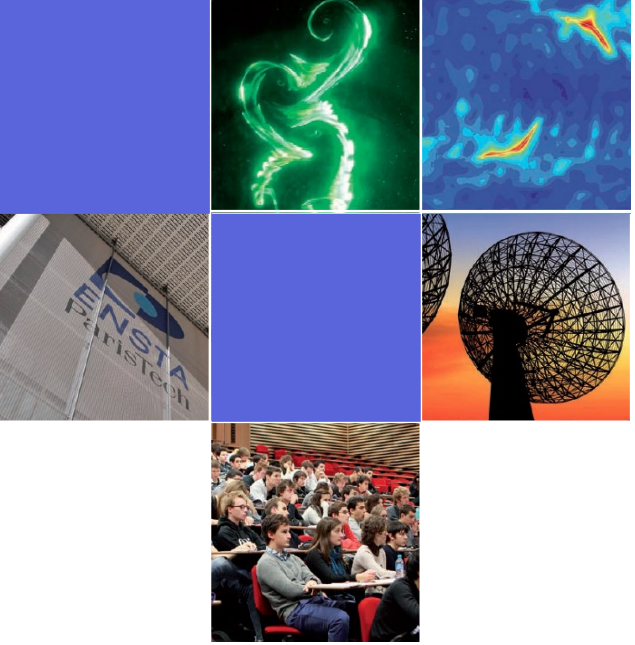


ENSTA | INSTITUT POLYTECHNIQUE DE PARIS

Introduction to ROS
CSC_5RO14_TA – Session 3

Emmanuel Battesti

Overview Course 3

- ROS Time
- ROS Bags
- TF2 Transformation System
- rqt User Interface
- Robot models (URDF)
- Simulation descriptions (SDF)

ROS Time

- Normally, ROS uses the PC's system clock as its time source : ***wall time***
- For simulations or playback of logged data, it is convenient to work with a **simulated time** (pause, slow-down, etc.)
- To work with a **simulated clock**:
  - Set the `/use_sim_time` parameter

```
> rosparam set use_sim_time true
```

  - One « **clock server** » should publish the time on the topic /clock, can be :
    - Gazebo (enabled by default)
    - ROS bag (use the `--clock` option)

- To take advantage of the simulated time, you should always use the ROS Time APIs everywhere in your code:

`ros::Time`

```
ros::Time begin = ros::Time::now();
double secs = begin.toSec();
```

`ros::Duration`

```
ros::Duration duration(0.5); // 0.5s
```

`ros::Rate`

```
ros::Rate rate(10); // 10Hz
```

- If **only *wall time*** is required, use `ros::WallTime`, `ros::WallDuration`, and `ros::WallRate`.

Reference :
https://wiki.ros.org/Clock
https://wiki.ros.org/roscpp/Overview/Time

ROS Time

- The returned value of `ros::time::now()` depends on whether the `use_sim_time` parameter is set:
  1) If **`use_sim_time == false`**, `ros::time::now()` returns the **system time** (seconds since 1970-01-01 0:00, so something like 1738912345.123456).
  2) If **`use_sim_time == true`**, and you play a **rosbag**, ros::time::now() returns the time when the rosbag was recorded (probably also something like 1738912345.123456).
  3) If **`use_sim_time == true`**, and you are running a **simulator** like Gazebo, `ros::time::now()` returns the time from when the simulation was started, starting from zero (so probably something like 63.123 if the simulator has been running for 63.123 seconds).
- **In simulation time** (cases 2 and 3), ros::Duration is constant, regardless of whether the rosbag (or the simulation) is running at 0.1x, 1.0x or 10.0x real time.
- **In simulation time,** ros::time::now() returns **time 0** until the first message is received on */clock*, so 0 = « client does not know clock time yet ». Idea : it may be useful to loop over now() until non-zero is returned.

ROS Bags

- A bag is a format for **storing message** data.
- Binary format with ***.bag** file extension.
- Suitable for logging and recording datasets for later visualization and analysis.
- Record all topics in a bag:

```
> rosbag record --all
```

- Record given topics:

```
> rosbag record topic1 topic2 topic3
```

- Stop recording with Ctrl + C
- Bags are saved with start date and time as file name in the current folder, e.g. 2023-02-10-14-27-13.bag.

- Show information about a bag:

```
> rosbag info bag_name.bag
```

- Read a bag and publish its contents:

```
> rosbag play bag_name.bag
```

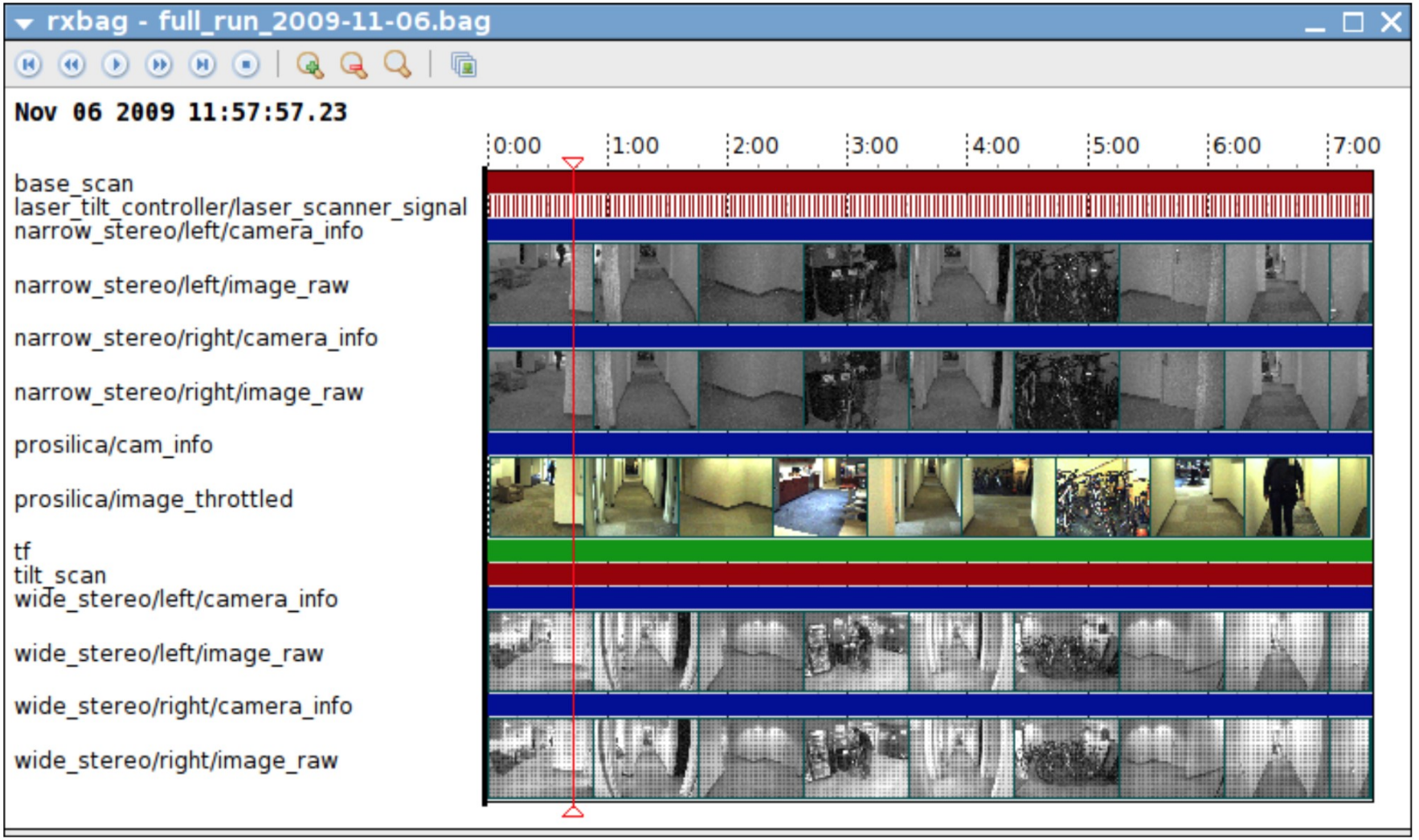
- Playback options can be defined e.g.:

```
> rosbag play --rate=0.5 bag_name.bag
```

