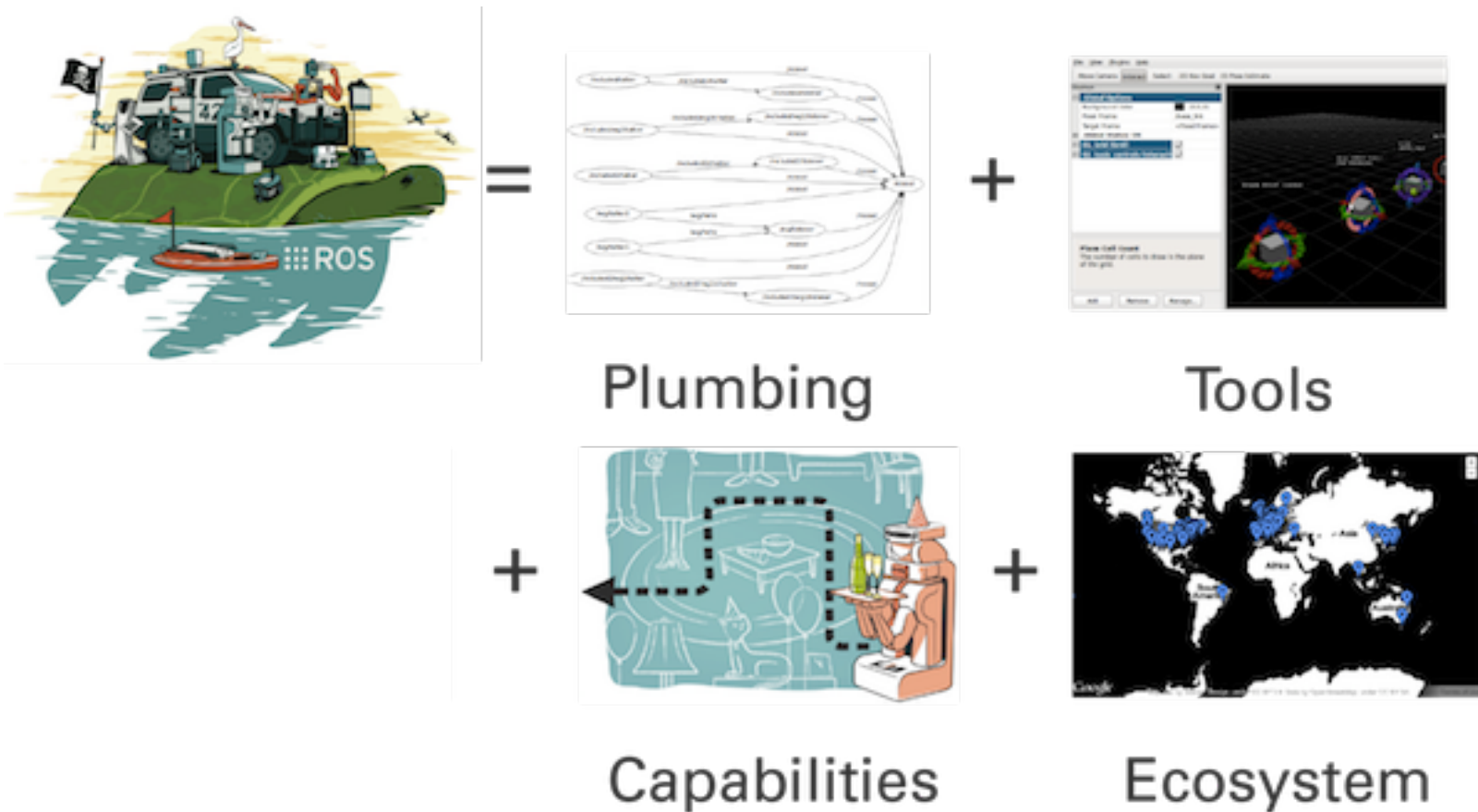
Alexandre Chapoutot

Goran Frehse

prenom.nom@ensta-paris.fr

Robot Operating System

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



- **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>

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

- **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.

