



This presentation is released under the terms of the  
**Creative Commons Attribution-Share Alike** license.

You are free to reuse it and modify it as much as you want as long as:

- (1) you mention Tony Belpaeme and Séverin Lemaignan as being the original authors,
- (2) you re-share your presentation under the same terms.

You can download the sources of this presentation here:  
**[github.com/severin-lemaignan/module-mobile-and-humanoid-robots](https://github.com/severin-lemaignan/module-mobile-and-humanoid-robots)**

# **ROBOTICS WITH PLYMOUTH UNIVERSITY**

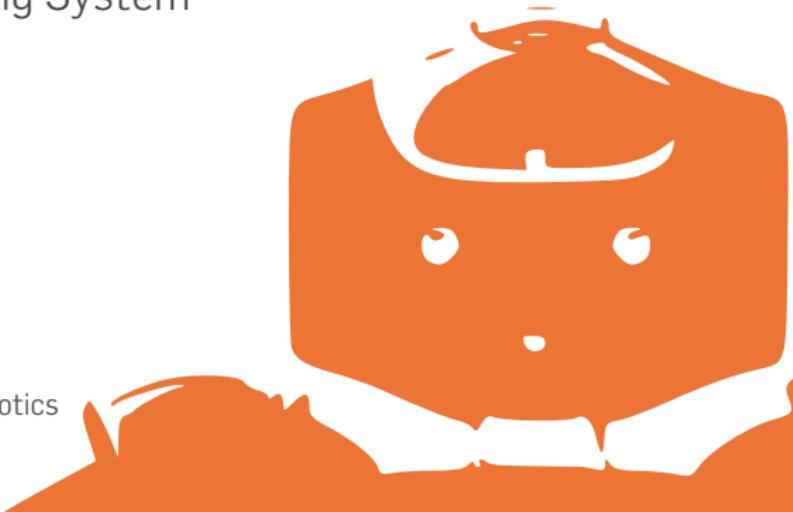
ROC0318

Mobile and Humanoid Robots

ROS, the Robot Operating System

Séverin Lemaignan

Centre for Neural Systems and Robotics  
**Plymouth University**



## COURSEWORK ASSESSMENT

- o Coursework: 30% of final mark; sensor project: 60% of coursework mark
- o Complete report submitted **Thursday 17:00, 11th January 2018**
- o One submission per group, single PDF file, max 10 pages (without images/code)

## COURSEWORK ASSESSMENT

- Coursework: 30% of final mark; sensor project: 60% of coursework mark
- Complete report submitted **Thursday 17:00, 11th January 2018**
- One submission per group, single PDF file, max 10 pages (without images/code)

Check the coursework description on the DLE for the details of the marking scheme.

ROS IS NOT AN OPERATING SYSTEM

## INSTEAD, ROS IS...

- A fairly simple peer-to-peer message passing system designed with robotics in mind

## INSTEAD, ROS IS...

- A fairly simple peer-to-peer message passing system designed with robotics in mind
- An API to this system (in several languages – C++ and Python are 1st tier)

## INSTEAD, ROS IS...

- A fairly simple peer-to-peer message passing system designed with robotics in mind
- An API to this system (in several languages – C++ and Python are 1st tier)
- A set of standard message types that facilitate interoperability between modules

## INSTEAD, ROS IS...

- o A fairly simple peer-to-peer message passing system designed with robotics in mind
- o An API to this system (in several languages – C++ and Python are 1st tier)
- o A set of standard message types that facilitate interoperability between modules
- o **A middleware?**

## INSTEAD, ROS IS...

- o A fairly simple peer-to-peer message passing system designed with robotics in mind
- o An API to this system (in several languages – C++ and Python are 1st tier)
- o A set of standard message types that facilitate interoperability between modules
- o A set of conventions to write and package robotic softwares

## INSTEAD, ROS IS...

- o A fairly simple peer-to-peer message passing system designed with robotics in mind
- o An API to this system (in several languages – C++ and Python are 1st tier)
- o A set of standard message types that facilitate interoperability between modules
  
- o A set of conventions to write and package robotic softwares
- o Deep integration of a few key open-source libraries (OpenCV, PCL, tf)

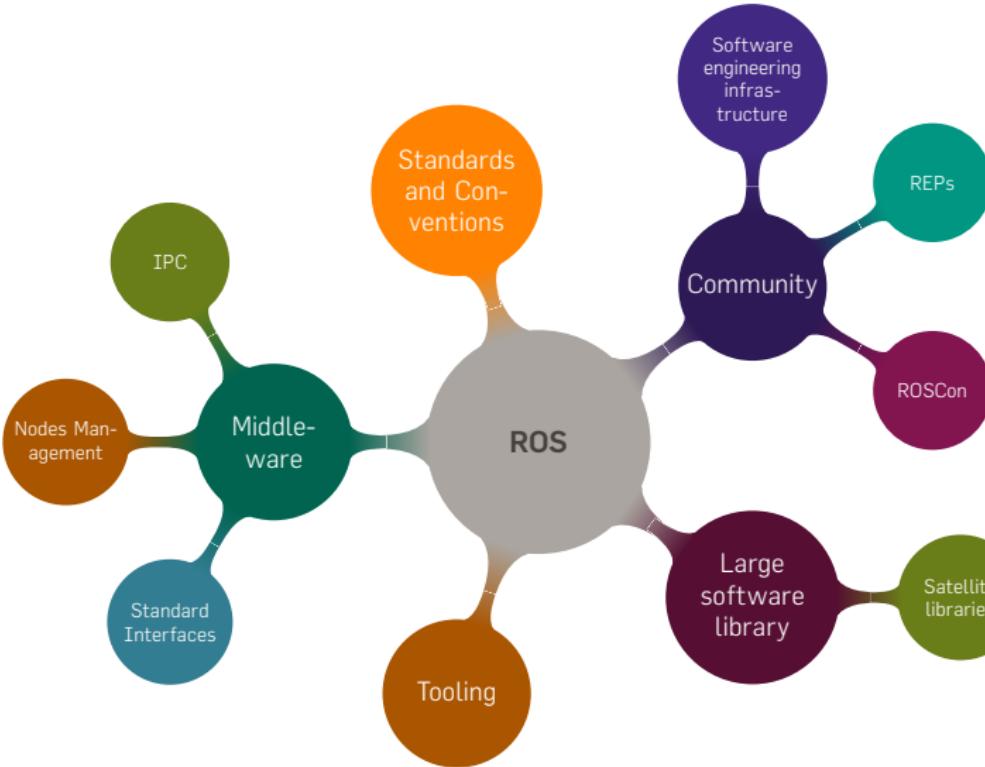
## INSTEAD, ROS IS...

- o A fairly simple peer-to-peer message passing system designed with robotics in mind
- o An API to this system (in several languages – C++ and Python are 1st tier)
- o A set of standard message types that facilitate interoperability between modules
  
- o A set of conventions to write and package robotic softwares
- o Deep integration of a few key open-source libraries (OpenCV, PCL, tf)
- o A set of tools to run and monitor the nodes

## INSTEAD, ROS IS...

- o A fairly simple peer-to-peer message passing system designed with robotics in mind
- o An API to this system (in several languages – C++ and Python are 1st tier)
- o A set of standard message types that facilitate interoperability between modules
  
- o A set of conventions to write and package robotic softwares
- o Deep integration of a few key open-source libraries (OpenCV, PCL, tf)
- o A set of tools to run and monitor the nodes
- o Engagement of a large academic community, leading to a library of thousands of nodes

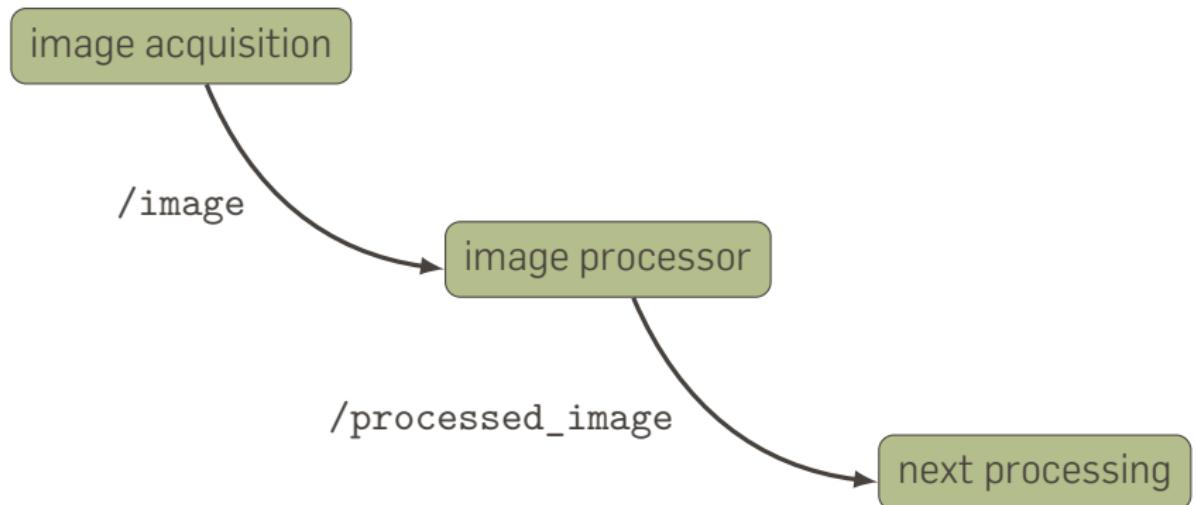
# ROS ECOSYSTEM



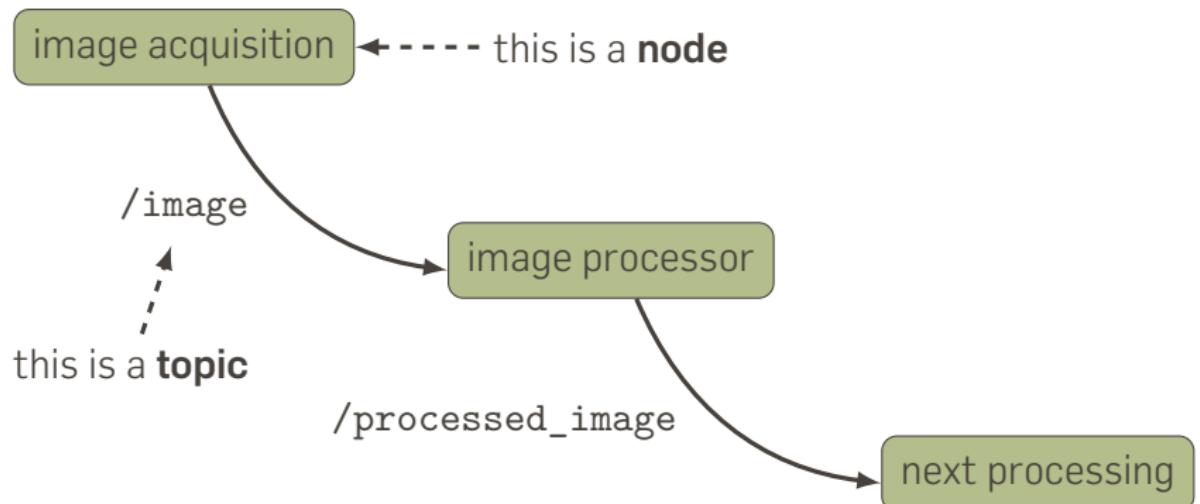
[we will revisit these slides at the end of the lecture]

# A FIRST EXAMPLE

# A SIMPLE IMAGE PROCESSING PIPELINE



# A SIMPLE IMAGE PROCESSING PIPELINE



```
1 import sys, cv2, rospy
2 from sensor_msgs.msg import Image
3 from cv_bridge import CvBridge
4
5 def on_image(image):
6     cv_image = bridge.imgmsg_to_cv2(image, "bgr8")
7     rows, cols, channels = cv_image.shape
8     cv2.circle(cv_image, (cols/2, rows/2), 50, (0,0,255), -1)
9     image_pub.publish(bridge.cv2_to_imgmsg(cv_image, "bgr8"))
10
11 rospy.init_node('image_processor')
12 bridge = CvBridge()
13 image_sub = rospy.Subscriber("image",Image, on_image)
14 image_pub = rospy.Publisher("processed_image",Image)
15
16 while not rospy.is_shutdown():
17     rospy.spin()
```

## HOW TO USE THIS CODE?

First, we need to write data onto the `/image` topic, for instance from a webcam:

---

```
> rosrun usb_cam usb_cam_node
```

---

## HOW TO USE THIS CODE?

First, we need to write data onto the `/image` topic, for instance from a webcam:

---

```
> rosrun usb_cam usb_cam_node
```

---

Then, we run our code:

---

```
> python image_processor.py
```

---

## HOW TO USE THIS CODE?

First, we need to write data onto the `/image` topic, for instance from a webcam:

---

```
> rosrun usb_cam usb_cam_node
```

---

Then, we run our code:

---

```
> python image_processor.py
```

---

Finally, we run a 3rd node to display the image:

---

```
> rosrun image_view image_view image:=/processed_image
```

---



# THE KEY ROS CONCEPTS

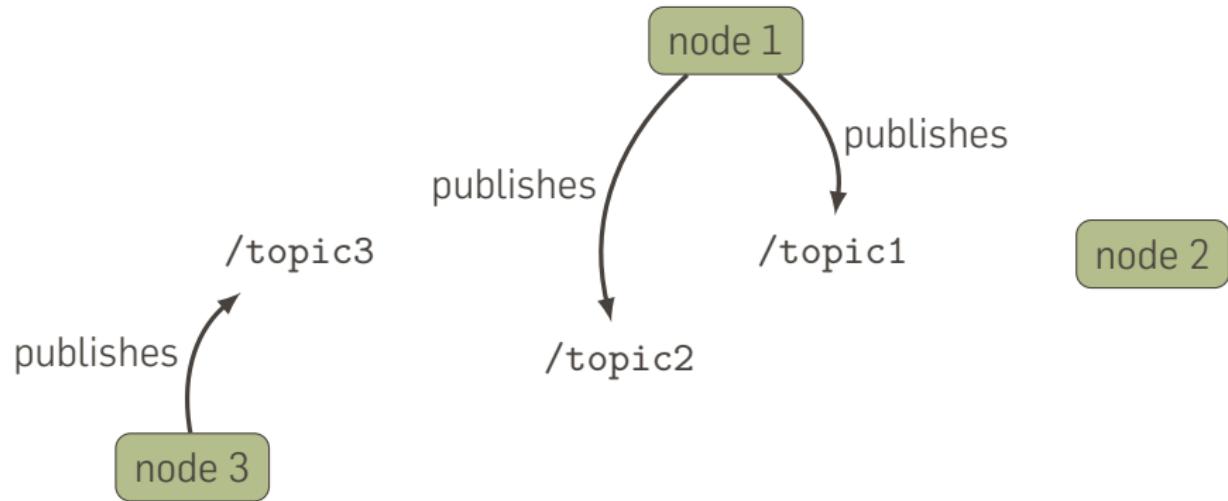
# TALKING NODES

node 1

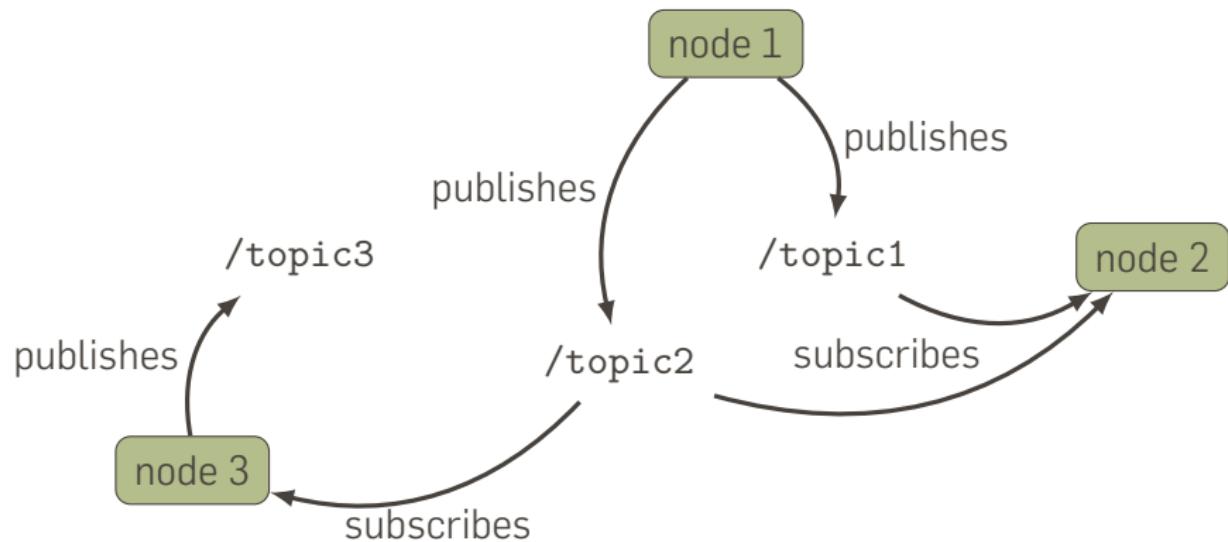
node 2

node 3

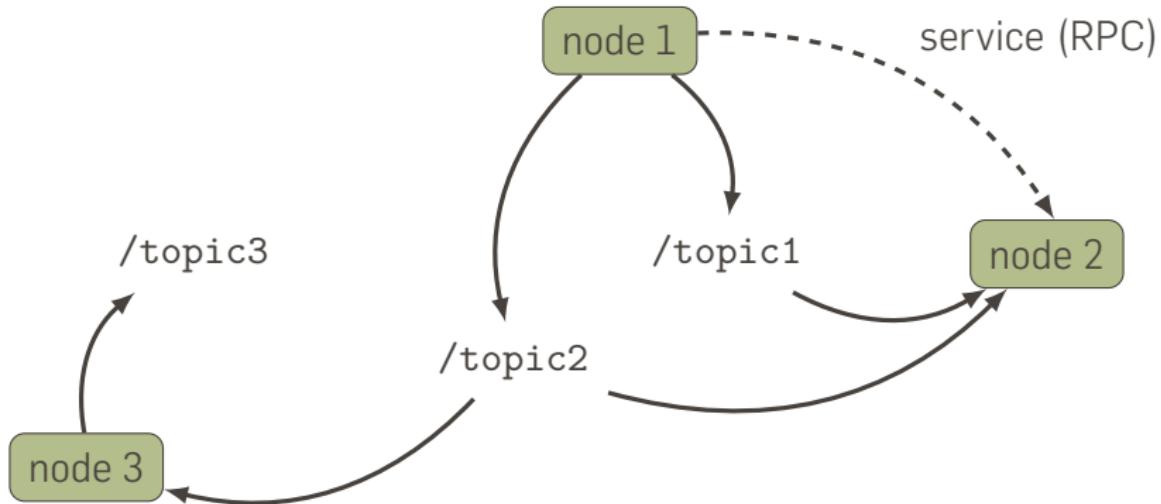
# TALKING NODES



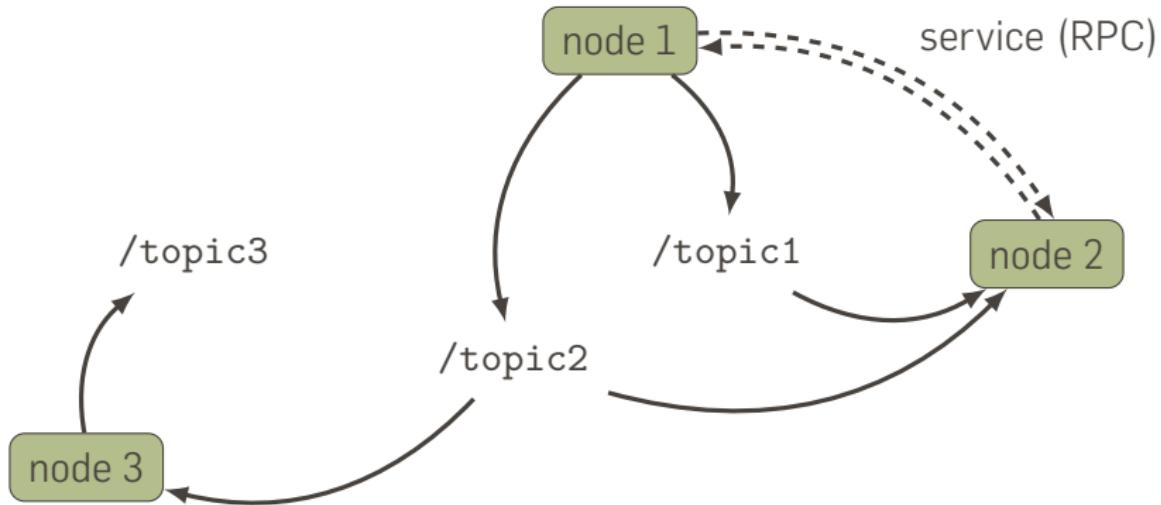
# TALKING NODES



# TALKING NODES

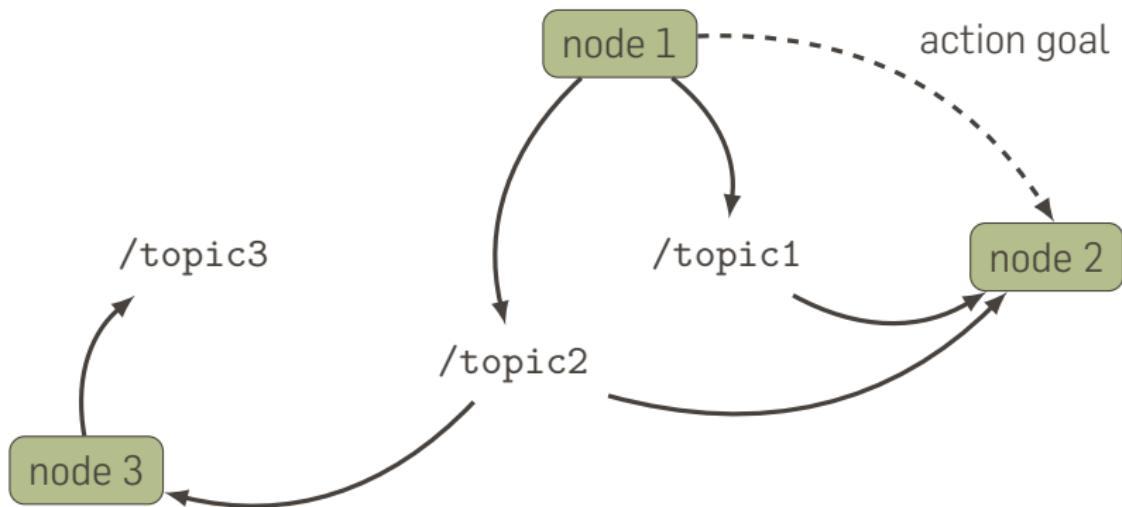


# TALKING NODES

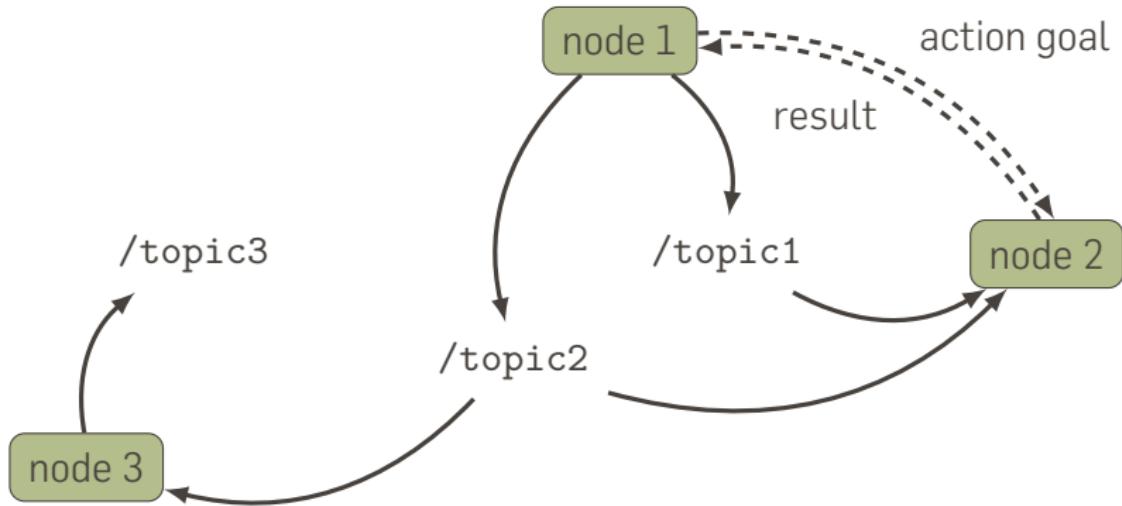


Services: **synchronous**: call is blocking, only suitable when very short processing (e.g. setting a parameter)