  - `--rate=factor` Publish the rate factor
  - `--clock` Publish the clock time (set the *use_sim_time* parameter to true)
  - `--loop` Loop playback

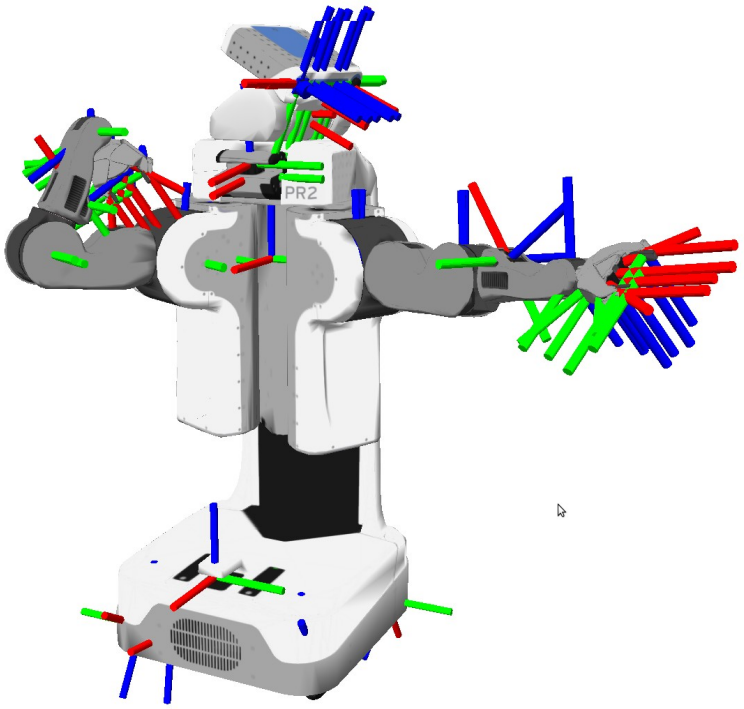
Reference :
https://wiki.ros.org/rosbag/

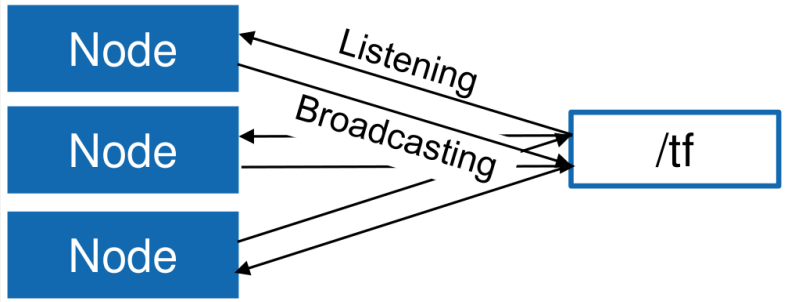
ROS Bags - rqt_bag



TF2 Transformation System

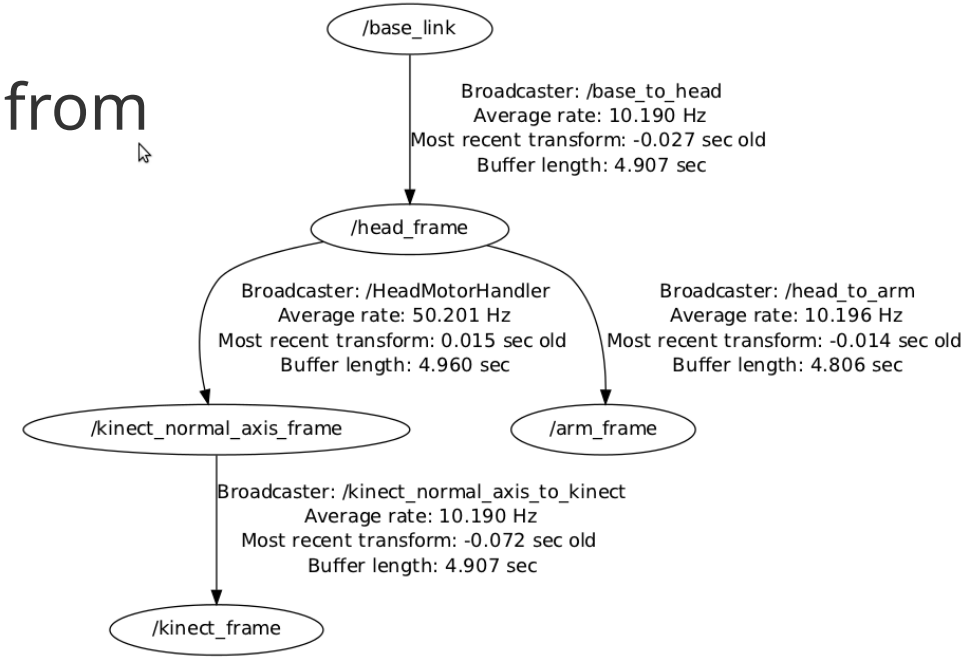
- Tool for keeping track of coordinate frames over time (such as a world frame, base frame, gripper frame, head frame, etc.).
- TF maintains the relationship between coordinate frames in a temporally buffered tree structure.
- Allow the user to transform points, vectors, etc. between coordinate frames at any desired point in time.
- Implemented as a publisher/subscriber model on the topics `/tf` and `/tf_static`.





TF2 Transformation System : Transform Tree

- TF listeners use a buffer to listen to all broadcasted transforms
- Query for specific transforms from the transform tree



tf2_msgs/TFMessage.msg

```
geometry_msgs/TransformStamped[] transforms
  std_msgs/Header header
    uint32 seqtime stamp
    string frame_id
  string child_frame_id
  geometry_msgs/Transform transform
    geometry_msgs/Vector3 translation
    geometry_msgs/Quaternion rotation
```

TF2 Transformation System : Tools

Command line

- Print information about the current transform tree

```
> rosrun tf tf_monitor
```