Node 1

Node 2

Run a node with

```
> rosruntime package_name node_type
```

See active nodes with

```
> rosnodes list
```

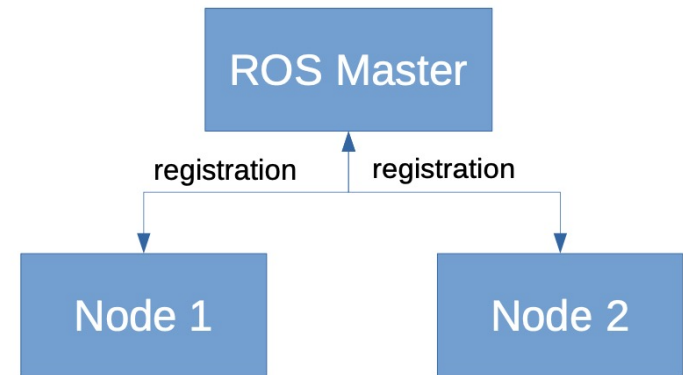
Retrieve information about a node with

```
> rosnodes 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

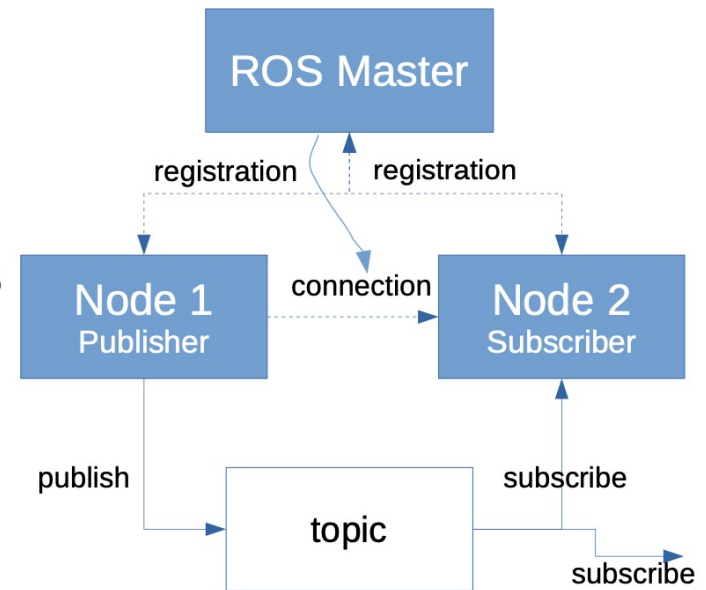
```
> rostopic list
```

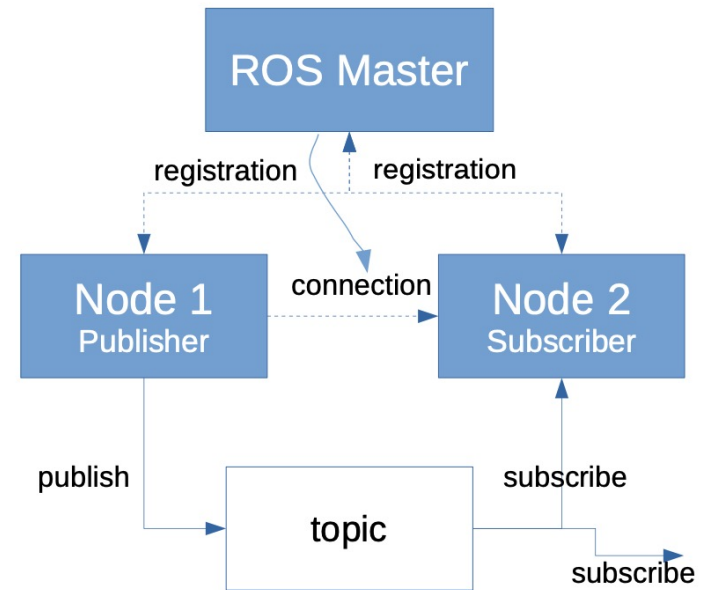
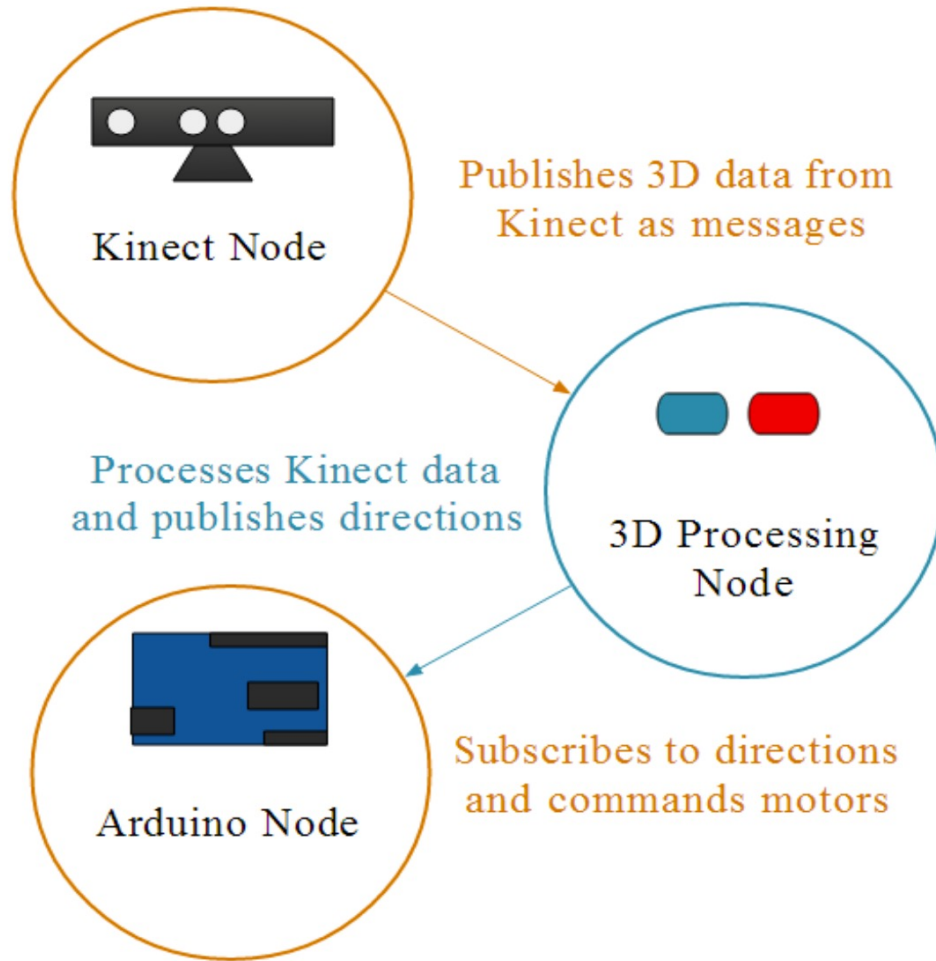
Subscribe and print the contents of a topic with

```
> rostopic echo /topic_name
```

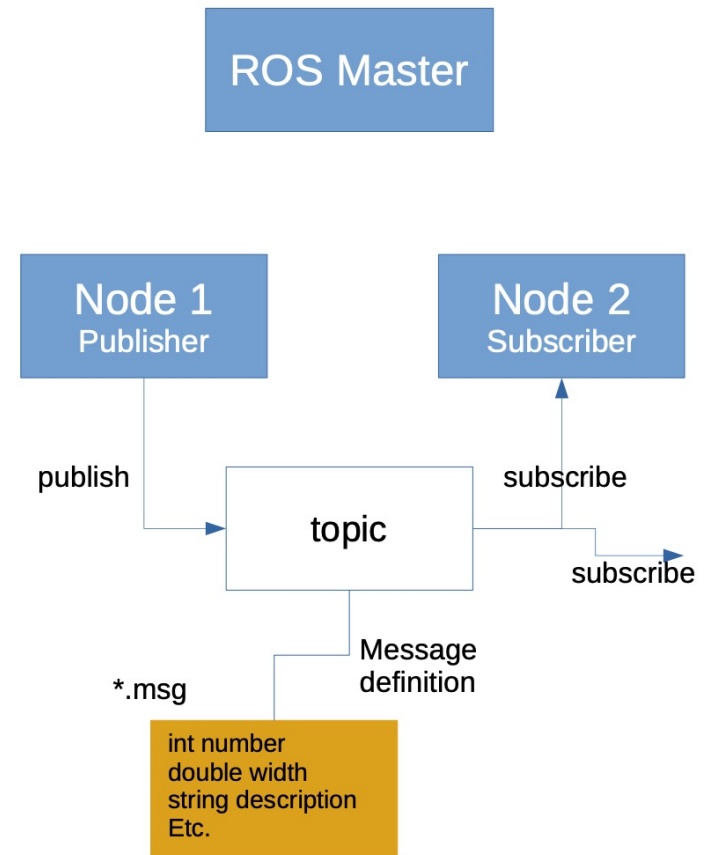
Show information about a topic with