# TALKING NODES

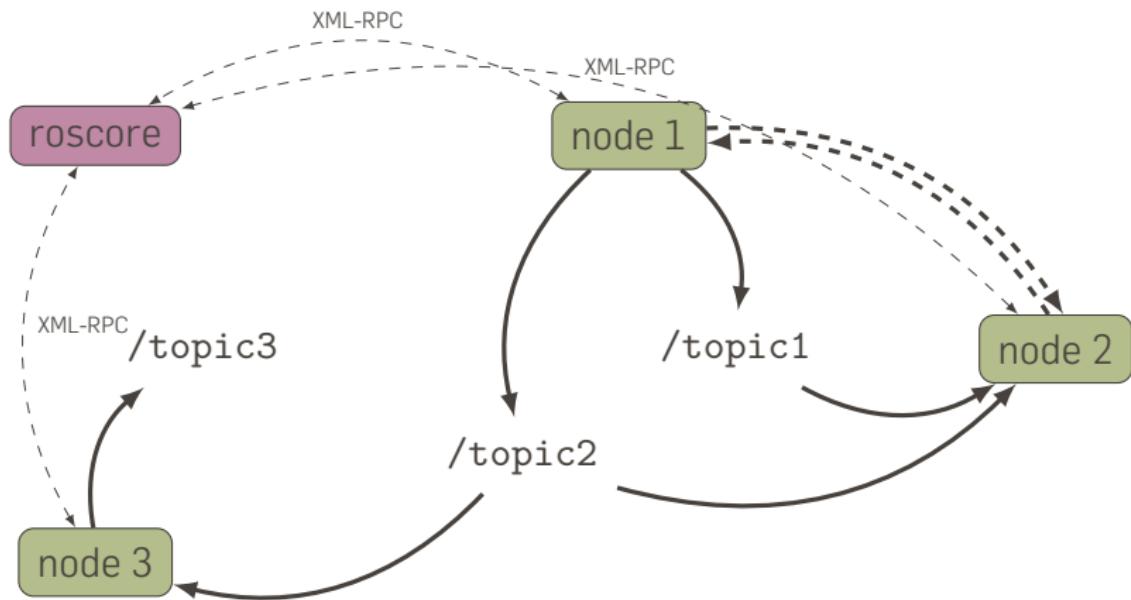


# TALKING NODES



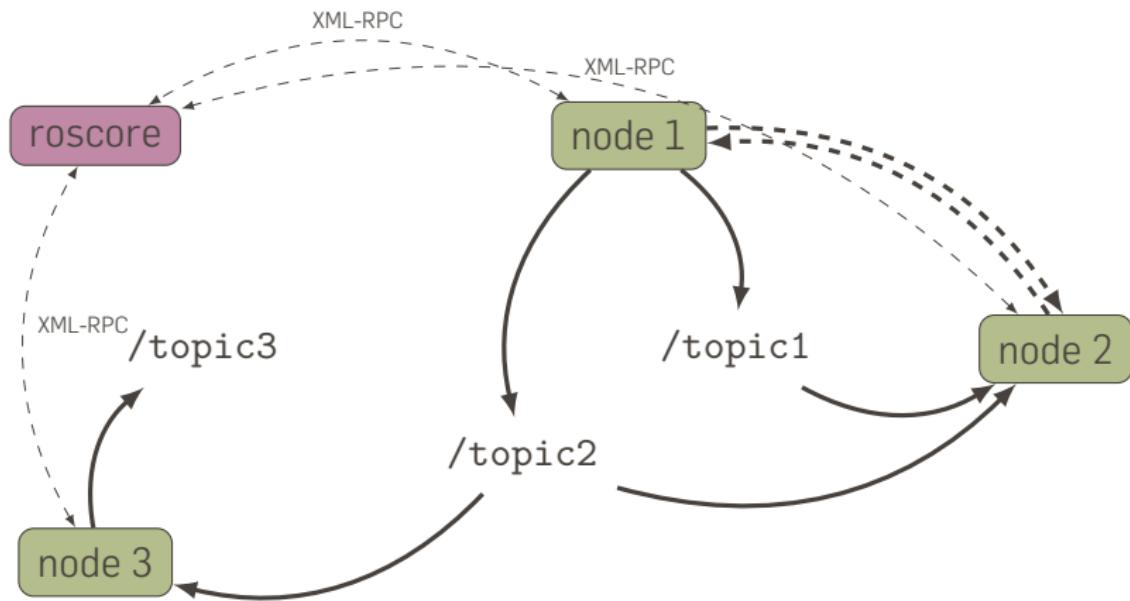
Actions: **asynchronous**: call is non-blocking, suitable for long processes (e.g. motion planning)

# TALKING NODES



The `roscore` daemon act as yellow pages for the nodes to discover each others.

# TALKING NODES



When nodes are distributed on different machines:  
`ROS_MASTER_URI=http://<host>:<port>` to point to **roscore**

## MESSAGES

**Topics** are TCP ports on which data is exchanged.

The data is **serialized** using a format specific to each type of data:  
ROS defines its **data interface** with *messages*.

# MESSAGES

**Topics** are TCP ports on which data is exchanged.

The data is **serialized** using a format specific to each type of data:  
ROS defines its **data interface** with *messages*.

---

```
> rosmsg show geometry_msgs/Pose
geometry_msgs/Point position
    float64 x
    float64 y
    float64 z
geometry_msgs/Quaternion orientation
    float64 x
    float64 y
    float64 z
    float64 w
```

---

Source: [geometry\\_msgs::Pose definition](#)

## MESSAGE EXAMPLE: JOINT STATE

---

```
> rosmsg show sensor_msgs/JointState
std_msgs/Header header
    uint32 seq
    time stamp
    string frame_id
string[] name
float64[] position
float64[] velocity
float64[] effort
```

---

Source: [sensor\\_msgs::JointState definition](#)

## MESSAGE EXAMPLE: IMAGE

---

```
> rosmsg show sensor_msgs/Image
std_msgs/Header header
    uint32 seq
    time stamp
    string frame_id
uint32 height
uint32 width
string encoding
uint8 is_bigendian
uint32 step
uint8[] data
```

---

Source: *sensor\_msgs::Image definition*

## MESSAGES CONTENT

---

```
> rostopic echo /camera/image_raw
header:
  seq: 56
  stamp:
    secs: 1449243166
    nsecs: 415330019
  frame_id: /camera_frame
height: 720
width: 1280
encoding: rgb8
is_bigendian: 0
step: 3840
data: [32, 57, 51, 36, 61, 55, 41, 63, 60, ...]
```

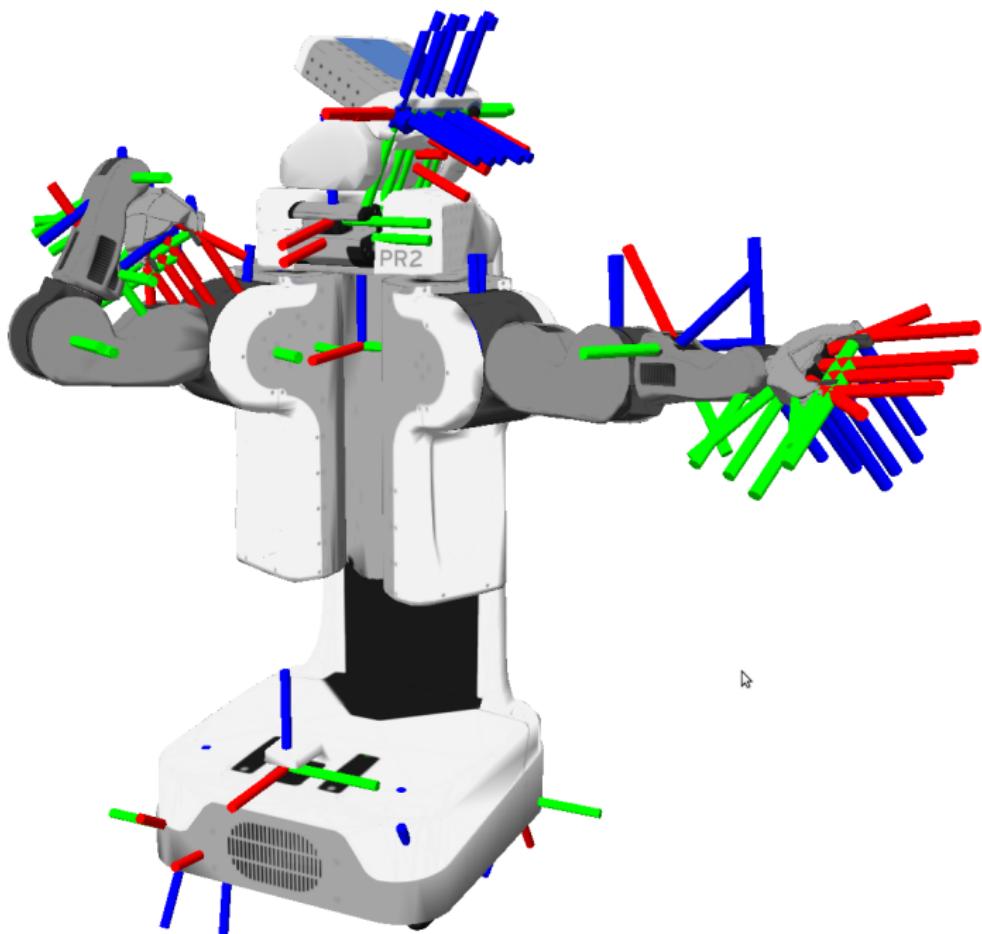
---

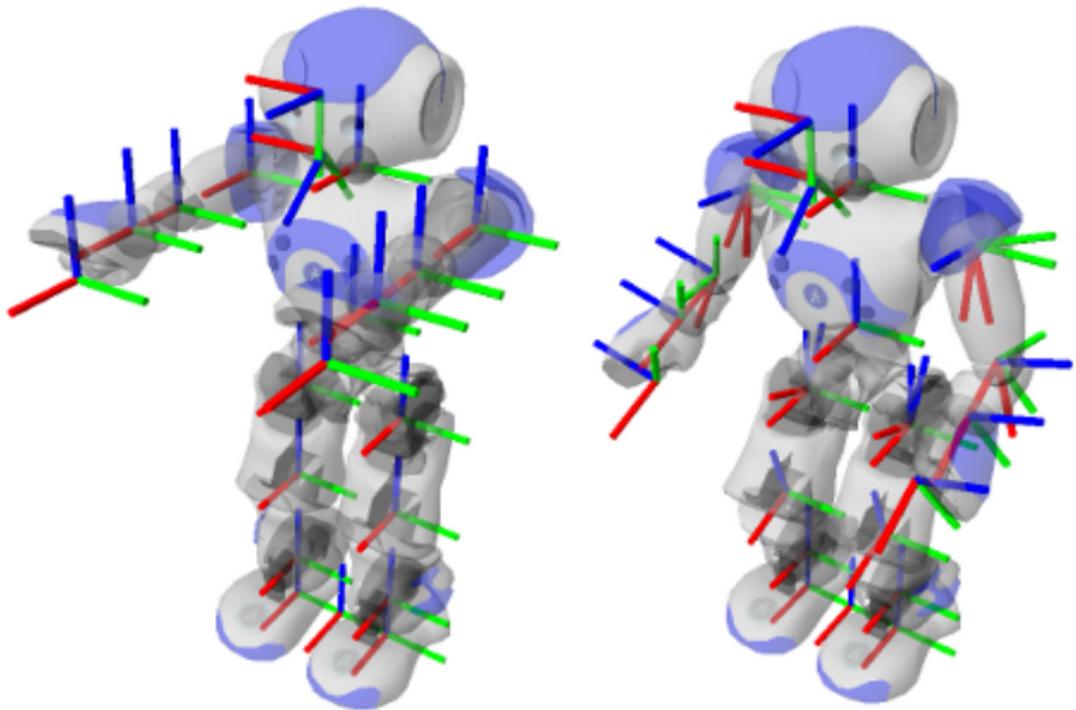
What are these *frames*?

What are these *frames*?

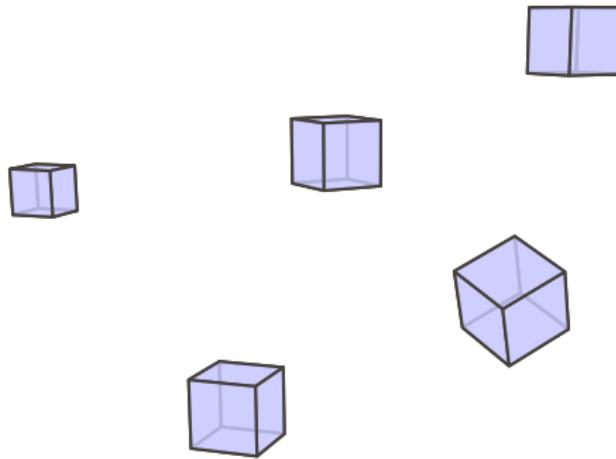
A **frame** is a labelled orthogonal basis with a convenient (6D) origin.

Each part of the robot has usually its own frame, sensors have their frames, objects in the environment have their frames, etc.

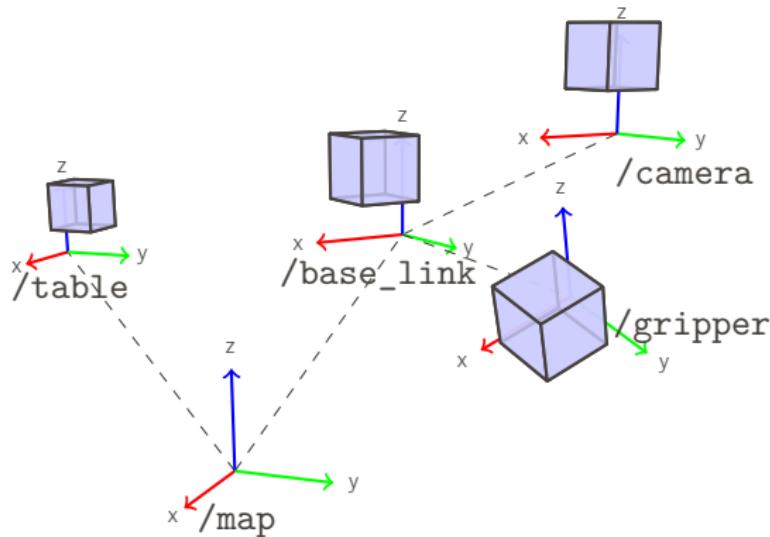




# FRAMES

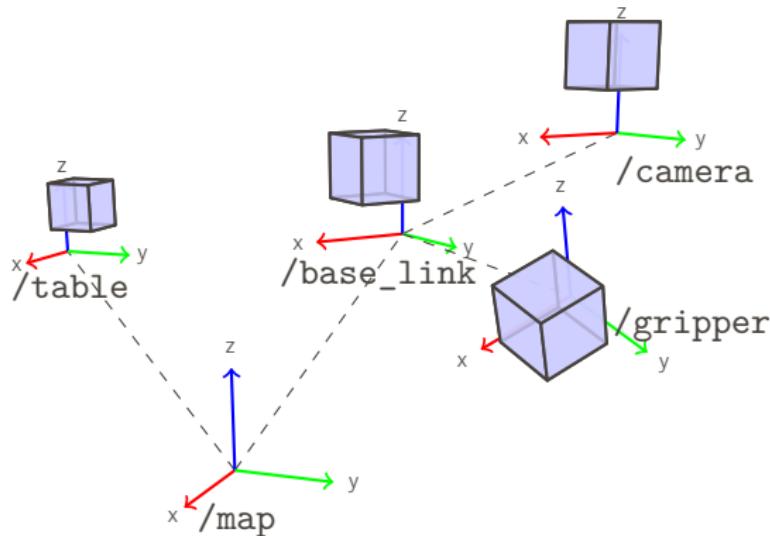


# FRAMES



The **TF** library is responsible for maintaining the full transformation tree, and calculating the transformation between any two frames.