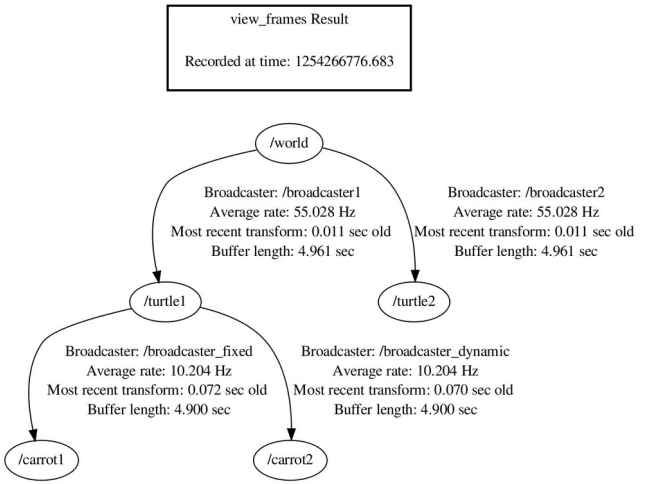
- Print information about the transformation between two frames

```
> rosrun tf tf_echo
source_frame target_frame
```

View Frames

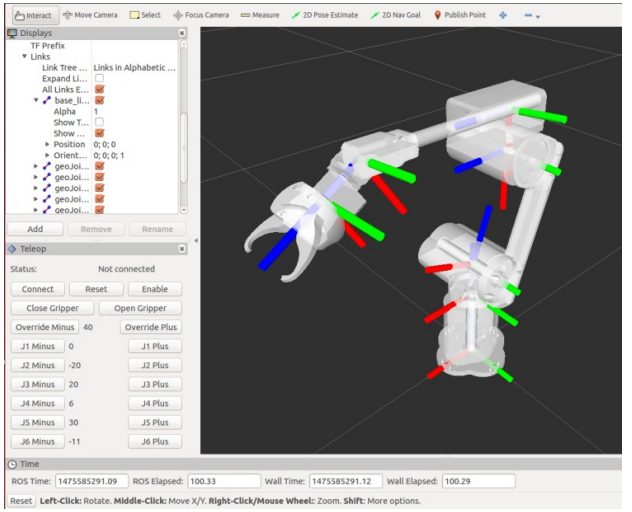
- Creates a visual graph (PDF) of the transformation tree

```
> rosrun tf view_frames
```

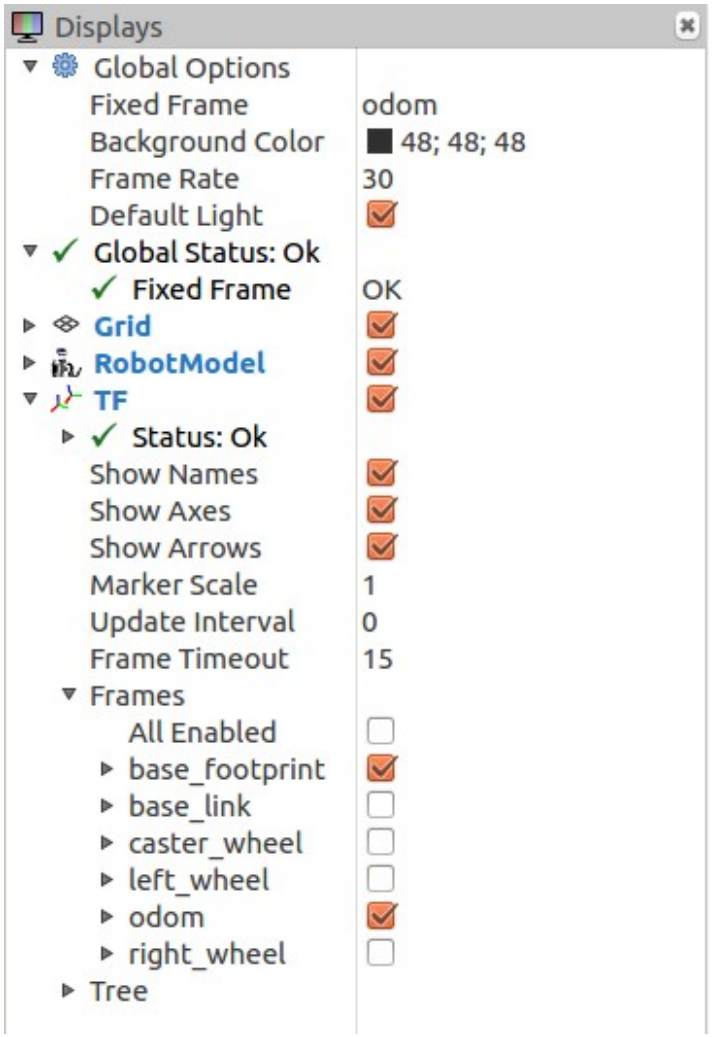


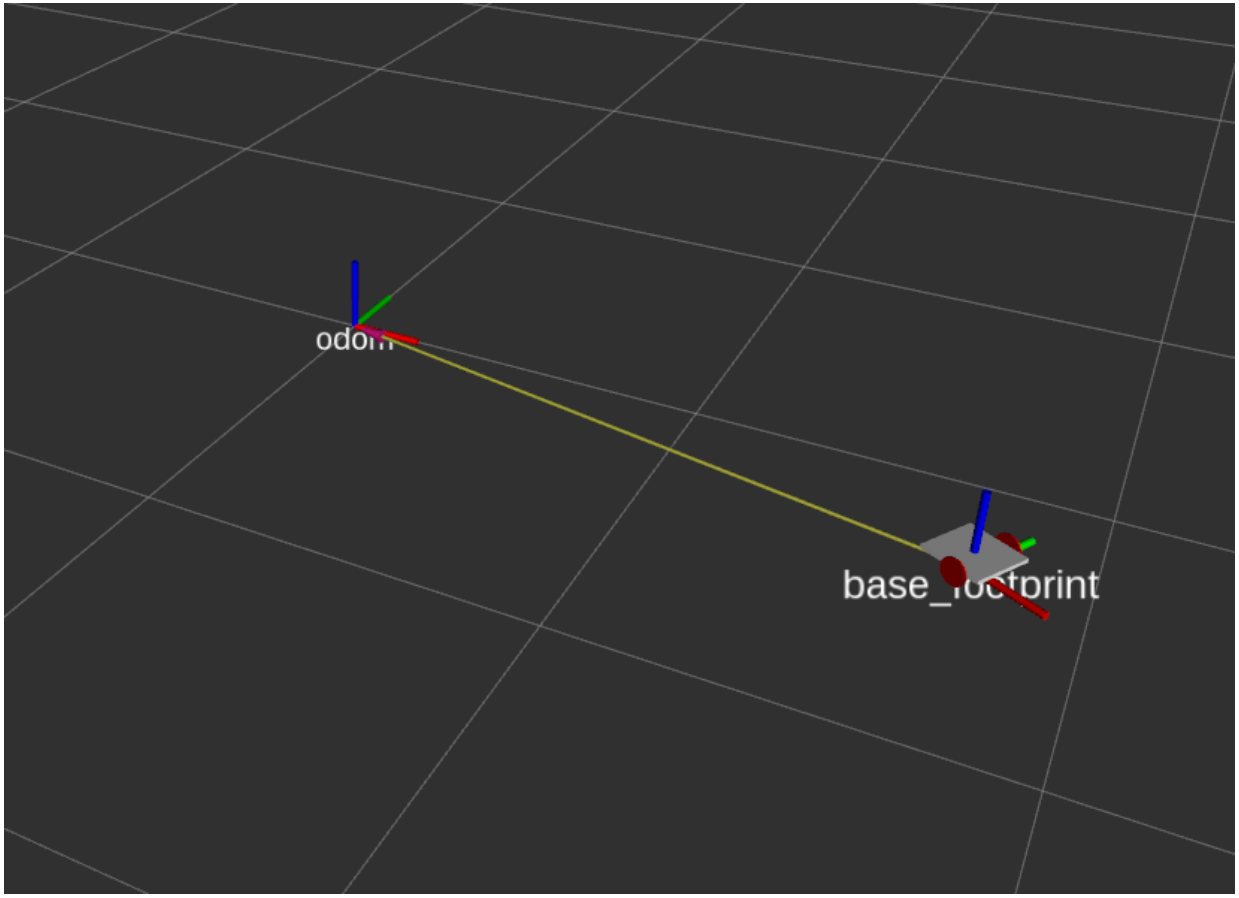
Rviz

- 3D visualization of the transformations



TF2 Transformation System : RViz Plugin





TF2: Transform Listener C++ API

- Create a TF listener to fill up a buffer. It will start listening immediately.
- Make sure, that the listener does not run out of scope!
- To look up transformations, use:

```
geometry_msgs::TransformStamped
transformStamped =
tfBuffer.lookupTransform(target_frame_id,
source_frame_id, time);
```