```
> rostopic info /topic_name
```





- 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



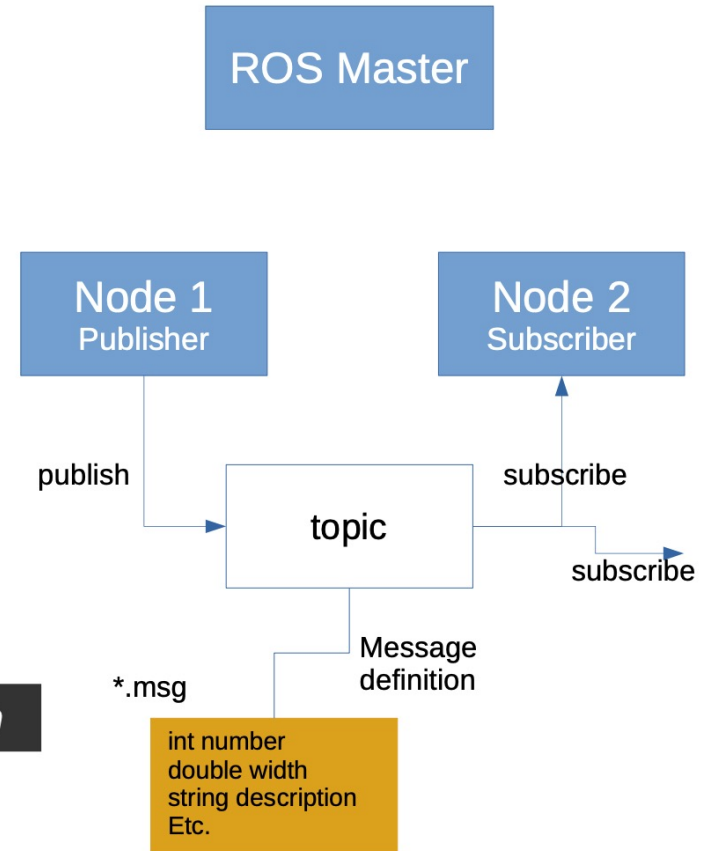
- Defined in **.msg* files
- You can create new ones.
- But to use all the tools, it is better to use the standard messages.

See the type of a topic

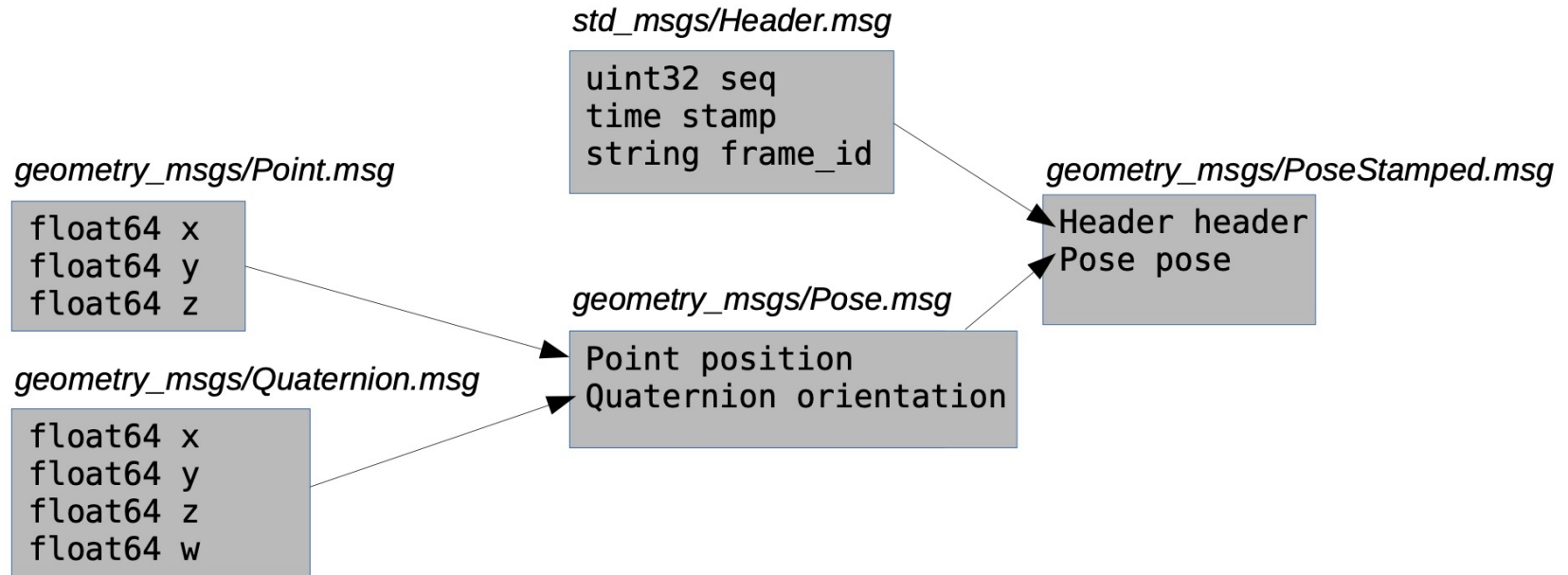
```
> rostopic type /topic_name
```

Publish a message to a topic

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

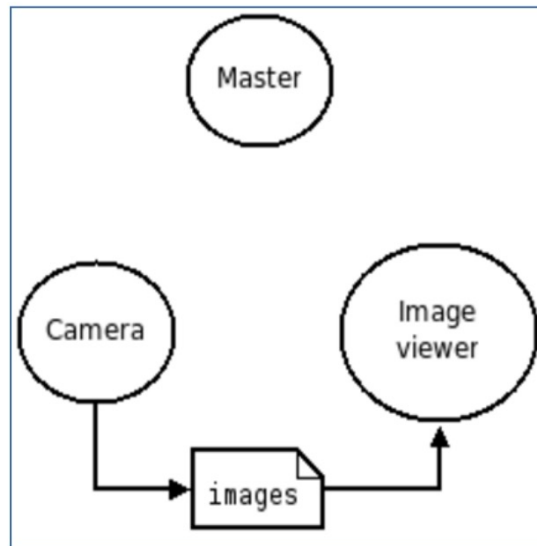
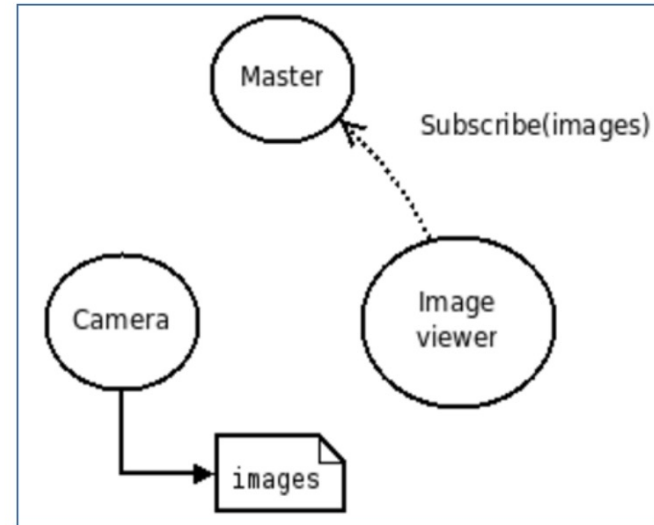
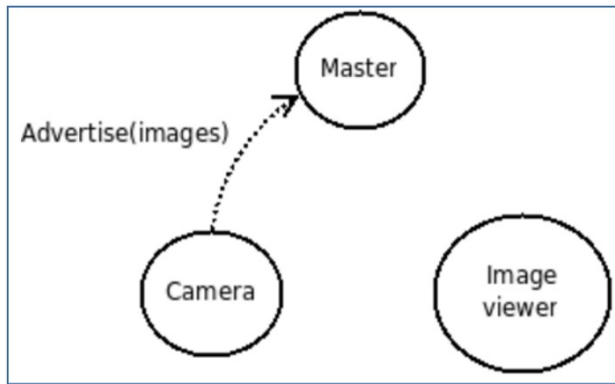


- Pose Stamped Example

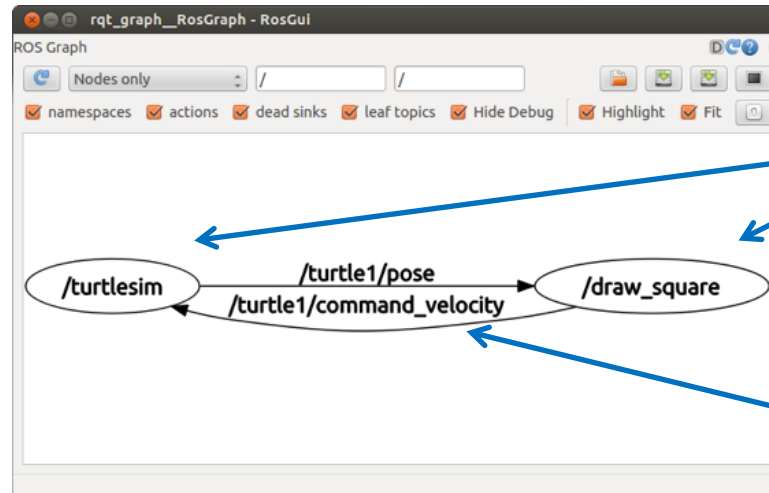
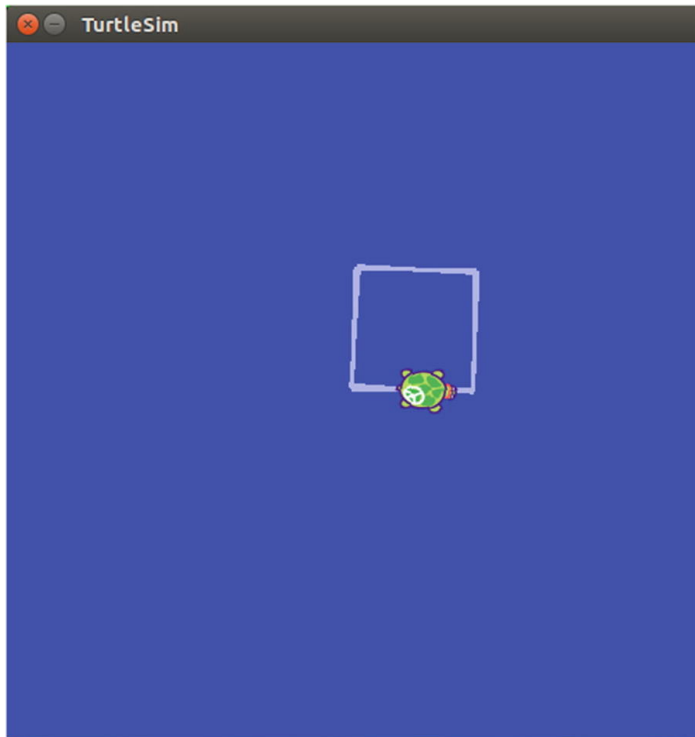