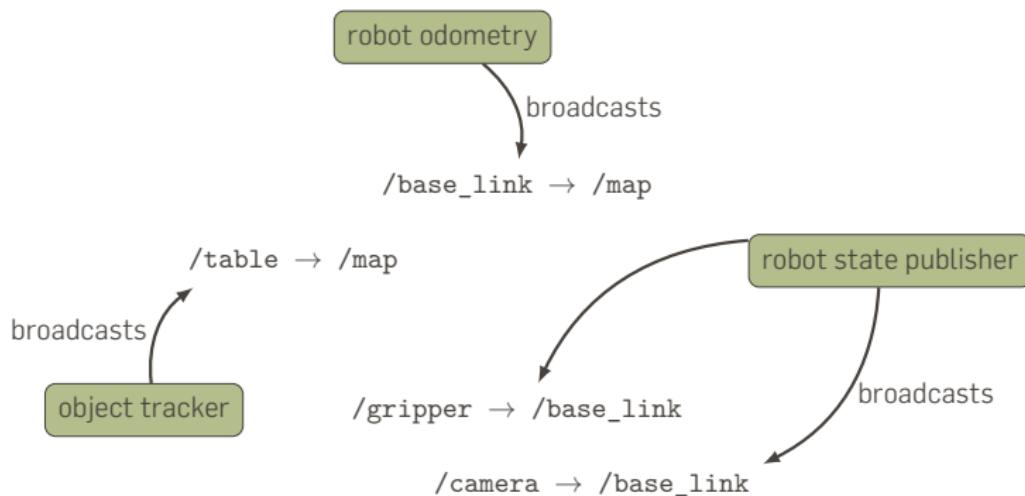
# FRAMES



Attention: even though they look similar, frames and topics are unrelated.

# "CREATING" FRAMES

Frames come to existence as soon as someone (a node) broadcast them.



# HOW TO WRITE A TF BROADCASTER?

C++

```
1 #include <ros/ros.h>
2 #include <tf/transform_broadcaster.h>
3
4
5 int main(int argc, char** argv){
6
7     float x=0.f,y=0.f,theta=0.f;
8
9     ros::init(argc, argv, "my_tf_broadcaster");
10    tf::TransformBroadcaster br;
11    ros::Rate rate(10); // 10 hz
12
13    while (ros::ok()) {
14        tf::Transform transform(
15            tf::Quaternion(0, 0, theta),
16            tf::Vector3(x, y, 0.0));
17
18        br.sendTransform(
19            tf::StampedTransform(transform,
20                ros::Time::now(),
21                "my_robot", "map"));
22
23        x++;
24        rate.sleep();
25    }
26
27    return 0;
}
```

Python

```
1 import rospy
2 import tf
3 from tf.transformations import quaternion_from_euler
4
5 if __name__ == '__main__':
6
7     x = 0.; y = 0.; theta = 0.
8
9     rospy.init_node('my_tf_broadcaster')
10    br = tf.TransformBroadcaster()
11    rate = rospy.Rate(10) # 10hz
12
13    while not rospy.is_shutdown():
14
15        br.sendTransform(
16            (x, y, 0),
17            quaternion_from_euler(0, 0, theta),
18            rospy.Time.now(),
19            "my_robot", "map")
20
21        x += 1
22        rate.sleep()
```

```
> rostopic echo tf
transforms:
-
  header:
    seq: 0
    stamp:
      secs: 1449488936
      nsecs: 480597909
    frame_id: map
  child_frame_id: my_robot
  transform:
    translation:
      x: 239.0
      y: 0.0
      z: 0.0
    rotation:
      x: 0.0
      y: 0.0
      z: 0.0
      w: 1.0
```

## TO SUMMARIZE: KEY CONCEPTS

- Node
- Master
- Messages
- Topics
- Services
- Actions
- Transformations/frames

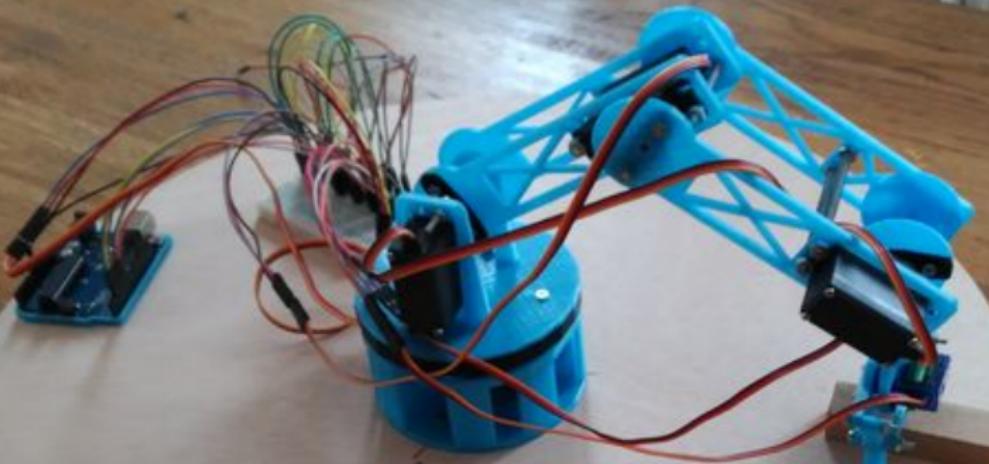
## TO SUMMARIZE: KEY CONCEPTS

- Node
- Master
- Messages
- Topics
- Services
- Actions
- Transformations/frames

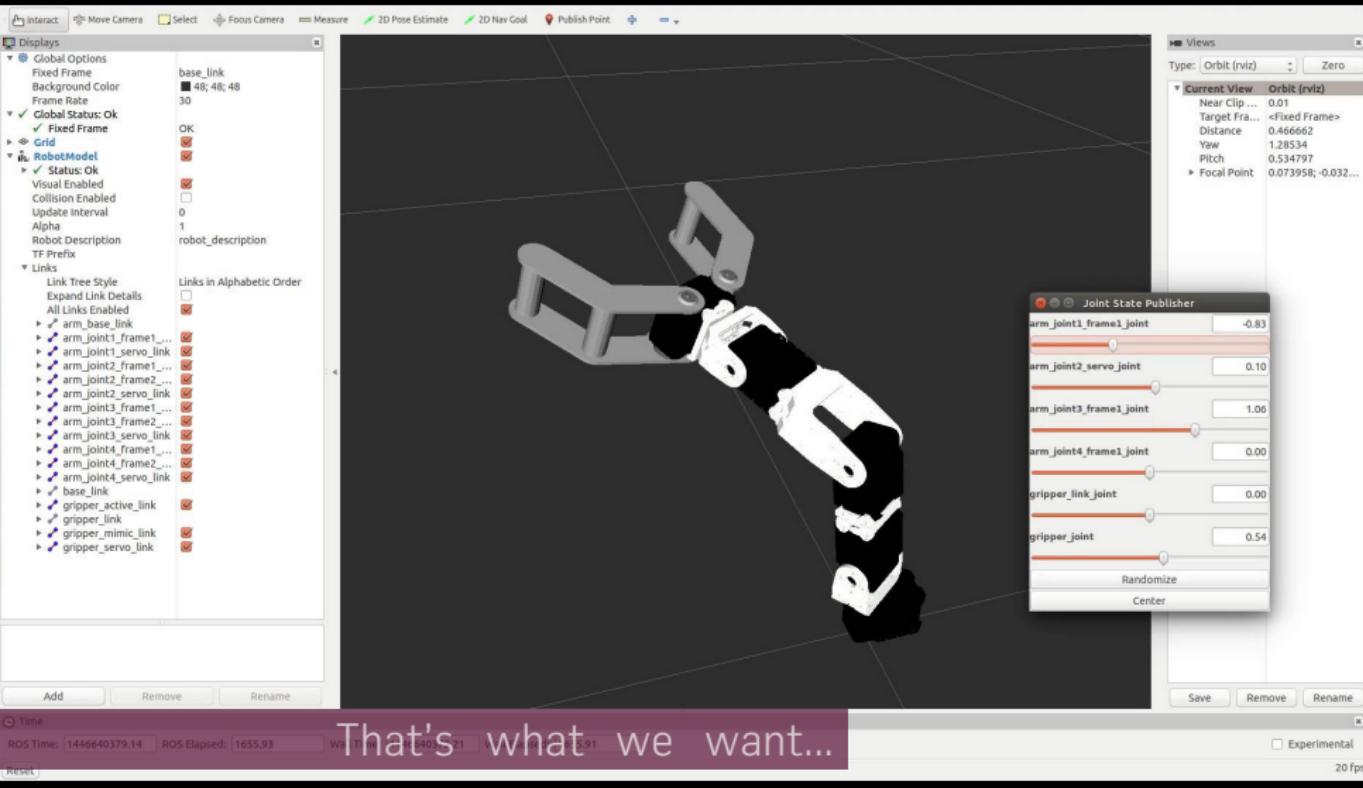
Some additional concepts that we will discuss later on:

- Package
- Launch file

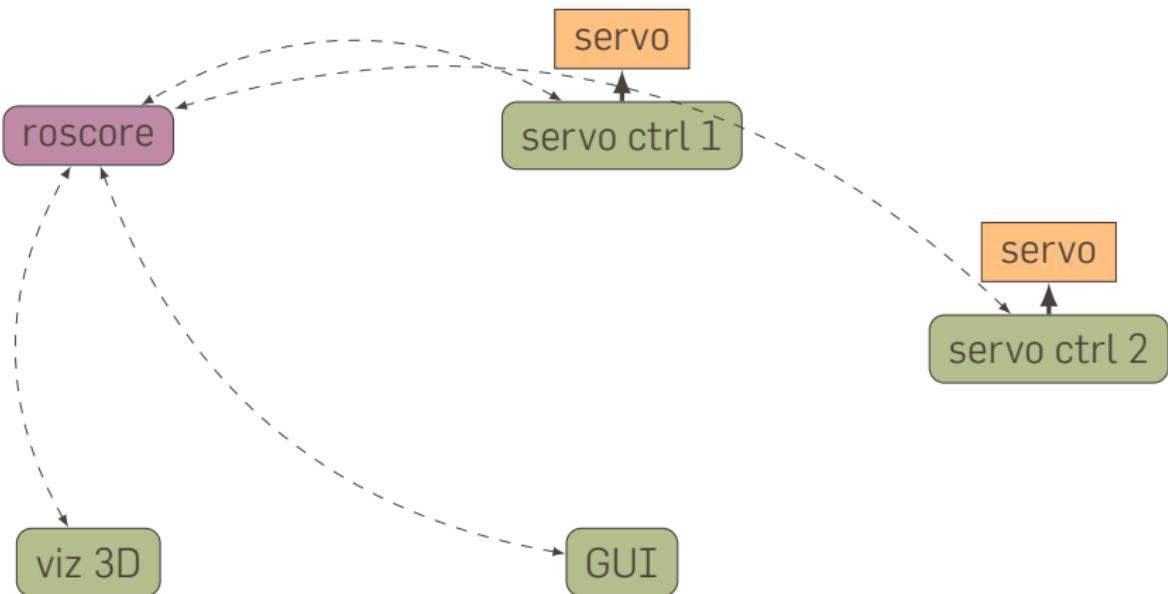
EXAMPLE: A ROBOTIC ARM WITH ROS



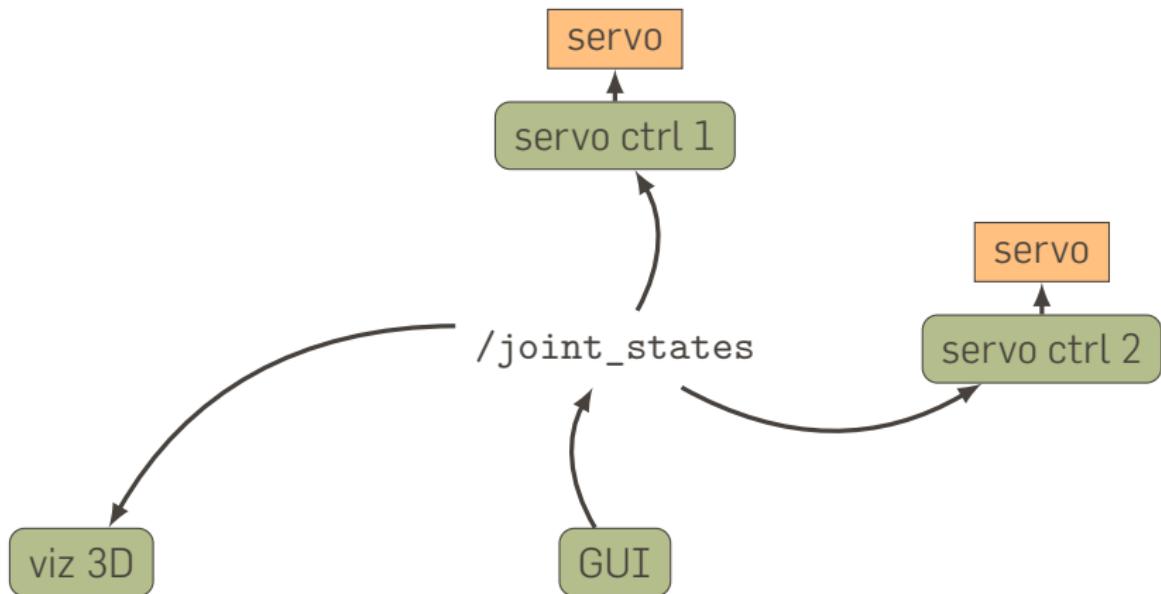
That's our hardware...