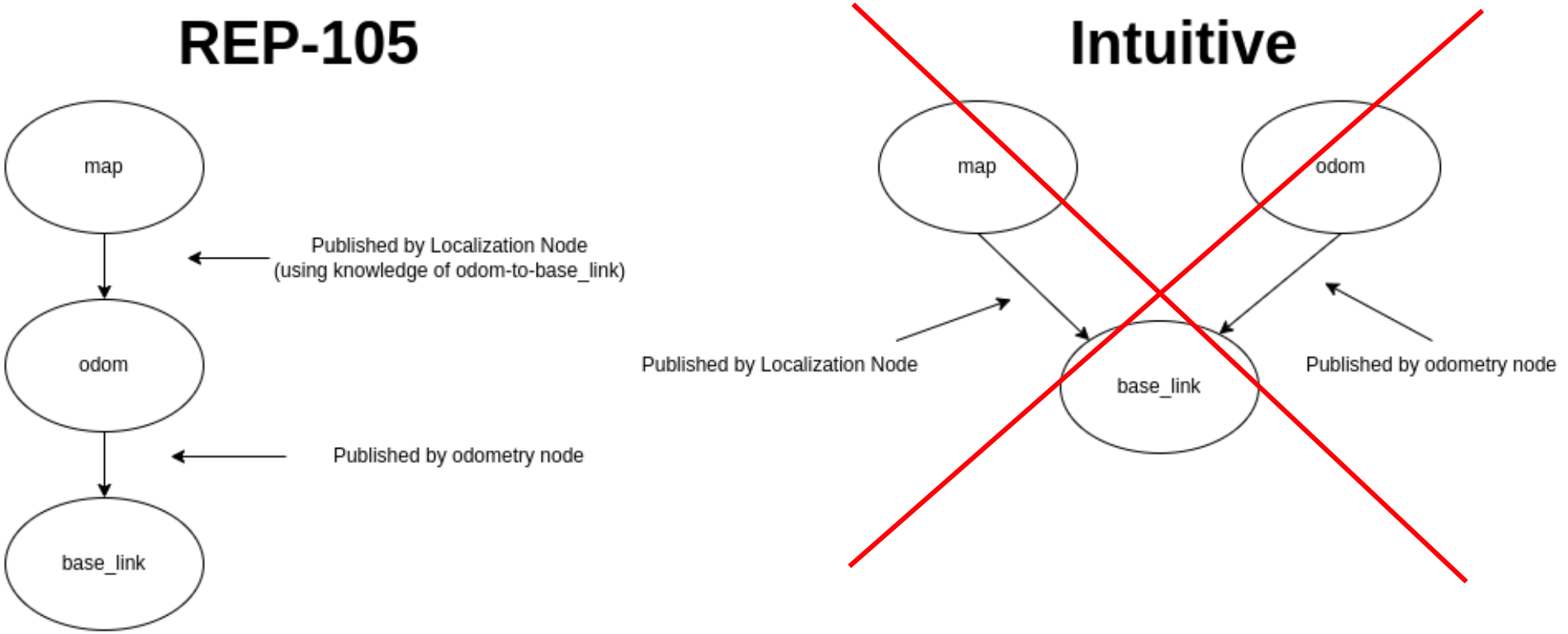
- For *time*, use `ros::Time(0)` to get the last available transform

```
#include <ros/ros.h>
#include <tf2_ros/transform_listener.h>
#include <geometry_msgs/TransformStamped.h>
#include <geometry_msgs/PointStamped.h>

int main(int argc, char** argv) {
  ros::init(argc, argv, "tf2_listener");
  ros::NodeHandle nodeHandle;

  tf2_ros::Buffer tfBuffer;
  tf2_ros::TransformListener tfListener(tfBuffer);

  ros::Rate rate(10.0);
  while (nodeHandle.ok()) {
    geometry_msgs::TransformStamped transfStamped =
tfBuffer.lookupTransform("base", "odom", ros::Time(0));

    geometry_msgs::PointStamped ptInOdom, ptInBase;
    ptInOdom.header.frame_id = "odom";
    ptInOdom.header.stamp = ros::Time::now();
    ptInOdom.point = {1.0, 2.0, 0.0};

    tf2::doTransform(ptInOdom, ptInBase, transfStamped);

    ROS_INFO("Point transformé : (%.2f, %.2f, %.2f)",
             pointInBase.point.x,
             pointInBase.point.y,
             pointInBase.point.z);
    rate.sleep();
  }
  return 0;
}
```