```
> rosmmsg show geometry_msgs/PoseStamped
```

```
> rosmmsg show geometry_msgs/Pose
```



Example application



Nodes

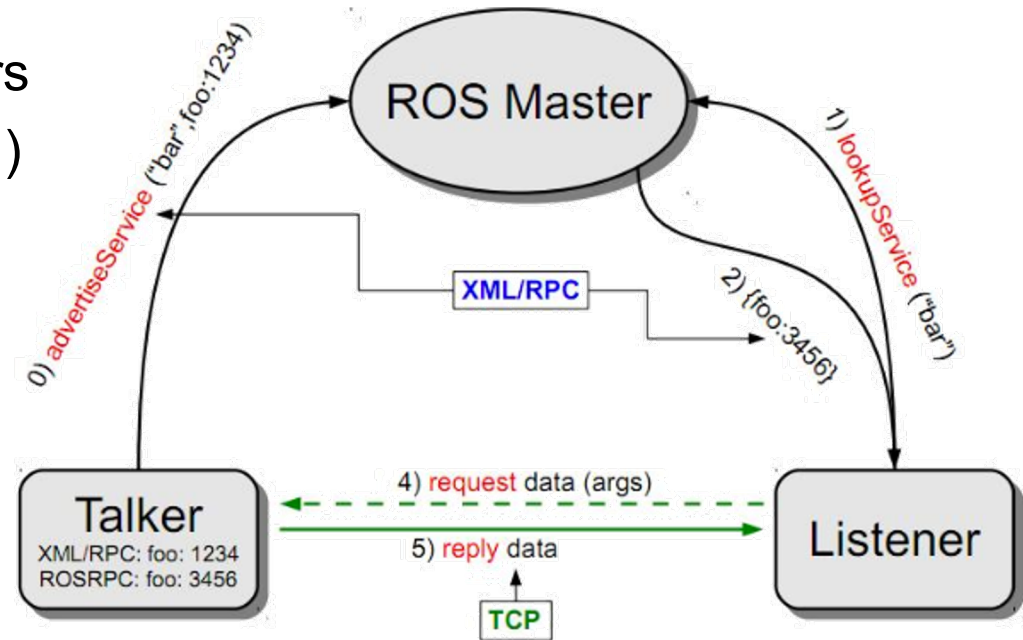
Topics

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