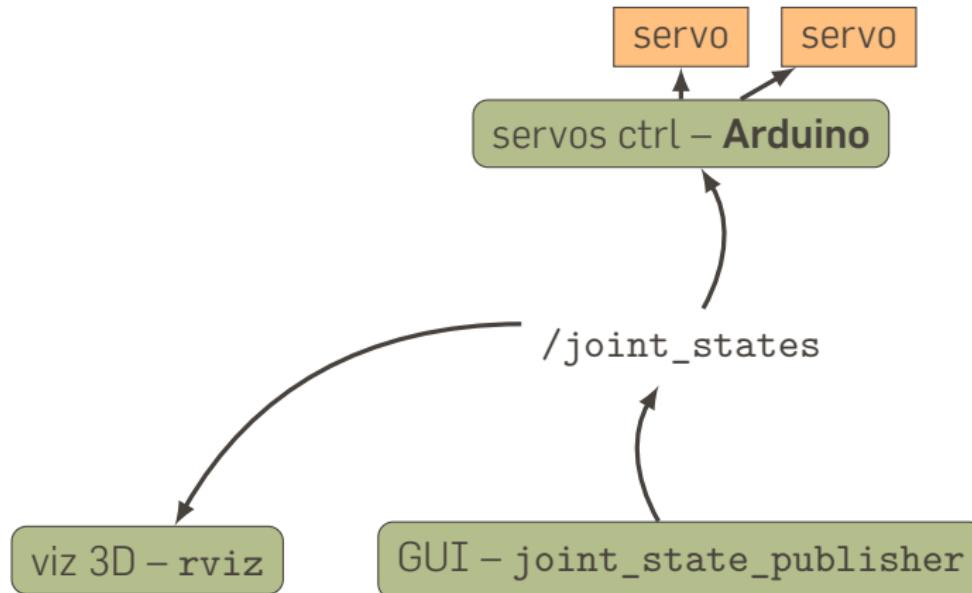
# NODES



# NODES

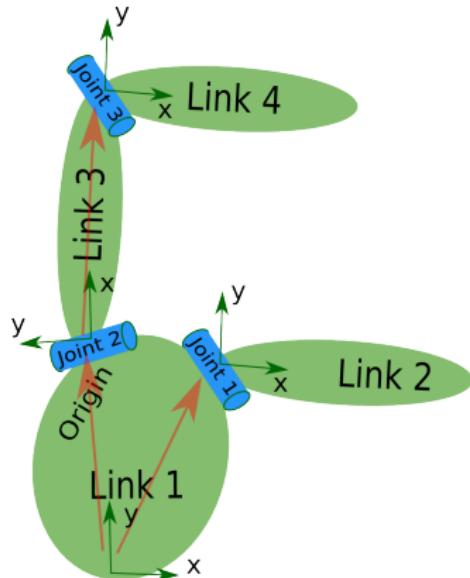


# NODES



# URDF

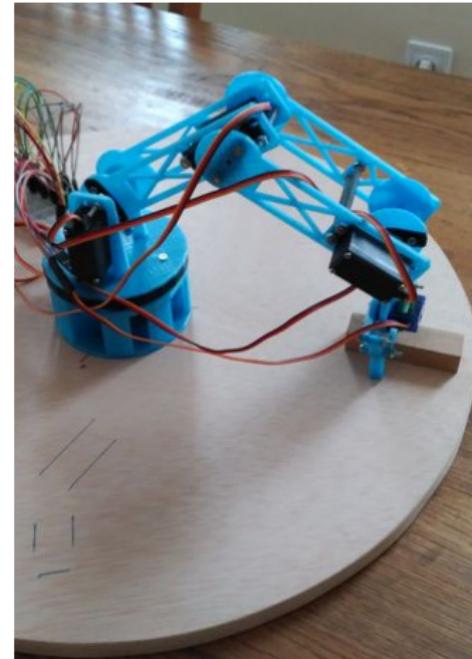
**URDF (*Unified Robot Description Format*)** is an XML-based language to describe a robot.



- Primitives (cylinders, cubes, spheres) to describe the geometry
- For complex geometries, any STL or Collada meshes can be used
- Only *tree structures* can be represented: no parallel robots
- Only *rigid links* can be represented: no soft robots

# URDF: DESCRIBING THE KINEMATICS OF OUR ROBOT

```
<?xml version="1.0"?>
<robot name="roco_arm">
  <link name="base_link">
    <visual>
      <geometry>
        <cylinder length="0.06" radius="0.1"/>
      </geometry>
    </visual>
  </link>
</robot>
```

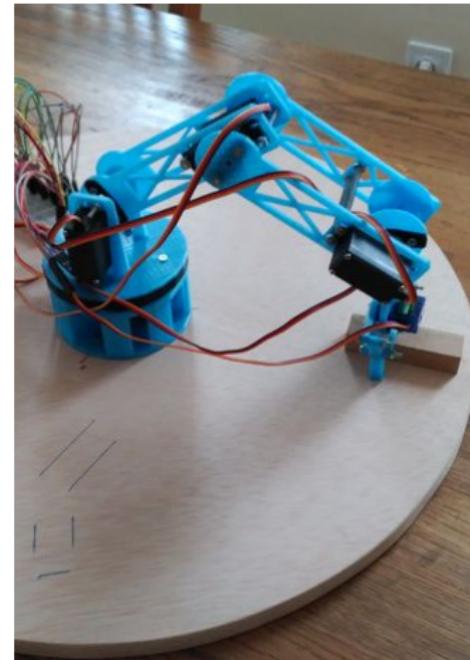


# URDF: DESCRIBING THE KINEMATICS OF OUR ROBOT

```
<?xml version="1.0"?>
<robot name="roco_arm">
  <link name="base_link">
    <visual>
      <geometry>
        <cylinder length="0.06" radius="0.1"/>
      </geometry>
    </visual>
  </link>

  <link name="first_segment">
    <visual>
      <geometry>
        <box size="0.6 0.05 0.1"/>
      </geometry>
      <origin rpy="0 0 0" xyz="-0.3 0 0" />
    </visual>
  </link>

</robot>
```

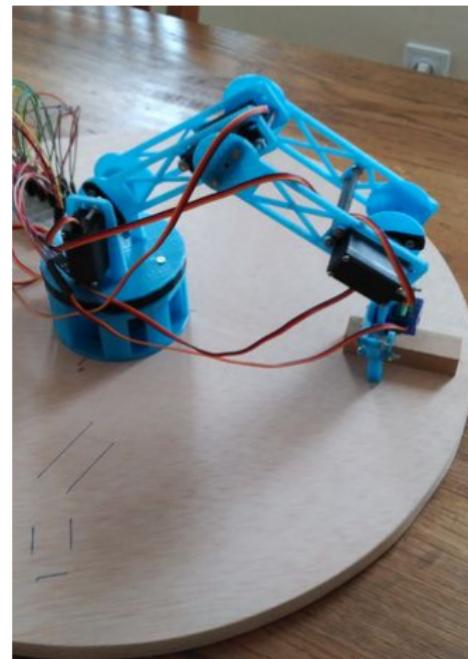


# URDF: DESCRIBING THE KINEMATICS OF OUR ROBOT

```
<?xml version="1.0"?>
<robot name="roco_arm">
  <link name="base_link">
    <visual>
      <geometry>
        <cylinder length="0.06" radius="0.1"/>
      </geometry>
    </visual>
  </link>

  <link name="first_segment">
    <visual>
      <geometry>
        <box size="0.6 0.05 0.1"/>
      </geometry>
      <origin rpy="0 0 0" xyz="-0.3 0 0" />
    </visual>
  </link>

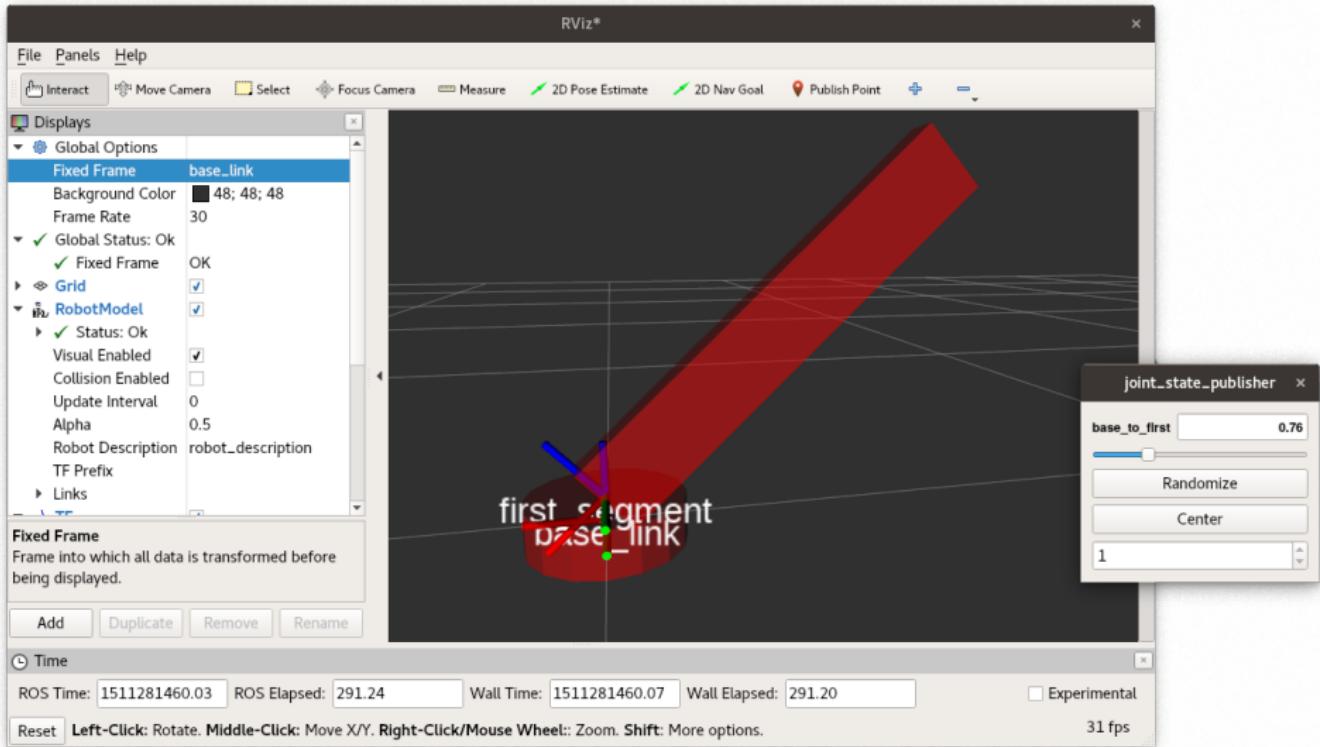
  <joint name="base_to_first" type="revolute">
    <axis xyz="0 1 0" />
    <limit effort="1000" lower="0"
          upper="3.14" velocity="0.5" />
    <parent link="base_link"/>
    <child link="first_segment"/>
    <origin xyz="0 0 0.03" />
  </joint>
</robot>
```

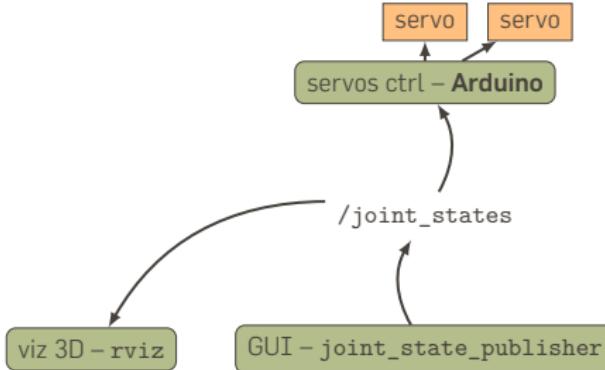


## DISPLAY THE MODEL

To display and interact with the URDF model:

```
> rosparam set robot_description -t code/robot-arm.urdf  
> rosrun robot_state_publisher robot_state_publisher  
> rosrun joint_state_publisher joint_state_publisher _use_gui:=true  
> rosrun rviz rviz
```

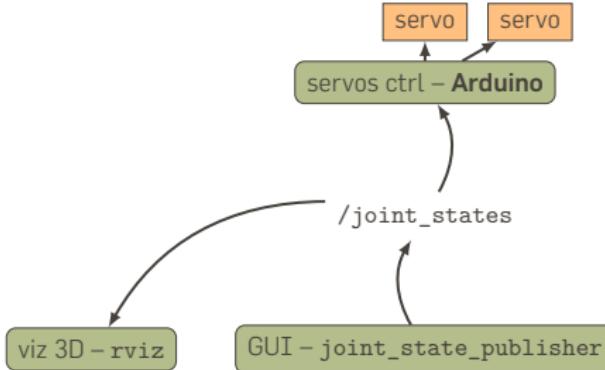




Why do we need `robot_state_publisher`?