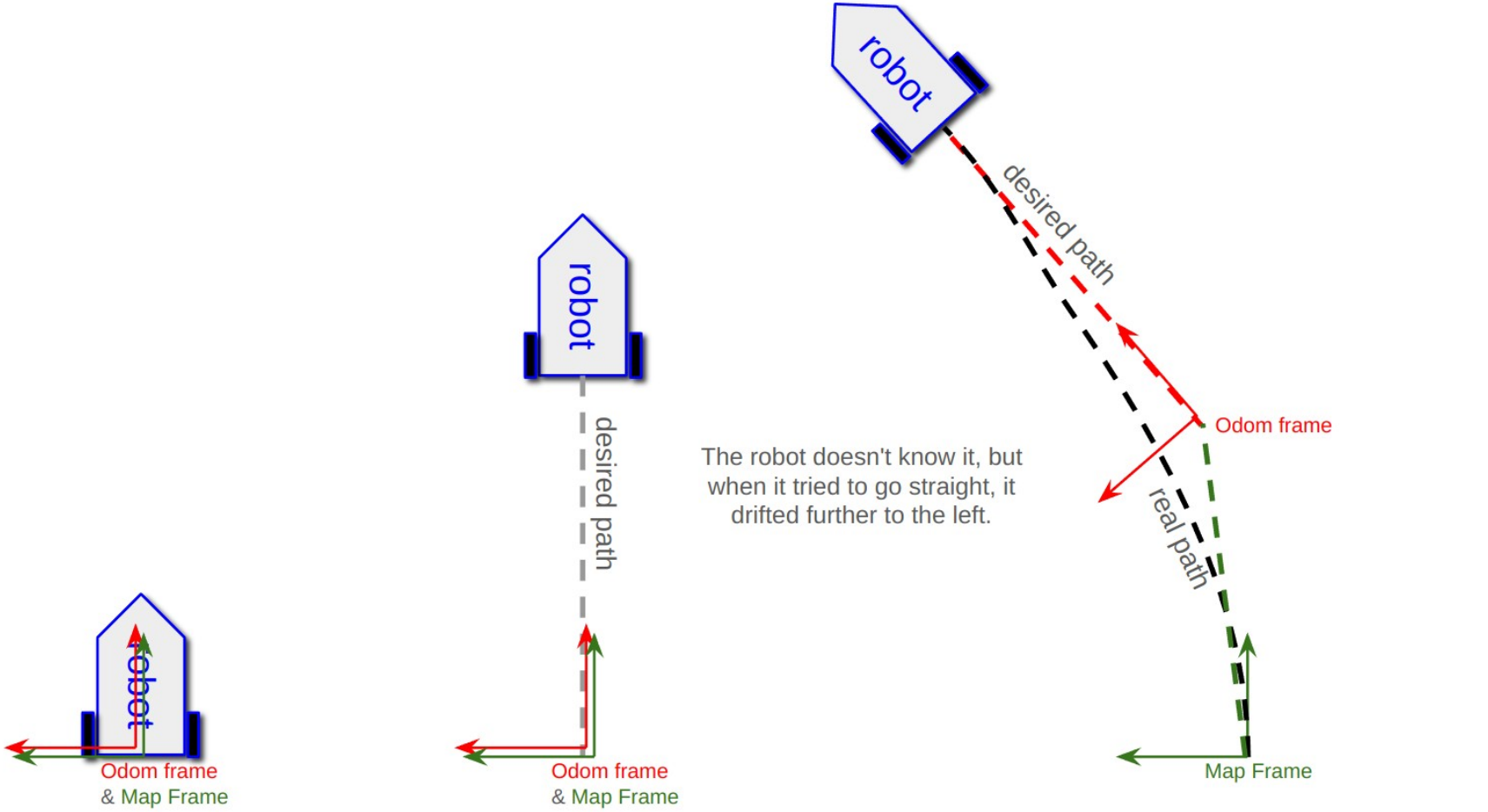
TF2 Transformation System : Conventions

- **base_link** : rigidly attached to the robot base.
- **odom** :
  - *odom* is a world-fixed frame, but starts from the robot's initial position
  - The pose can **drift** over time, without any limits.
  - Pose is guaranteed to be **continuous**, and **accurate** for a short period of time
  - It is computed based on an **odometry** source, such as wheel odometry, visual odometry or an inertial measurement unit.
  - **High frequency** and low latency
- **map** :
  - *map* is a world-fixed frame.
  - Map frame is not continuous, can change in **discrete jumps** at any time.
  - Typically, a localization component constantly re-computes the robot pose in the map frame based on sensor observations
  - Low frequency and high latency

TF2 Transformation System : Conventions

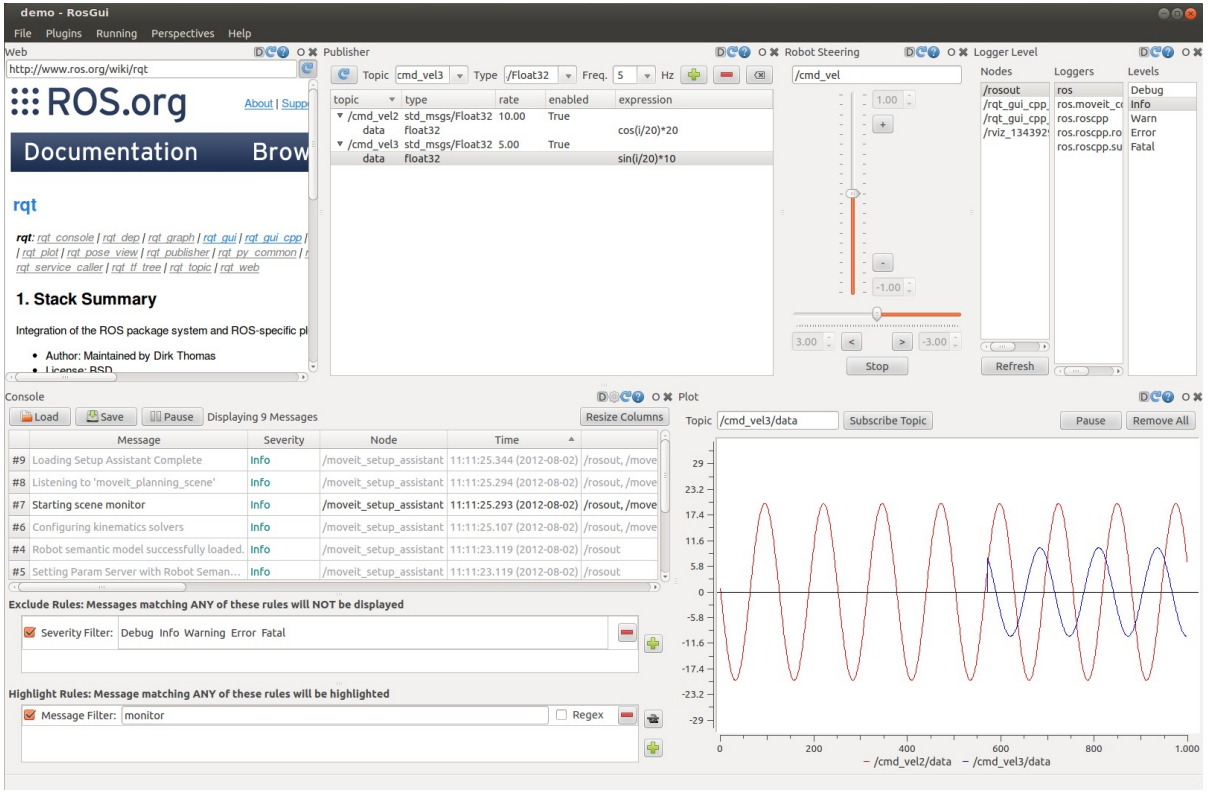


TF2 Transformation System : Conventions



rqt User Interface

- Qt based user interface
- Custom interfaces can be created,
- Many plugins available
- Easy to write your own plugins




```
> rosrun rqt_gui rqt_gui
```