```
1 #!/usr/bin/env python
2 # license removed for brevity
3 import rospy
4 from std_msgs.msg import String
5
6 def talker():
7     pub = rospy.Publisher('chatter', String, queue_size=10)
8     rospy.init_node('talker', anonymous=True)
9     rate = rospy.Rate(10) # 10hz
10    while not rospy.is_shutdown():
11        hello_str = "hello world %s" % rospy.get_time()
12        rospy.loginfo(hello_str)
13        pub.publish(hello_str)
14        rate.sleep()
15
16 if __name__ == '__main__':
17     try:
18         talker()
19     except rospy.ROSInterruptException:
20         pass
```

```
1 #!/usr/bin/env python
2 import rospy
3 from std_msgs.msg import String
4
5 def callback(data):
6     rospy.loginfo(rospy.get_caller_id() + "I heard %s", data.data)
7
8 def listener():
9
10    # In ROS, nodes are uniquely named. If two nodes with the same
11    # name are launched, the previous one is kicked off. The
12    # anonymous=True flag means that rospy will choose a unique
13    # name for our 'listener' node so that multiple listeners can
14    # run simultaneously.
15    rospy.init_node('listener', anonymous=True)
16
17    rospy.Subscriber("chatter", String, callback)
18
19    # spin() simply keeps python from exiting until this node is stopped
20    rospy.spin()
21
22 if __name__ == '__main__':
23     listener()
```

roslaunch

- Run an executable in a package

```
roslaunch 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

roscall

- list / info / kill / clean nodes

```
roscall list
```

```
...
```

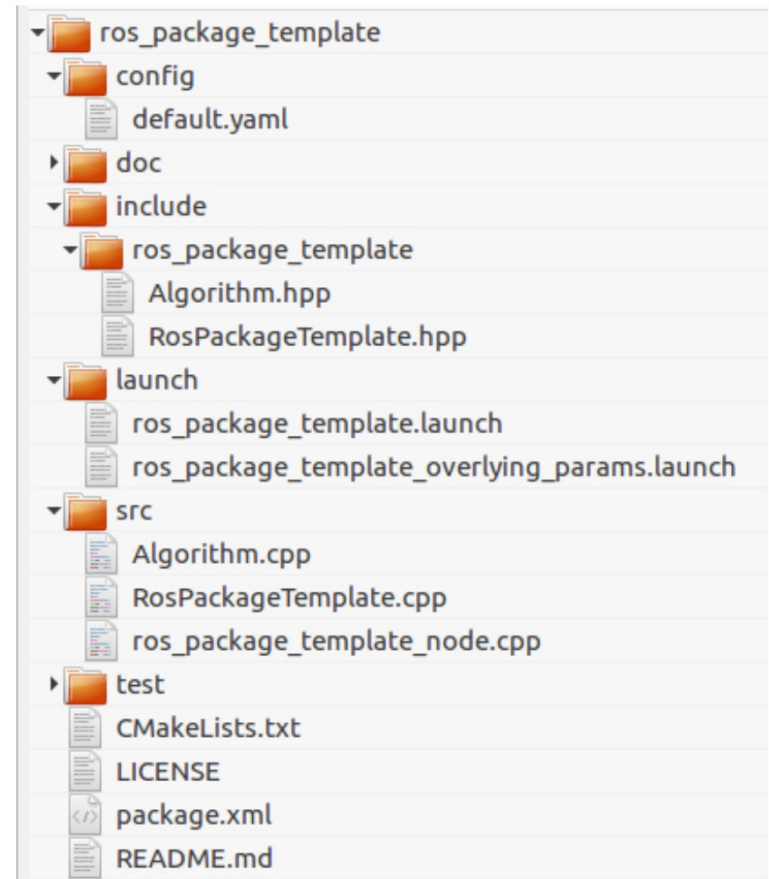
- Catkin is the ROS build system to generate executables, libraries, and interfaces
- A *catkin workspace* is the place in which one or more catkin packages can be built.
- Create and build a *catkin workspace*:

```
> 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*

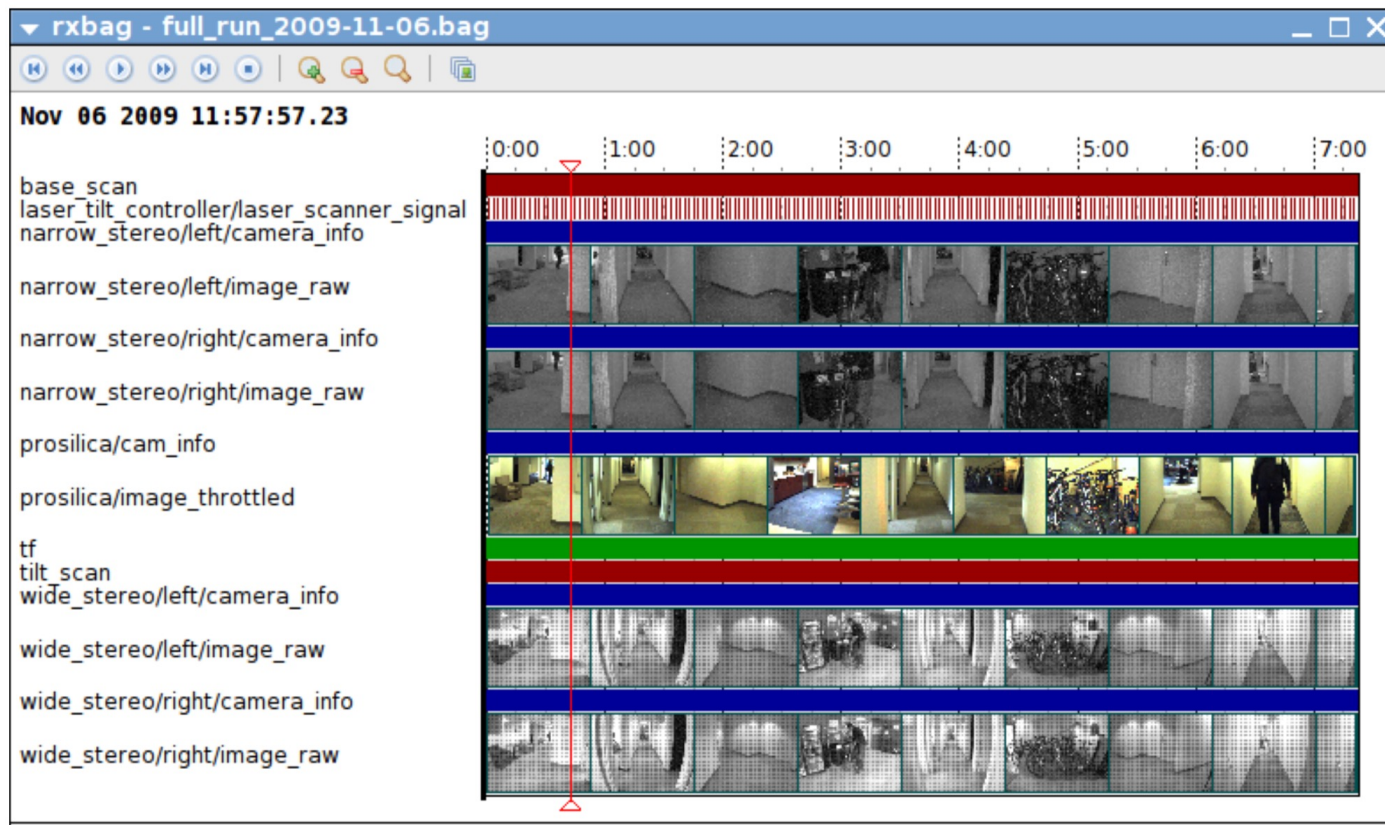
- In your catkin workspace, you have 3 folders *build*, *devel* and *src*
 - **src** : *Work here*
 - The source space contains the source code. This is where you can clone, create, and edit source code for the packages you want to build.
 - **build** : *Don't touch*
 - *The build space is where CMake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



rosvbag

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



roscpp

- Get/set params in the roscore