`rviz` needs the transformations between each geometry. Going from a **joint state** (i.e. the angles for each joint) to transformations (i.e. **frames**) requires **forward kinematics**.

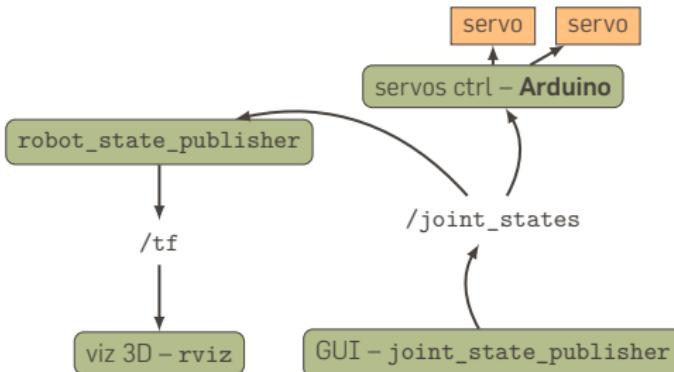
`robot_state_publisher` *subscribes* to the joint state topic `/joint_states` and *broadcasts* the corresponding TF frames.



Why do we need `robot_state_publisher`?

`rviz` needs the transformations between each geometry. Going from a **joint state** (i.e. the angles for each joint) to transformations (i.e. **frames**) requires **forward kinematics**.

`robot_state_publisher` *subscribes* to the joint state topic `/joint_states` and *broadcasts* the corresponding TF frames.



Why do we need `robot_state_publisher`?

`rviz` needs the transformations between each geometry. Going from a **joint state** (i.e. the angles for each joint) to transformations (i.e. **frames**) requires **forward kinematics**.

`robot_state_publisher` *subscribes* to the joint state topic `/joint_states` and *broadcasts* the corresponding TF frames.

## JOINT STATE

---

```
> rosmsg show sensor_msgs/JointState
std_msgs/Header header
    uint32 seq
    time stamp
    string frame_id
string[] name
float64[] position
float64[] velocity
float64[] effort
```

---

Source: [sensor\\_msgs::JointState definition](#)

# READING THE JOINT STATE ON AN ARDUINO

```
1 #include <Servo.h>
2 #include <ros.h>
3 #include <sensor_msgs/JointState.h>
4
5 using namespace ros;
6
7 NodeHandle nh;
8 Servo servo;
9
10 void cb( const sensor_msgs::JointState& msg){
11     int angle = (int) (msg.position[0] * 180/3.14);
12     servo.write(angle); // 0-180
13 }
14
15 Subscriber<sensor_msgs::JointState> sub("joint_states", cb);
16
17 void setup(){
18     nh.initNode();
19     nh.subscribe(sub);
20
21     servo.attach(9); //attach it to pin 9
22 }
23
24 void loop(){
25     nh.spinOnce();
26     delay(1);
27 }
```

# CODE MANAGEMENT

## PACKAGES

The source code of ROS nodes is usually organised into a *package*.

---

```
> cd my_package
> ls
CMakeLists.txt
cfg/
include/
launch/
msgs/
nodes/
package.xml
src/
> rosrun my_package my_node
```

---

One package often contains more than one node.

Besides the source code, packages may contain as well specific messages, configuration files, launch files, etc.

## PACKAGES

The source code of ROS nodes is usually organised into a *package*.

---

```
> cd my_package
> ls
CMakeLists.txt
cfg/
include/
launch/
msgs/
nodes/
package.xml
src/
> rosrun my_package my_node
```

---

Packages must contain a *manifest*, `package.xml`. It contains the package name, version, authors, as well as the dependencies.

## PACKAGES

The source code of ROS nodes is usually organised into a *package*.

---

```
> cd my_package
> ls
CMakeLists.txt
cfg/
include/
launch/
msgs/
nodes/
package.xml
src/
> rosrun my_package my_node
```

---

A new package template can be created easily:

```
> catkin_create_pkg my_package <ROS dependencies>
```

# LET'S CREATE A PROPER ROS PACKAGE

---

```
> cd $HOME
> mkdir src && cd src
> catkin_create_pkg imgproc rospy
```

---

# LET'S CREATE A PROPER ROS PACKAGE

---

```
> cd $HOME  
> mkdir src && cd src  
> catkin_create_pkg imgproc rospy
```

---

---

```
> ls imgproc  
CMakeLists.txt  package.xml  src
```

---

# FIRST, ADD SOME CODE

---

```
> cd imgproc
> mkdir -p src/imgproc && cd src/imgproc
> touch __init__.py # required to create a Python module
> gedit recognition.py
```

---

## FIRST, ADD SOME CODE

---

```
> cd imgproc
> mkdir -p src/imgproc && cd src/imgproc
> touch __init__.py # required to create a Python module
> gedit recognition.py
```

---

Just a simple stub for a Python module:

```
def run(dataset):
    print('Dataset: ' + dataset)
```

## FIRST, ADD SOME INITIAL CODE

Create as well an executable (our future ROS node) in `scripts/`:

---

```
> cd ../..
> mkdir -p scripts && cd scripts
> gedit reco
```

---

## FIRST, ADD SOME INITIAL CODE

Create as well an executable (our future ROS node) in `scripts/`:

---

```
> cd ../..
> mkdir -p scripts && cd scripts
> gedit reco
```

---

```
#!/usr/bin/env python

import imgproc.reco

if __name__ == '__main__':
    imgproc.reco.run("my_faces")
```

## FIRST, ADD SOME INITIAL CODE

Create as well an executable (our future ROS node) in `scripts/`:

---

```
> cd ../..
> mkdir -p scripts && cd scripts
> gedit reco
```

---

```
#!/usr/bin/env python

import imgproc.reco

if __name__ == '__main__':
    imgproc.reco.run("my_faces")
```

---

```
> chmod +x reco
```

---

## CONFIGURE THE PYTHON 'BUILD'

Because our node is written in Python, our `CMakeLists.txt` is simple:

```
cmake_minimum_required(VERSION 2.8.3)
project(imgproc)

find_package(catkin REQUIRED COMPONENTS
    rospy
)

catkin_python_setup()
catkin_package()

install(PROGRAMS
    scripts/reco
    DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION}
)
```

## CONFIGURE THE PYTHON 'BUILD'

However, we need a `setup.py` (standard Python `distutils`-based packaging):

```
from distutils.core import setup
from catkin_pkg.python_setup import generate_distutils_setup

# fetch values from package.xml
setup_args = generate_distutils_setup(
    packages=['imgproc'],
    package_dir={'': 'src'},
)
setup(**setup_args)
```

# INSTALL THE NODE

We can now install our node:

---

```
> cd ..  
> mkdir -p build && cd build  
> cmake -DCMAKE_INSTALL_PREFIX=<install prefix> ..  
> make install
```

---

## INSTALL THE NODE

We can now install our node:

---

```
> cd ..  
> mkdir -p build && cd build  
> cmake -DCMAKE_INSTALL_PREFIX=<install prefix> ..  
> make install
```

---

Assuming ROS is correctly installed, we can run our node:

---

```
> export ROS_PACKAGE_PATH=<prefix>/share:$ROS_PACKAGE_PATH  
> rosrun imgproc reco  
Dataset: my_faces
```

---

# IMAGE PROCESSING

Let's update the node `reco` and the library (Python *module*) `recognition.py` to perform simple image processing:

`recognition.py:`

```
import cv2

def run(image):
    rows, cols, channels = image.shape
    cv2.circle(image, (cols/2, rows/2), 50, (0,0,255), -1)
```

# IMAGE PROCESSING

reco:

```
#! /usr/bin/env python

import sys, rospy
from sensor_msgs.msg import Image
from cv_bridge import CvBridge

import imgproc.reco

def on_image(image):
    cv_image = bridge.imgmsg_to_cv2(image, "bgr8")
    imgproc.reco.run(cv_image)
    image_pub.publish(bridge.cv2_to_imgmsg(cv_image, "bgr8"))

if __name__ == '__main__':
    rospy.init_node('image_processor')
    bridge = CvBridge()
    image_sub = rospy.Subscriber("image",Image, on_image)
    image_pub = rospy.Publisher("processed_image",Image, queue_size=1)

    while not rospy.is_shutdown():
        rospy.spin()
```

## TO USE THE NODE

---

```
> rosrun usb_cam usb_cam_node
```

---

---

```
> rosrun imgproc reco image:=/usb_cam/image_raw
```

---

---

```
> rqt_image_view image:=/processed_image
```

---

Let's try it!

## LAUNCH FILES

*Launch files* allow to group together nodes and their configuration.

```
<launch>
  <arg name="model" />
  <arg name="gui" default="true" />

  <param name="robot_description" textfile="$(arg model)" />
  <param name="use_gui" value="$(arg gui)"/>

  <node name="joint_state_publisher" pkg="joint_state_publisher"
        type="joint_state_publisher" />
  <node name="robot_state_publisher" pkg="robot_state_publisher"
        type="state_publisher" />
  <node name="rviz" pkg="rviz" type="rviz" required="true" />

</launch>
```

## LAUNCH FILES

```
<launch>
  <arg name="model" />
  <arg name="gui" default="true" />

  <param name="robot_description" textfile="$(arg model)" />
  <param name="use_gui" value="$(arg gui)"/>

  <node name="joint_state_publisher" pkg="joint_state_publisher"
        type="joint_state_publisher" />
  <node name="robot_state_publisher" pkg="robot_state_publisher"
        type="state_publisher" />
  <node name="rviz" pkg="rviz" type="rviz" required="true" />

</launch>
```

To use the launch file, call `roslaunch` instead of `rosrun`:

```
> rosrun my_package interactive_arm.launch
```

## LAUNCH FILES

```
<launch>
  <arg name="model" />
  <arg name="gui" default="true" />

  <param name="robot_description" textfile="$(arg model)" />
  <param name="use_gui" value="$(arg gui)"/>

  <node name="joint_state_publisher" pkg="joint_state_publisher"
        type="joint_state_publisher" />
  <node name="robot_state_publisher" pkg="robot_state_publisher"
        type="state_publisher" />
  <node name="rviz" pkg="rviz" type="rviz" required="true" />

</launch>
```

Arguments (<arg>) can be provided from the command-line:

```
> roslaunch my_package interactive_arm.launch
model:=my_arm.urdf
```

## LAUNCH FILES

```
<launch>
  <arg name="model" />
  <arg name="gui" default="true" />

  <param name="robot_description" textfile="$(arg model)" />
  <param name="use_gui" value="$(arg gui)"/>

  <node name="joint_state_publisher" pkg="joint_state_publisher"
        type="joint_state_publisher" />
  <node name="robot_state_publisher" pkg="robot_state_publisher"
        type="state_publisher" />
  <node name="rviz" pkg="rviz" type="rviz" required="true" />

</launch>
```

Parameters (`<param>`) are loaded to the ROS *parameter server* and shared with all the ROS nodes.

## LAUNCH FILES

```
<launch>
  <arg name="model" />
  <arg name="gui" default="true" />

  <param name="robot_description" textfile="$(arg model)" />
  <param name="use_gui" value="$(arg gui)"/>

  <node name="joint_state_publisher" pkg="joint_state_publisher"
        type="joint_state_publisher" />
  <node name="robot_state_publisher" pkg="robot_state_publisher"
        type="state_publisher" />
  <node name="rviz" pkg="rviz" type="rviz" required="true" />

</launch>
```

Some nodes may be marked as `required`: if they die (closed or crashed), all the other nodes in the launch file are killed as well.

## LAUNCH FILES

```
<launch>
  <arg name="model" />
  <arg name="gui" default="true" />

  <param name="robot_description" textfile="$(arg model)" />
  <param name="use_gui" value="$(arg gui)"/>

  <node name="joint_state_publisher" pkg="joint_state_publisher"
        type="joint_state_publisher" />
  <node name="robot_state_publisher" pkg="robot_state_publisher"
        type="state_publisher" />
  <node name="rviz" pkg="rviz" type="rviz" required="true" />

</launch>
```

Many more possibilities, see roslaunch documentation.