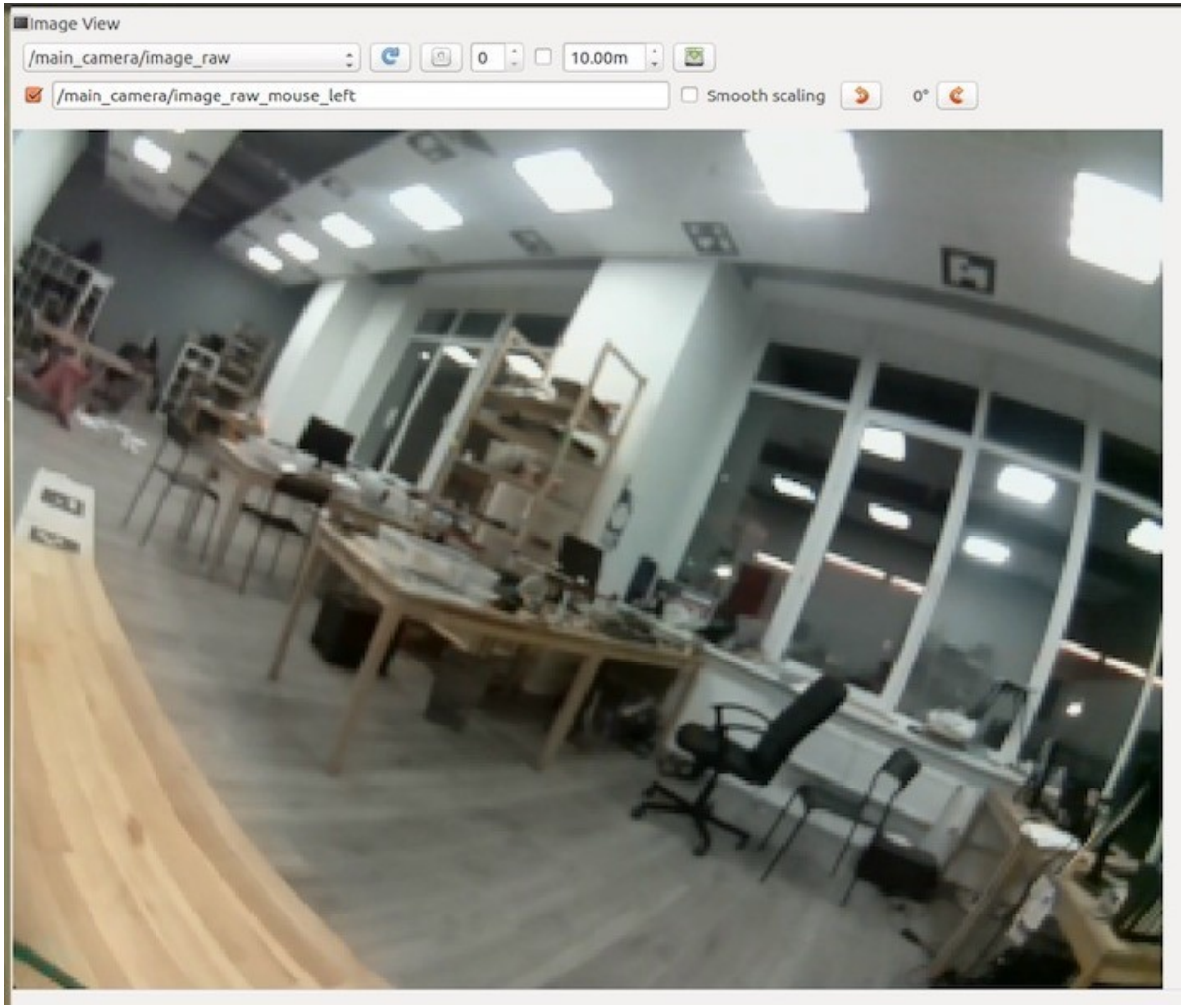
or

```
> rqt
```

rqt User Interface : rqt_image_view

- Visualizing images

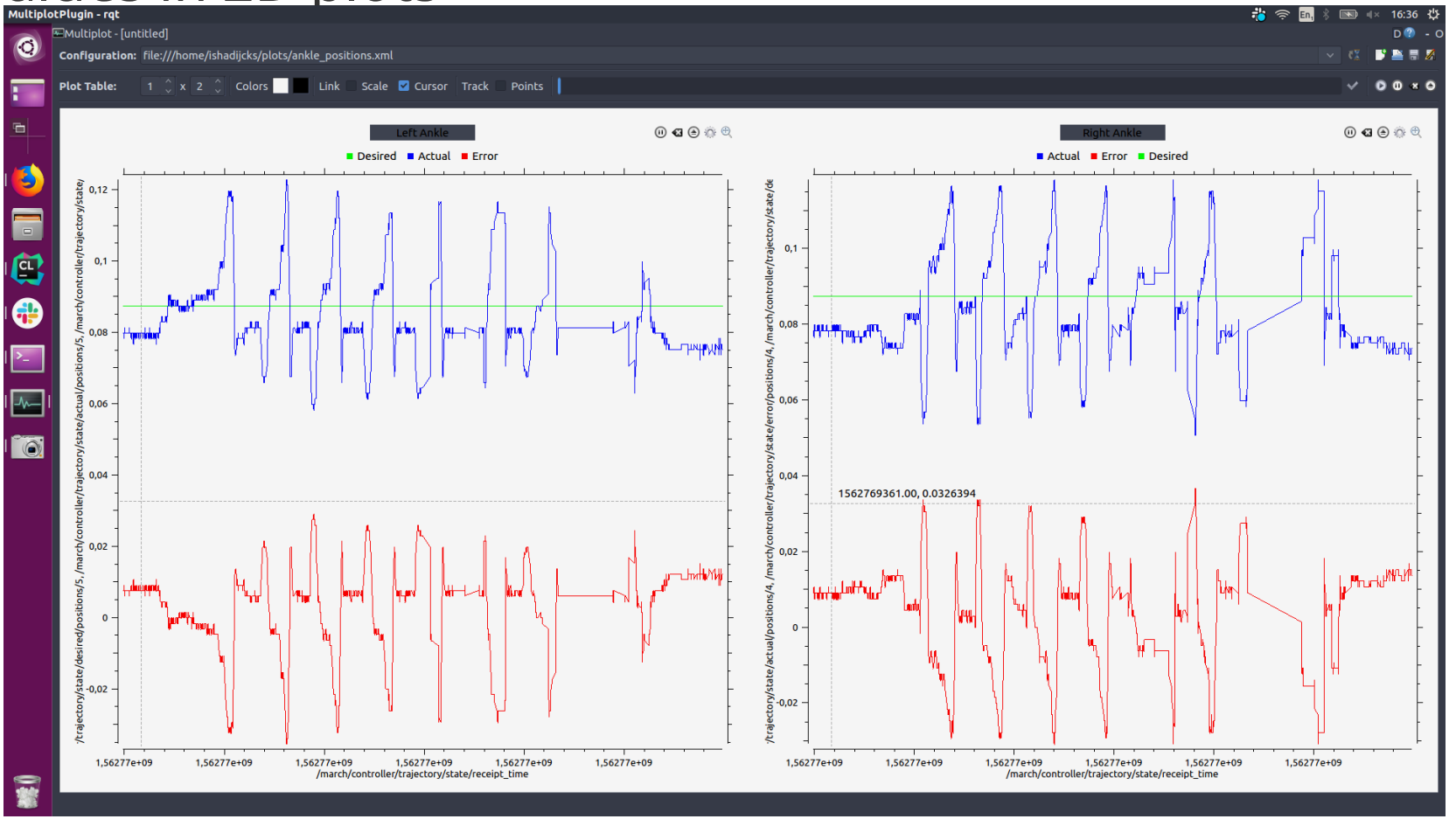
```
> rosrun rqt_image_view rqt_image_view
```