```
roscpp set
```

```
roscpp get
```

- Accessible through roscpp/rospy...

```
global_name = ros.get_param("/global_name")
```

```
ros.set_param('a_string', 'baz')
```

dynamic_reconfigure

- package to dynamically change node params

The image shows a dynamic_reconfigure GUI. On the left is a tree view of parameters for the node `/narrow_stereo_textured`. The tree is expanded to show the `left` node, which contains `image_color`, `image_mono`, `image_raw`, `image_rect`, `image_rect_color`, `narrow_stereo_textured_proc`, `narrow_stereo_textured_proc_debayer_left`, `narrow_stereo_textured_proc_debayer_right`, and `narrow_stereo_textured_proc_rectify_color_left`. The `image_color` node is expanded to show `compressed`, `compressedDepth`, and `theora` sub-nodes. The `compressed` node contains `format`, `jpeg_quality`, and `png_level`. The `compressedDepth` node contains `depth_max`, `depth_quantization`, and `png_level`. The `theora` node contains `optimize_for`, `target_bitrate`, `quality`, and `keyframe_frequency`.

On the right, two panels show the detailed configuration for the `compressedDepth` and `theora` parameters. The `compressedDepth` panel shows sliders for `depth_max` (1.0 to 100.0, value 10.0), `depth_quantization` (1.0 to 150.0, value 100.0), and `png_level` (1 to 9, value 9). The `theora` panel shows a dropdown for `optimize_for` (Quality (1)), a slider for `target_bitrate` (0 to 99200000, value 00000), a slider for `quality` (0 to 63, value 31), and a slider for `keyframe_frequency` (1 to 64, value 64).

roscconsole

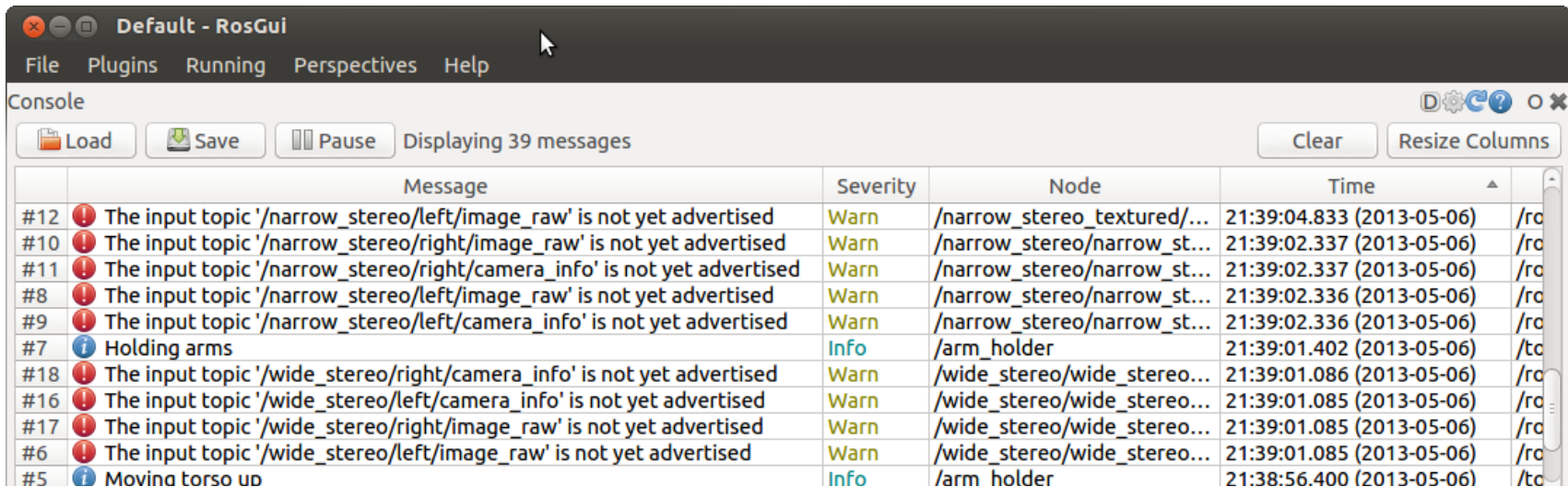
- Logging capability in roscpp (also in rospy)