TOOLS

# RViz\*

File Panels Help

Interact Move Camera Select Focus Camera Measure 2D Pose Estimate 2D Nav Goal Publish Point + -

## Displays

### Global Options

Fixed Frame base\_link  
Background Color 48; 48; 48  
Frame Rate 30

### Global Status: Ok

✓ Fixed Frame

### Grid



### Grid

Displays a grid along the ground plane, centered at the origin of the target frame of reference.

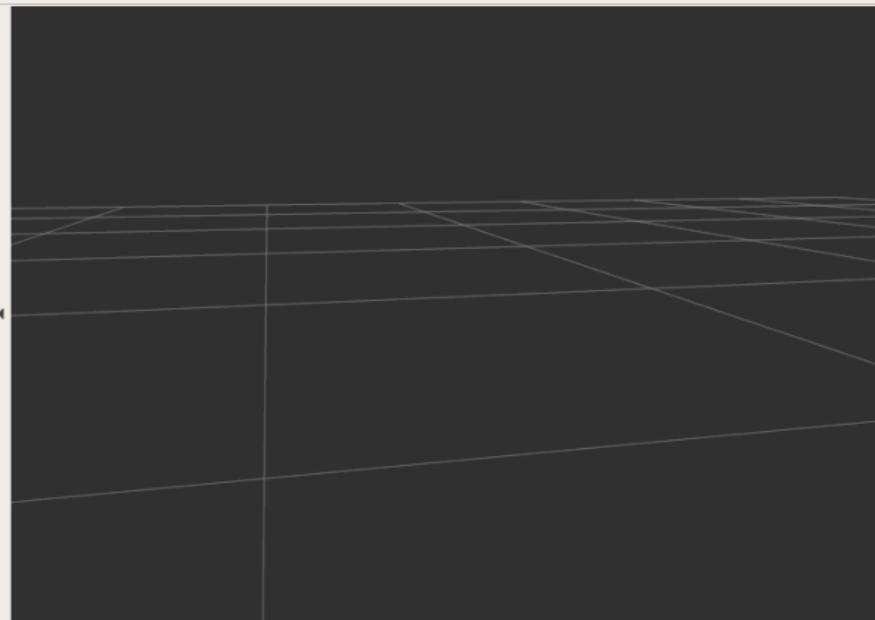
[More Information](#)

Add

Duplicate

Remove

Rename



### Time

ROS Time: 1511281694.55

ROS Elapsed: 525.75

Wall Time: 1511281694.58

Wall Elapsed: 525.75

Experimental

Reset

31 fps

## RViz\*

File Panels Help

Interact

Move Camera

Select

Focus Camera

Measure

2D Pose Estimate

2D Nav Goal

Publish Point

+

-

## Displays

## Global Options

Fixed Frame base\_link  
 Background Color [ 48; 48; 48 ]  
 Frame Rate 30

## Global Status: Ok

Fixed Frame OK

## Grid

## Grid

Displays a grid along the ground plane, centered at the origin of the target frame of reference.

[More Information](#)

Add

Duplicate

Remove

Rename

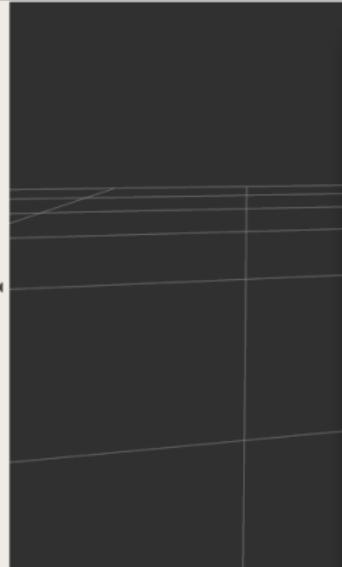
## Time

ROS Time: 1511281588.67

ROS Elapsed: 419.88

Wall Time: 1511281588.71

Reset Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click/Mouse Wheel: Zoom. Shift: More



rviz

Create visualization

By display type By topic

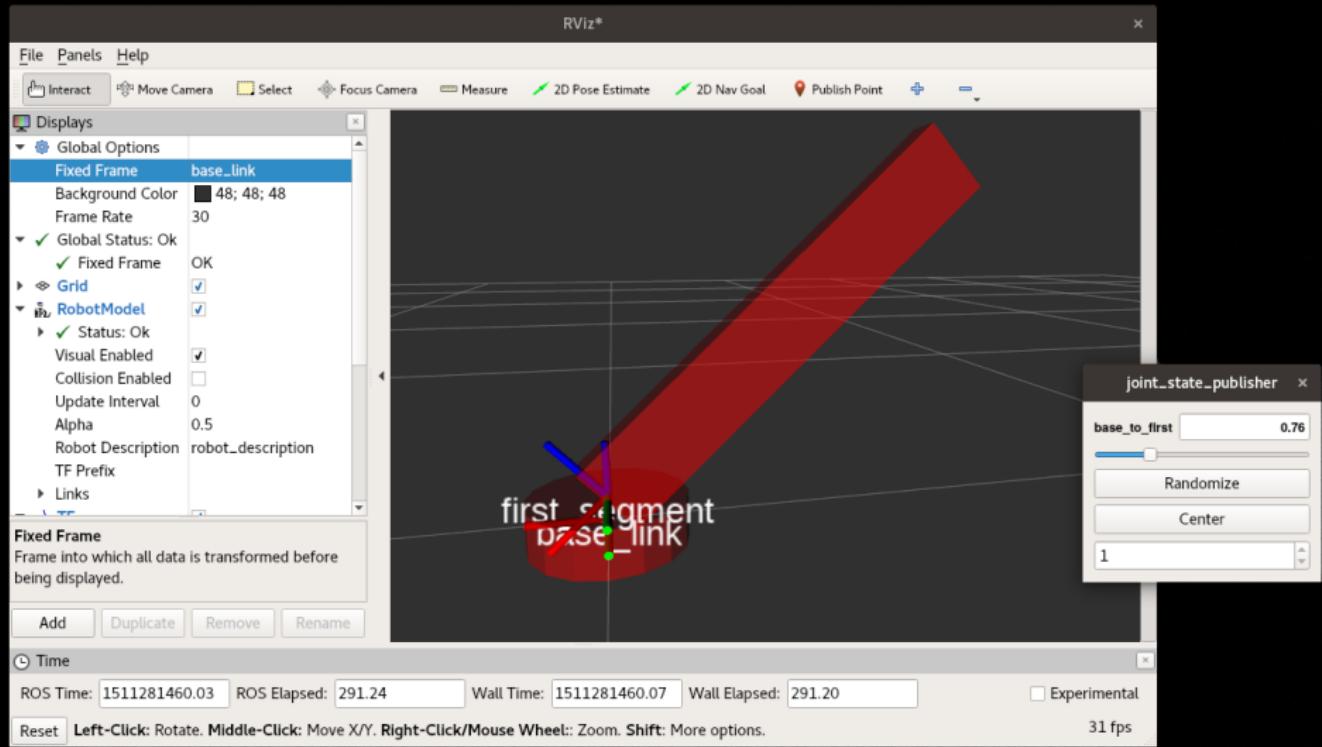
- Odometry
- Path
- PointCloud
- PointCloud2
- PointStamped
- Polygon
- Pose
- PoseArray
- Range
- RelativeHumidity
- RobotModel
- TF
- Temperature
- WrenchStamped

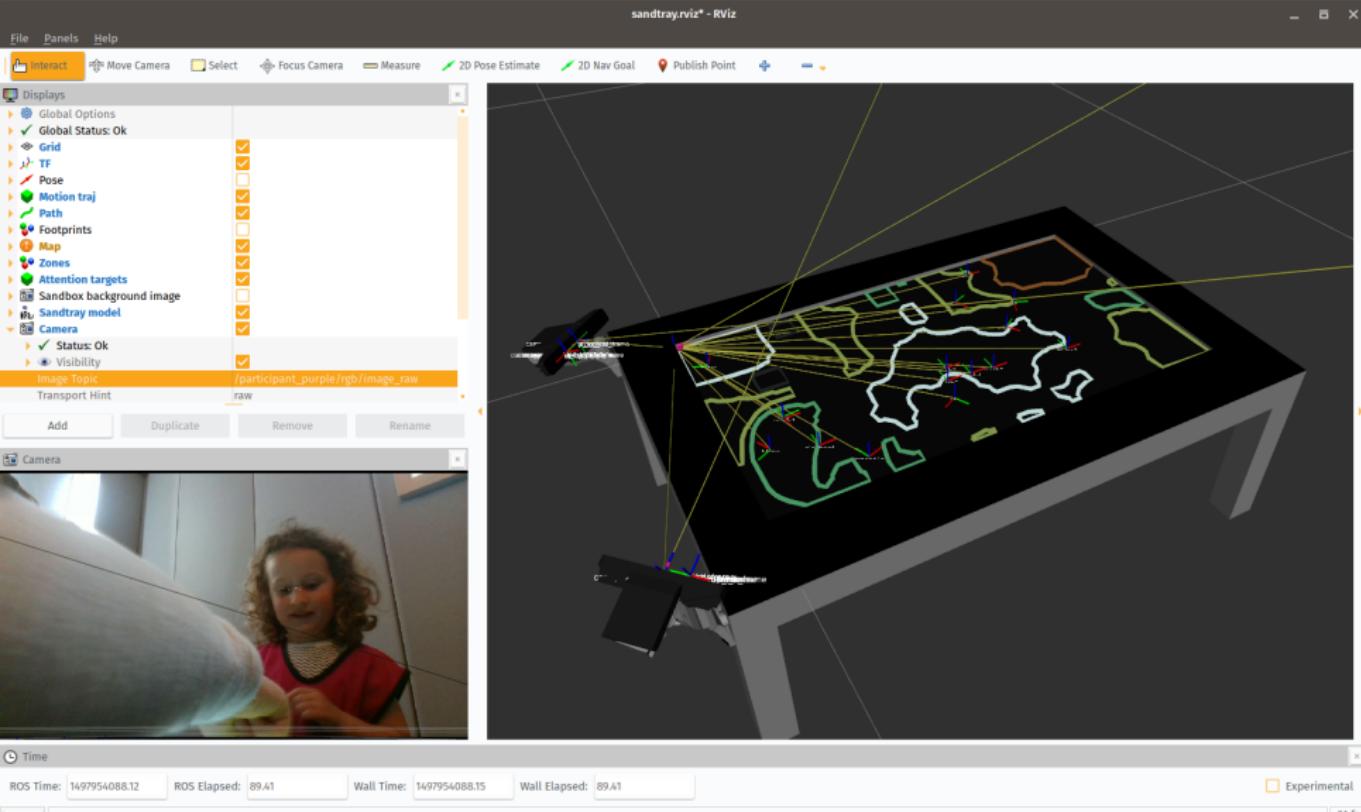
Description:

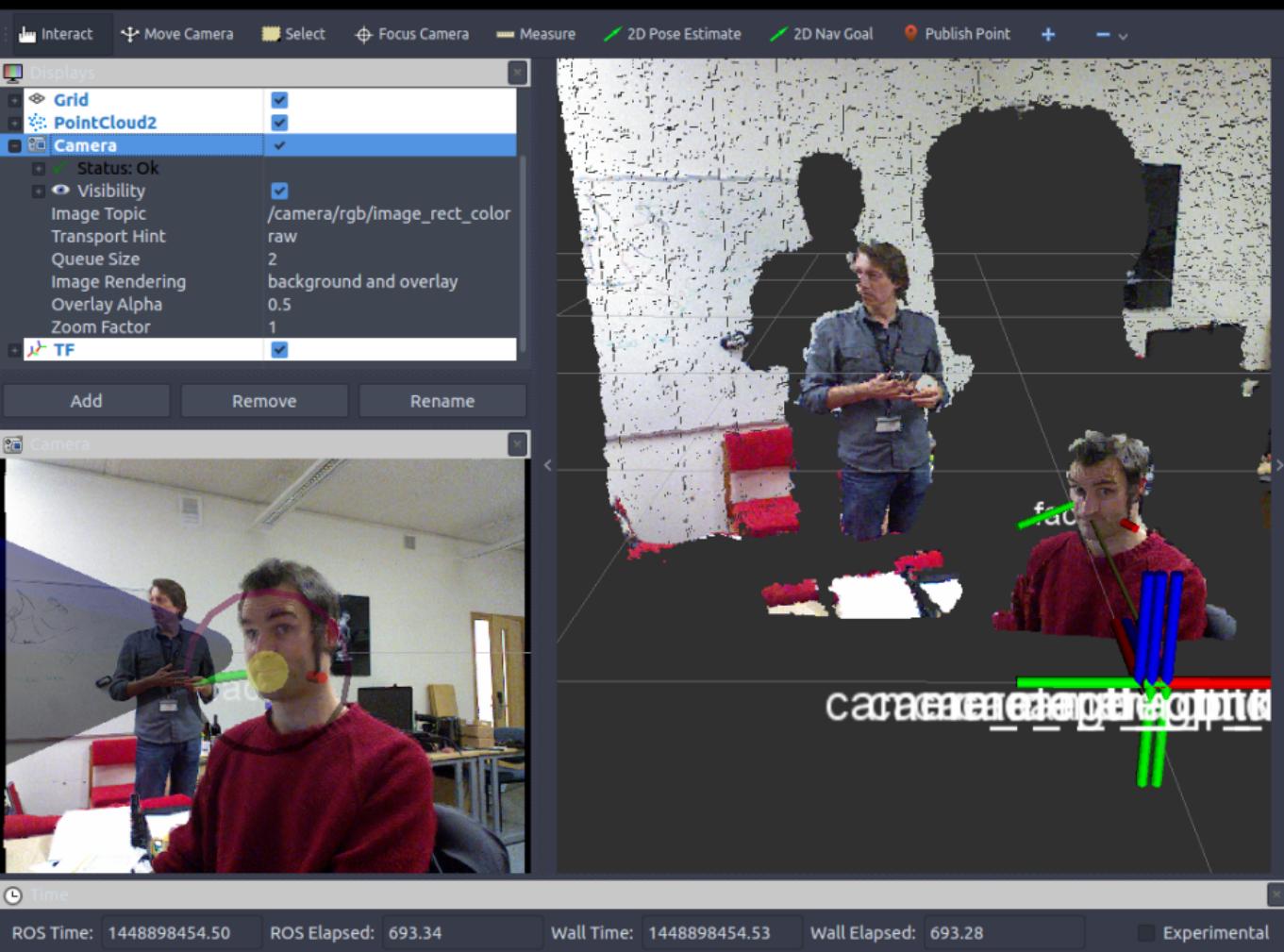
Displays a visual representation of a robot in the correct pose (as defined by the current TF transforms). [More Information](#).