rqt User Interface: rqt_multiplot

- Visualizing numeric values in 2D plots

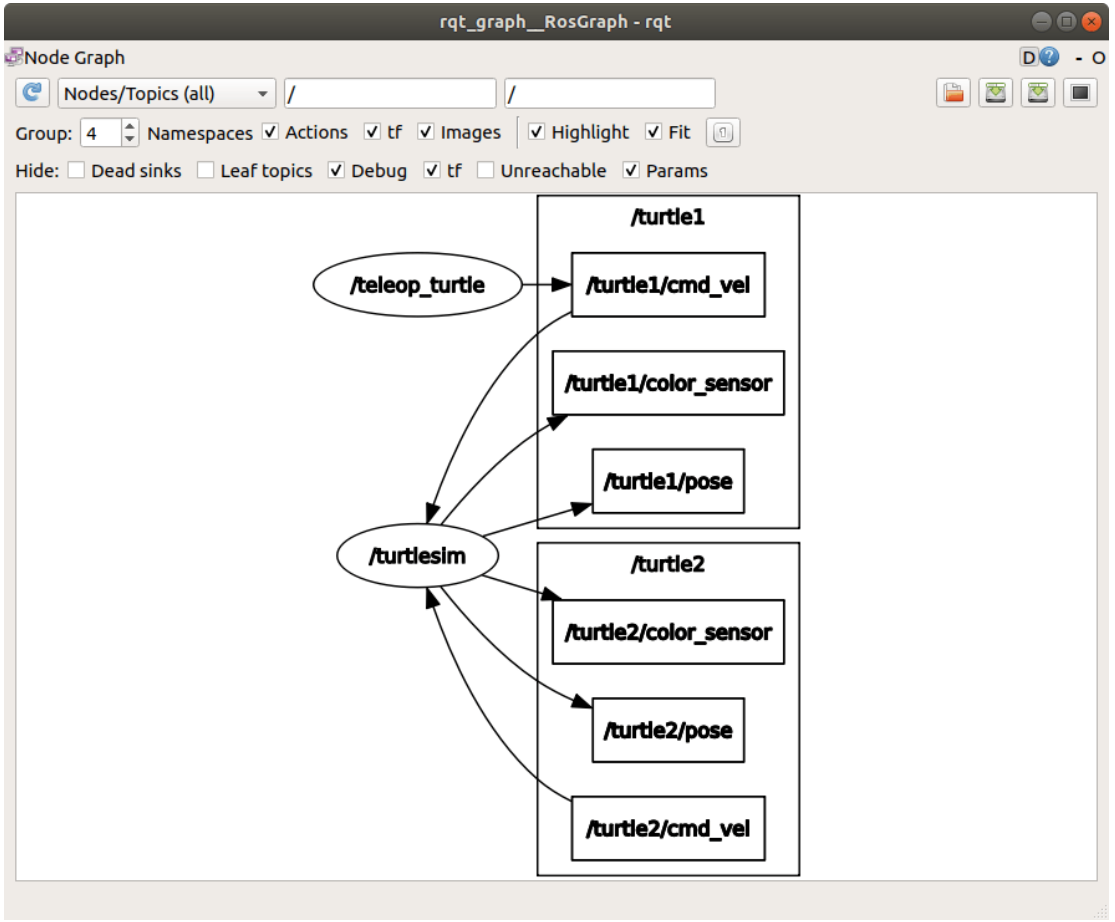
```
> rosrun rqt_multiplot rqt_multiplot
```



rqt User Interface: rqt_graph

- Visualizing the ROS computation graph

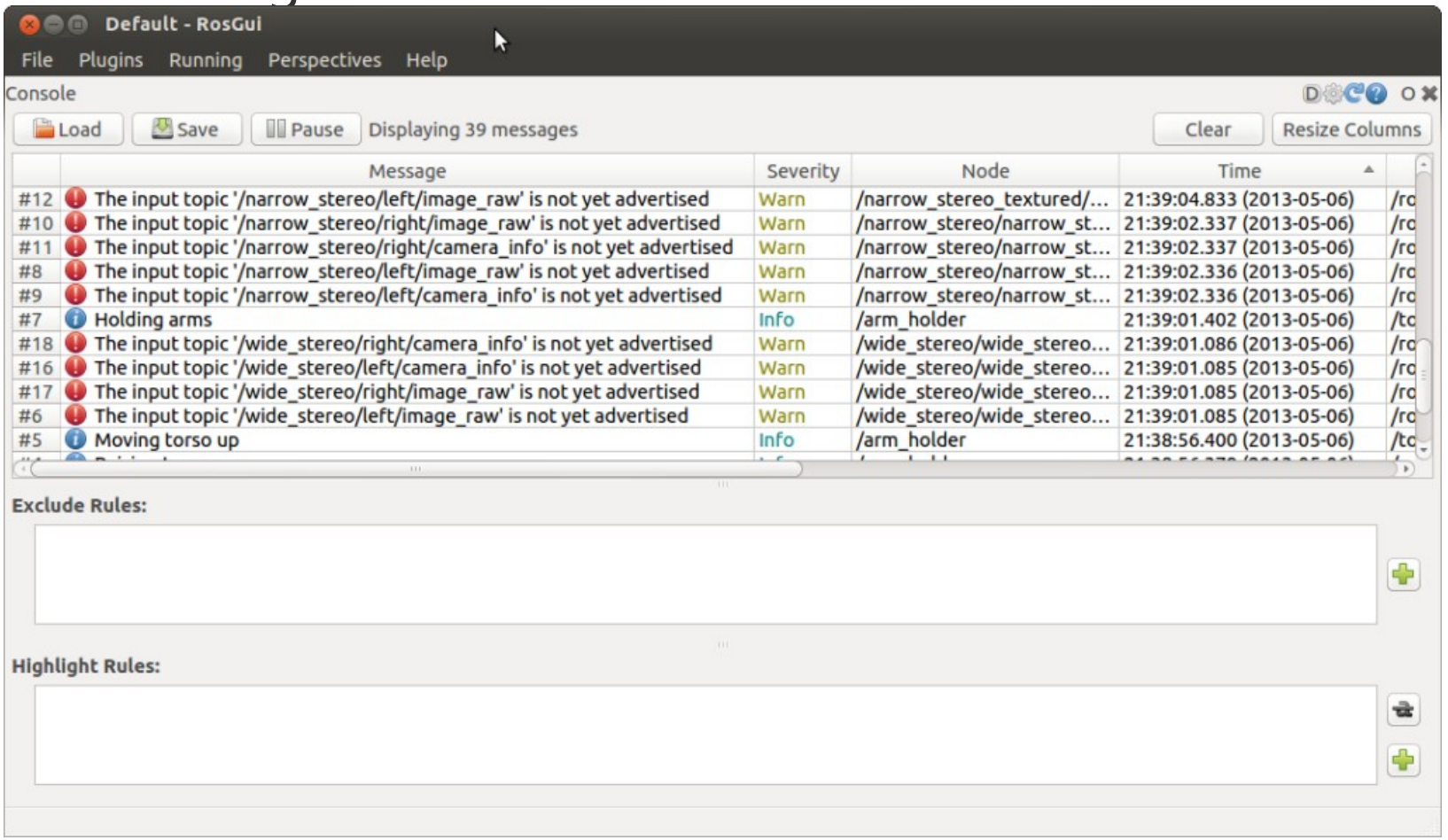
```
> rosrun rqt_graph rqt_graph
```



rqt User Interface: rqt_console

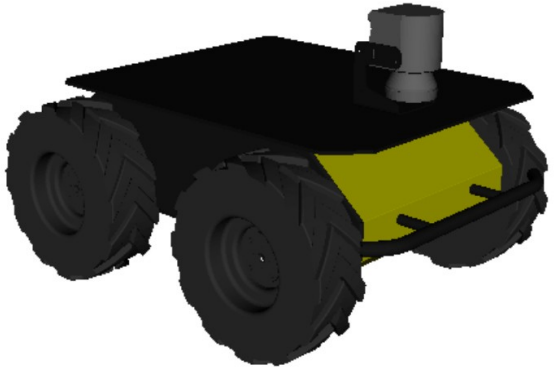
- Displaying and filtering ROS messages

```
> rosrun rqt_console rqt_console
```