```
#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



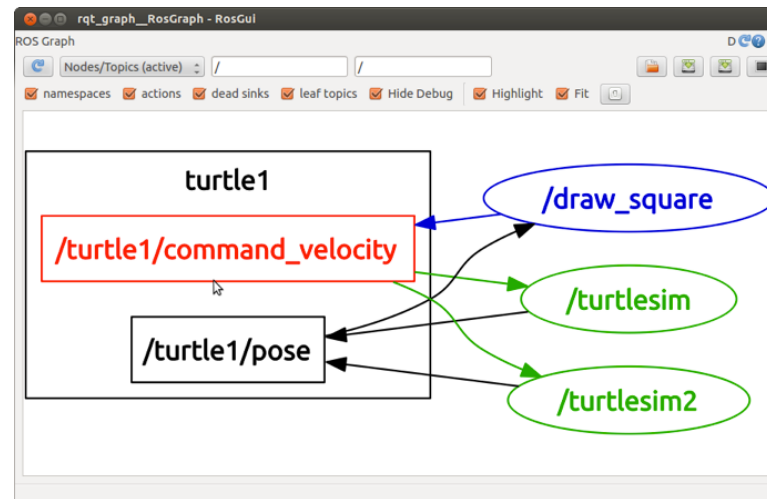
The screenshot shows the RosGui interface with a console window titled "Default - RosGui". The console displays 39 messages. The messages are organized into columns: Message, Severity, Node, and Time. The messages include warnings about unadvertised topics and an info message about holding arms.

#	Message	Severity	Node	Time
#12	The input topic '/narrow_stereo/left/image_raw' is not yet advertised	Warn	/narrow_stereo_textured/...	21:39:04.833 (2013-05-06)
#10	The input topic '/narrow_stereo/right/image_raw' is not yet advertised	Warn	/narrow_stereo/narrow_st...	21:39:02.337 (2013-05-06)
#11	The input topic '/narrow_stereo/right/camera_info' is not yet advertised	Warn	/narrow_stereo/narrow_st...	21:39:02.337 (2013-05-06)
#8	The input topic '/narrow_stereo/left/image_raw' is not yet advertised	Warn	/narrow_stereo/narrow_st...	21:39:02.336 (2013-05-06)
#9	The input topic '/narrow_stereo/left/camera_info' is not yet advertised	Warn	/narrow_stereo/narrow_st...	21:39:02.336 (2013-05-06)
#7	Holding arms	Info	/arm_holder	21:39:01.402 (2013-05-06)
#18	The input topic '/wide_stereo/right/camera_info' is not yet advertised	Warn	/wide_stereo/wide_stereo...	21:39:01.086 (2013-05-06)
#16	The input topic '/wide_stereo/left/camera_info' is not yet advertised	Warn	/wide_stereo/wide_stereo...	21:39:01.085 (2013-05-06)
#17	The input topic '/wide_stereo/right/image_raw' is not yet advertised	Warn	/wide_stereo/wide_stereo...	21:39:01.085 (2013-05-06)
#6	The input topic '/wide_stereo/left/image_raw' is not yet advertised	Warn	/wide_stereo/wide_stereo...	21:39:01.085 (2013-05-06)
#5	Movina torso up	Info	/arm_holder	21:38:56.400 (2013-05-06)

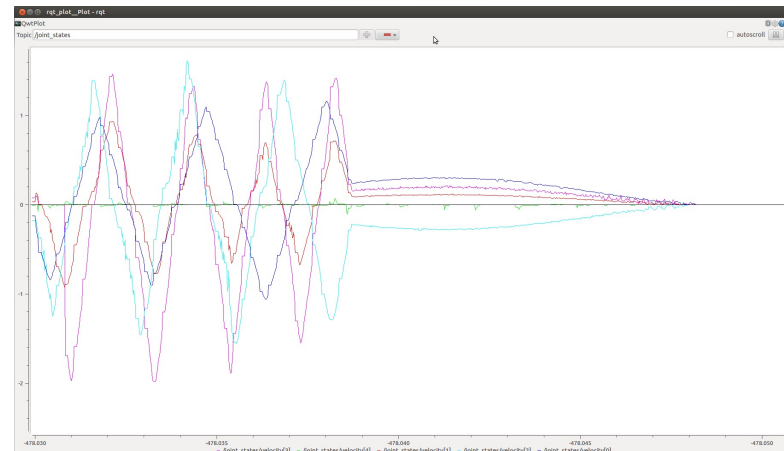
RQT

– Graphical User Interface with pugins

– Rqt_graph



– Rqt_plot



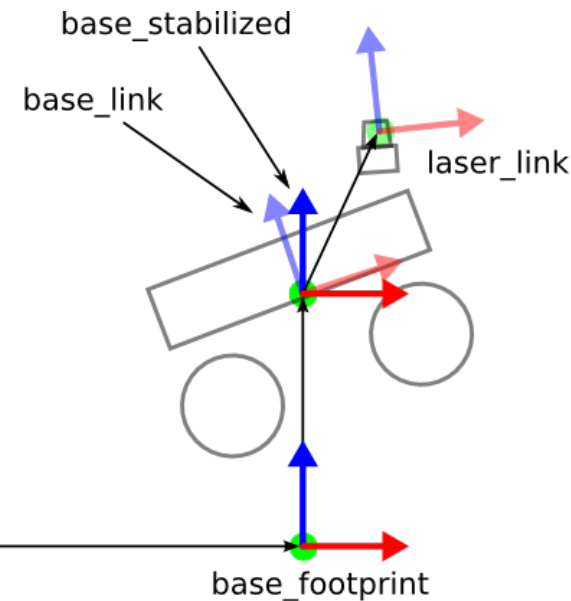
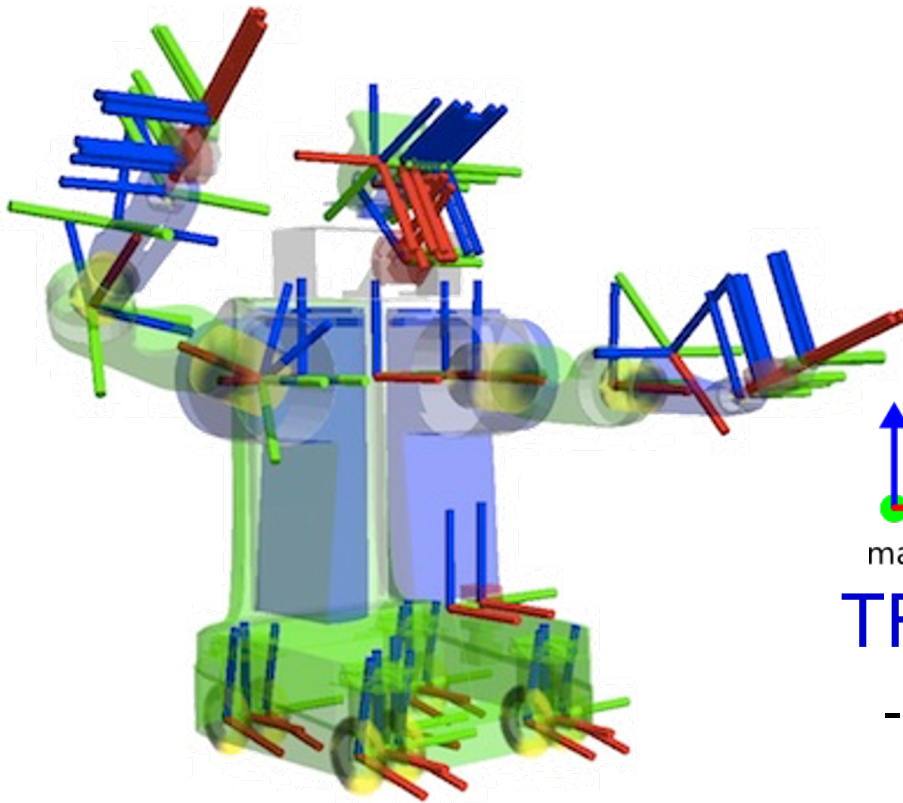
RVIZ