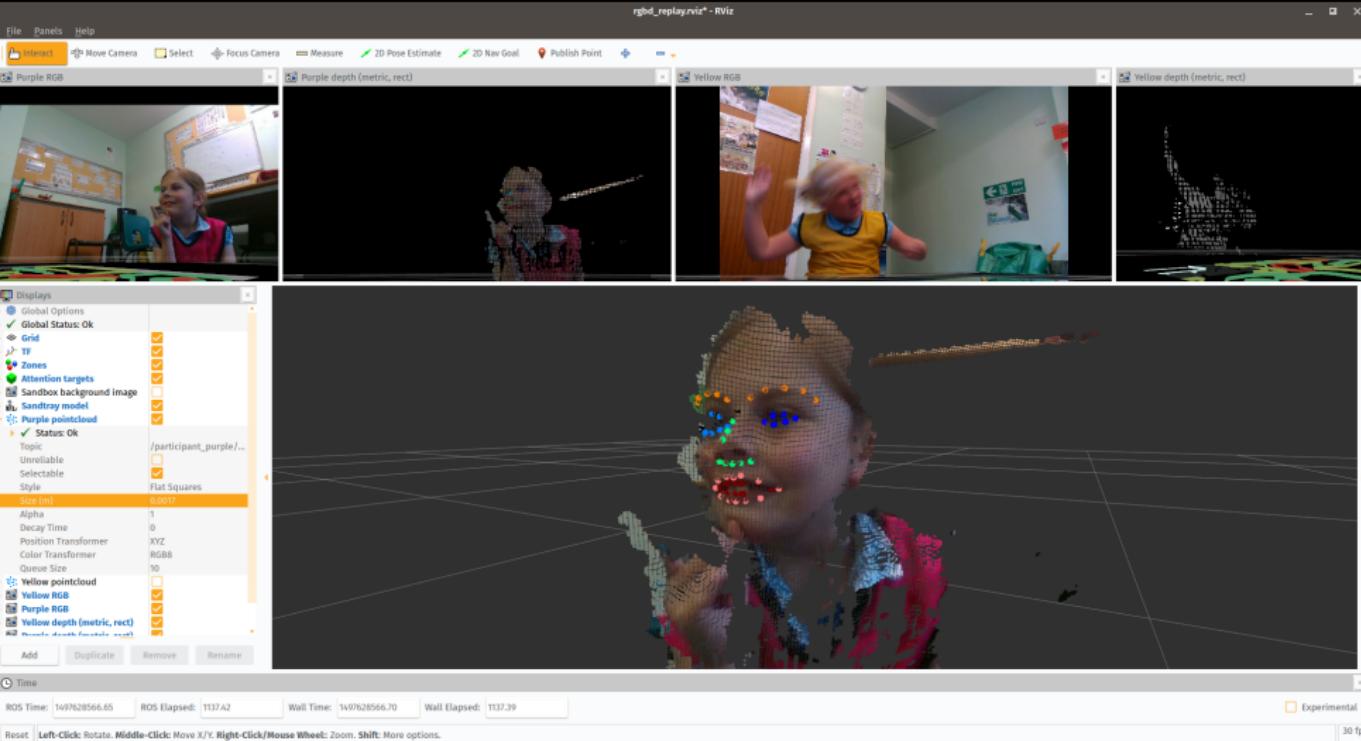
Display Name

Cancel OK









## OTHER TOOLS

---

<code>rosnode</code> , <code>rostopic</code> ,...	Print out/publish/call messages, services, nodes
<code>rosconsole</code>	Centralized logging
<code>rosbag</code>	Record and replay messages
<code>rqt_reconfigure</code>	Live configuration of nodes
<code>rqt_diagnostics</code>	Standardized diagnostics
<code>rosgraph</code>	plots the node network

---

```
> rosnode list
/camera_base_link
/camera_base_link1
/camera_base_link2
/camera_base_link3
/ros_attention_tracker
/rosout
/camera/camera_nodelet_manager
/camera/debayer
/camera/depth_metric
/camera/depth_metric_rect
/camera/depth_points
/camera/depth_rectify_depth
/camera/depth_registered_hw_metric_rect
/camera/depth_registered_metric
/camera/depth_registered_rectify_depth
/camera/depth_registered_sw_metric_rect
/camera/disparity_depth
/camera/disparity_registered_hw
/camera/disparity_registered_sw
```

```
> rostopic list
/camera_info
/image
/nb_detected_faces
/rosout
/rosout_agg
/tf
/camera/depth/image_rect_raw
/camera/depth/image_rect_raw/compressed
/camera/depth/image_rect_raw/compressed/parameter_descriptions
/camera/depth/image_rect_raw/compressed/parameter_updates
/camera/depth/image_rect_raw/compressedDepth
/camera/depth/image_rect_raw/compressedDepth/parameter_descript
/camera/depth/image_rect_raw/compressedDepth/parameter_updates
/camera/depth/image_rect_raw/theora
/camera/depth/image_rect_raw/theora/parameter_descriptions
/camera/depth/image_rect_raw/theora/parameter_updates
/camera/depth_rectify_depth/parameter_descriptions
/camera/depth_rectify_depth/parameter_updates
```

```
> rostopic echo tf
transforms:
-
  header:
    seq: 0
    stamp:
      secs: 1449222890
      nsecs: 396561780
      frame_id: /camera_link
    child_frame_id: /camera_rgb_frame
    transform:
      translation:
        x: 0.0
        y: -0.045
        z: 0.0
      rotation:
        x: 0.0
        y: 0.0
        z: 0.0
        w: 1.0
```

## rqt\_console\_\_Console - rqt

Console

D ? - O

Displaying 10 messages



Fit Columns

#	Message	Severity	Node	Stamp	Topics	Location
#8	Connection::drop(0)	Debug	/ros_attention_tracker	16:17:21.98...	/attention_...	/home/sl...
#7	TCP socket [18] closed	Debug	/ros_attention_tracker	16:17:21.98...	/attention_...	/home/sl...
#6	Connection::drop(2)	Debug	/ros_attention_tracker	16:17:21.98...	/attention_...	/home/sl...
#5	head_pose_estimator is r...	Info	/ros_attention_tracker	16:17:02.54...	/attention_...	/home/sl...
#4	Initializing the face detec...	Info	/ros_attention_tracker	16:17:01.48...	/rosout	/home/sl...
#3	Started stream.	Info	/v4l/gscam_driver_v4l	16:16:46.65...	/rosout, /v4...	/home/sl...
#2	Publishing stream...	Info	/v4l/gscam_driver_v4l	16:16:46.65...	/rosout, /v4...	/home/sl...
#1	Time offset: 1448897724....	Info	/v4l/gscam_driver_v4l	16:16:45.45...	/rosout	/home/sl...

Exclude Messages...

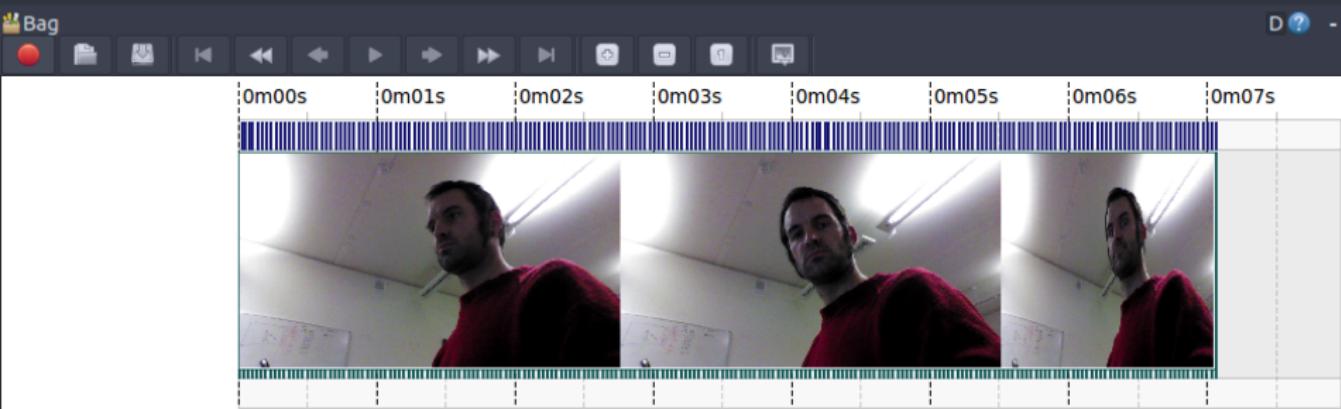
 ...with severities: Debug Info Warn Error Fatal
 

Highlight Messages...

 ...containing: face
 

Regex

## rqt\_bag\_Bag - rqt



1449246342.413s Dec 04 2015 16:25:42.413 0.000s

## rqt\_reconfigure\_Param - rqt

### Dynamic Reconfigure

Filter key:

Collapse all

Expand all

+ attention\_tracker

- camera  
debayer

+ depth  
depth\_rectify\_depth

+ depth\_registered  
depth\_registered\_rectify\_depth  
driver

+ ir  
rectify\_color  
rectify\_ir  
rectify\_mono

+ rgb

Refresh

/camera driver

image_mode	SXGA_15Hz (1)	▼
depth_mode	VGA_30Hz (2)	▼
depth_registration	<input checked="" type="checkbox"/>	
data_skip	0	1000 0
depth_time_offset	-1.0	1.0 0.0
image_time_offset	-1.0	1.0 0.0
depth_ir_offset_x	-10.0	10.0 5.0
depth_ir_offset_y	-10.0	10.0 4.0
z_offset_mm	-200	200 0
z_scaling	0.5	1.5 1.0

# ROS DOCUMENTATION

# ROS DOCUMENTATION

Plenty!

Some examples:

- o Tutorial: [wiki.ros.org/ROS/Tutorials](http://wiki.ros.org/ROS/Tutorials)
- o Supported robots: [robots.ros.org](http://robots.ros.org)
- o Message definition example: [sensor\\_msgs/LaserScan](http://sensor_msgs/LaserScan)
- o Node documentation example: [wiki.ros.org/face\\_detector](http://wiki.ros.org/face_detector)

SO, WHAT IS ROS?

# ROS IS...

- o A fairly simple peer-to-peer message passing system designed with robotics in mind

# ROS IS...

- A fairly simple peer-to-peer message passing system designed with robotics in mind
- An API to this system (in several languages – C++ and Python are 1st tier)

# ROS IS...

- A fairly simple peer-to-peer message passing system designed with robotics in mind
- An API to this system (in several languages – C++ and Python are 1st tier)
- A set of standard message types that facilitate interoperability between modules

# ROS IS...

- A fairly simple peer-to-peer message passing system designed with robotics in mind
- An API to this system (in several languages – C++ and Python are 1st tier)
- A set of standard message types that facilitate interoperability between modules
- A set of conventions to write and package robotic softwares

## ROS IS...

- o A fairly simple peer-to-peer message passing system designed with robotics in mind
- o An API to this system (in several languages – C++ and Python are 1st tier)
- o A set of standard message types that facilitate interoperability between modules
- o A set of conventions to write and package robotic softwares
- o Deep integration of a few key open-source libraries (OpenCV, PCL, tf)

## ROS IS...

- A fairly simple peer-to-peer message passing system designed with robotics in mind
- An API to this system (in several languages – C++ and Python are 1st tier)
- A set of standard message types that facilitate interoperability between modules
- A set of conventions to write and package robotic softwares
- Deep integration of a few key open-source libraries (OpenCV, PCL, tf)
- A set of tools to run and monitor the nodes

That's all, folks!

Questions:

Portland Square B316 or **severin.lemaignan@plymouth.ac.uk**

Slides:

[github.com/severin-lemaignan/module-mobile-and-humanoid-robots](https://github.com/severin-lemaignan/module-mobile-and-humanoid-robots)