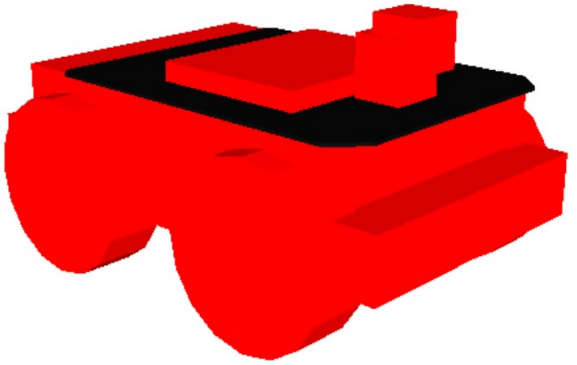


Robot Models : URDF

- URDF = **U**nified **R**obot **D**escription **F**ormat
- Defines an XML format for representing a robot model
  - Kinematic and dynamic description
  - Visual representation
  - Collision model
- URDF generation can be done by scripting with the XACRO language



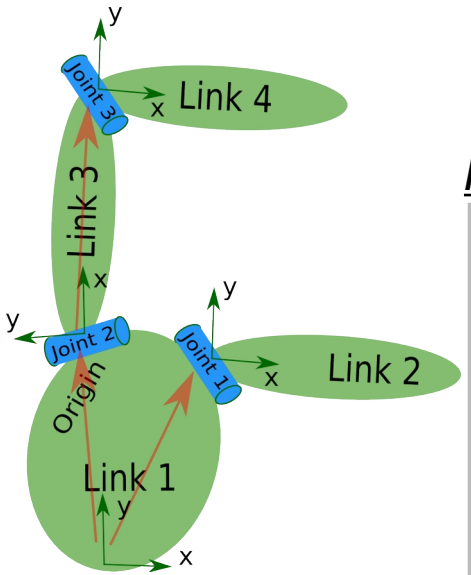
Mesh for visuals



Primitives for collision

Robot Models : URDF

- Description consists of a set of link elements and a set of joint elements
- Joints connect the links together



Robot.urdf

```
<robot name="robot">
  <link> ... </link>
  <link> ... </link>
  <link> ... </link>

  <joint> .... </joint>
  <joint> .... </joint>
  <joint> .... </joint>
</robot>
```

```
<link name="link_name">
  <visual>
    <geometry>
      <mesh filename="mesh.dae"/>
    </geometry>
  </visual>
  <collision>
    <geometry>
      <cylinder length="0.6" radius="0.2"/>
    </geometry>
  </collision>
  <inertial>
    <mass value="10"/>
    <inertia ixx="0.4" ixy="0.0" …/>
  </inertial>
</link>
```

```
<joint name="joint_name" type="revolute">
  <axis xyz="0 0 1"/>
  <limit effort="1000.0" upper="0.548" … />
  <origin rpy="0 0 0" xyz="0.2 0.01 0"/>
  <parent link="parent_link_name"/>
  <child link="child_link_name"/>
</joint>
```

Robot Models: Usage in ROS

- The robot description (URDF) is stored on the parameter server (typically) under **`/robot_description`**
- You can visualize the robot model in RViz using the ***RobotModel*** plugin
- In *description.launch*, we use xacro. **Xacro** is a simple scripting language that makes it easy to create a URDF file.
- **Xacro** allows you to use constants, mathematical functions or macros.

spawn_husky.launch

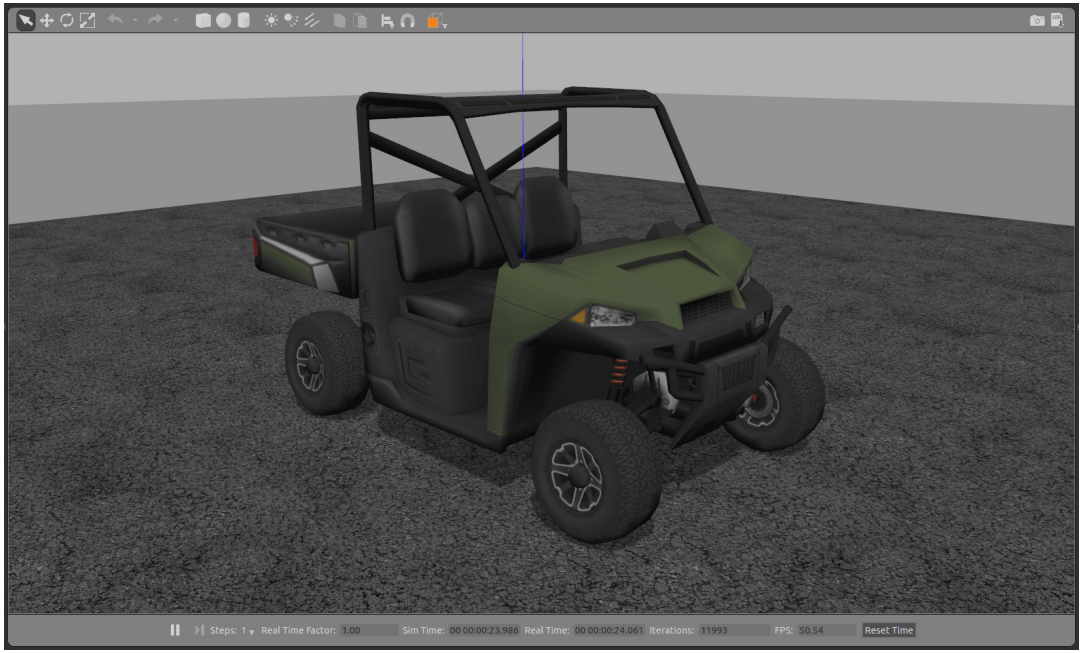
```
…
<include file="$(find
husky_description)/launch/description.launch" >
  <arg name="robot_namespace" value="$(arg robot_namespace)"/>
  <arg name="laser_enabled" default="$(arg laser_enabled)"/>
  <arg name="kinect_enabled" default="$(arg kinect_enabled)"/>
  <arg name="urdf_extras" default="$(arg urdf_extras)"/>
</include>
…
```

description.launch

```
…
<param name="robot_description" command="$(find xacro)/xacro
'$(find husky_description)/urdf/husky.urdf.xacro'
  --inorder
  robot_namespace:=$(arg robot_namespace)
  laser_enabled:=$(arg laser_enabled)
  kinect_enabled:=$(arg kinect_enabled)
  urdf_extras:=$(arg urdf_extras)" />
…
```

Simulation Description Format (SDF)

- Defines an XML format for describing:
  - Environments (lighting, gravity etc.)
  - Objects (static and dynamic)
  - Sensors
  - Robots
- SDF is the default format for Gazebo
- Gazebo will automatically convert a URDF to SDF.



Further References

- **Site du cours :** https://perso.ensta-paris.fr/~battesti/website/teachings/rob314/
- **ROS Wiki:**
  - https://wiki.ros.org/
- **Installation:**
  - https://wiki.ros.org/ROS/Installation
- **Tutorials:**
  - https://wiki.ros.org/ROS/Tutorials
- **Available packages:**
  - https://index.ros.org/packages/#melodic
- **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