- Visualization of Std messages
- Extensible through plugins

The screenshot displays the RVIZ (Robot Visualization) interface. On the left, a 'Displays' panel lists 15 items, with '07. Local Path (Path)', '08. Global Path (Path)', '10. Hokuyo Laser (Laser Scan)', '11. Obstacles (Grid Cells)', '12. Inflated Obstacles (Grid Cells)', '13. Point Cloud2 (Point Cloud2)', '14. Robot Model (Robot Model)', and '15. Camera (Camera)' checked. Below this, a description for '13. Point Cloud2 (Point Cloud2)' is visible. A 'Camera' panel shows a live video feed of a person working at a desk in a room with large windows. The main 3D view shows a robot model in a simulated environment with a grid floor, obstacles, and a point cloud visualization of a sensor's field of view. The bottom status bar shows 'Wall Time: 1326732115.235823', 'Wall Elapsed: 391.082859', 'ROS Time: 1326732115.235819', 'ROS Elapsed: 391.082859', and a 'Reset' button.

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

Hardware abstraction / std_msg

- Many sensor abstractions
- Common interface to many robots

Interaction with libraries

- Opencv/PCL bridge
- MoveIt

Navigation

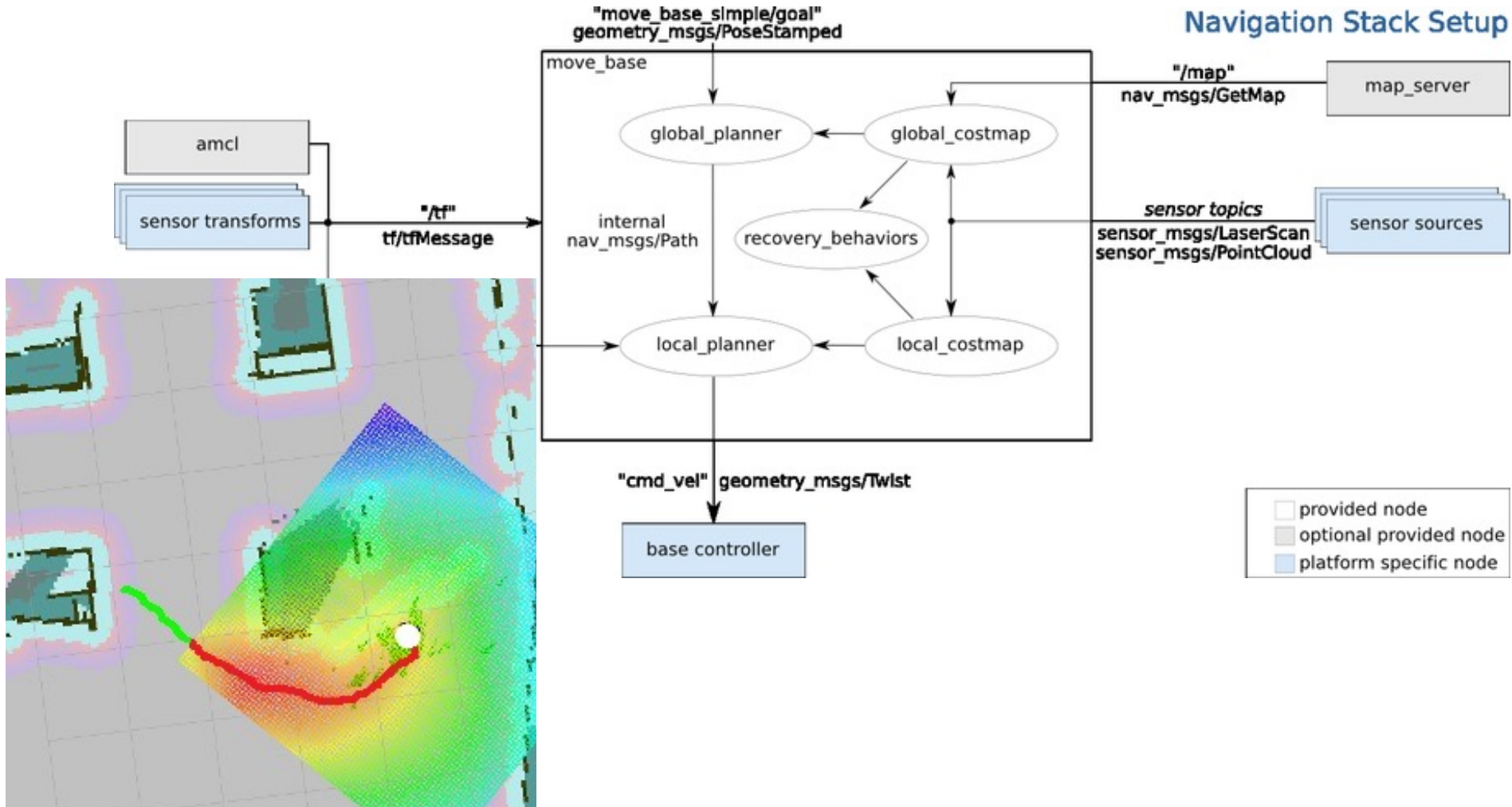
- SLAM (gmapping, hector ...)
- Localisation
- Planning / Obstacle avoidance / Recovery behavior

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