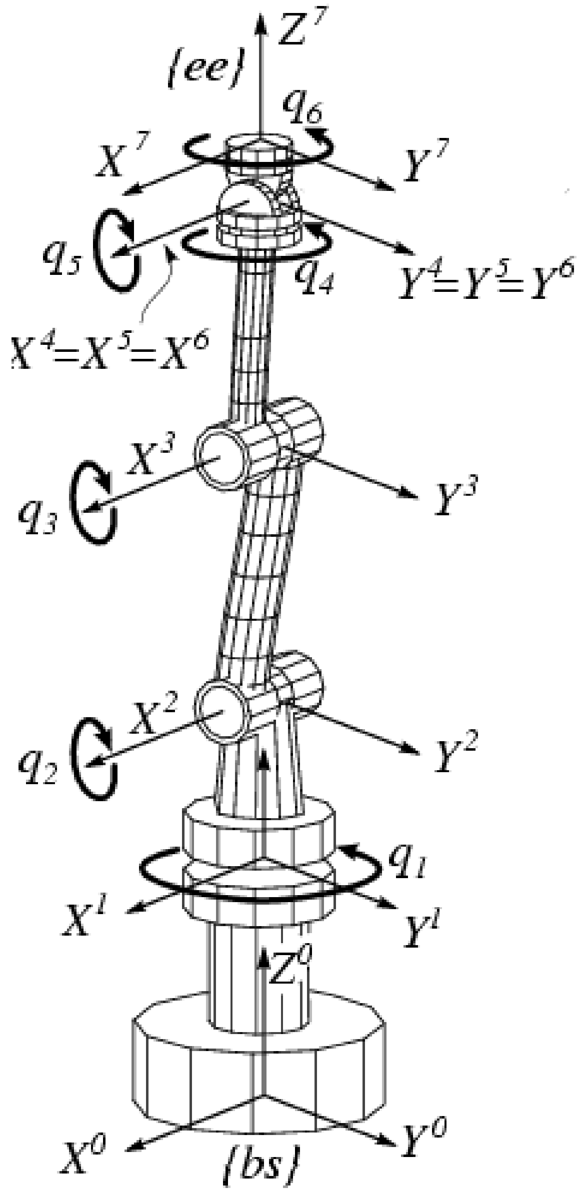
Module B – Geometric modeling in ROS

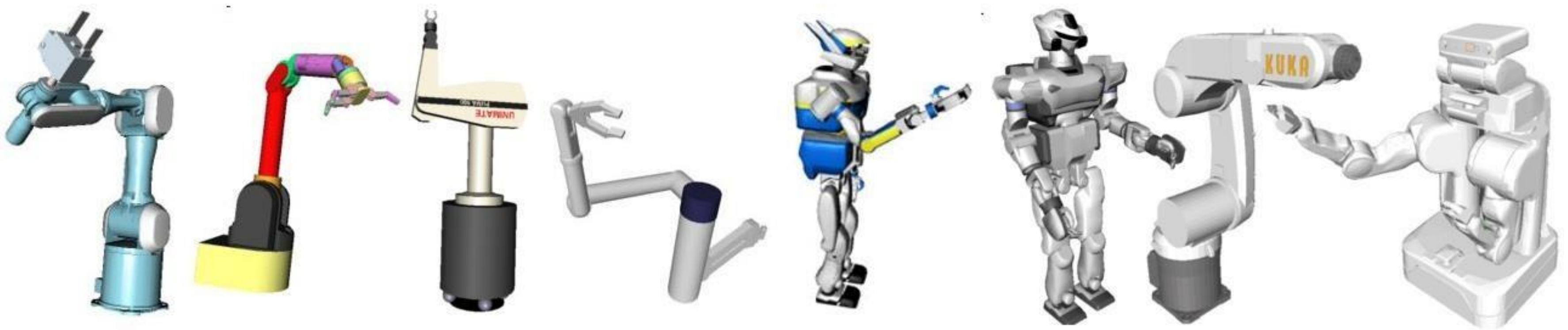
Robot Programming and Control
Accademic Year 2021-2022

Diego Dall'Alba diego.dallalba@univr.it
Department of Computer Science – University of Verona
Altair Robotics Lab

Geometric modelling in ROS

- Robotic applications modelling
  - Kinematics modeling: URDF
  - Rigid transformation: Tf2
- 3D Visualization tools
  - RViz
  - Visualization vs Simulation





Kinematic modelling in ROS: URDF+Xacro

Unified Robot Description Format (**URDF**) is an XML format for representing a robot model.

It enable to describe kinematic, visual and dynamic properties of a manipulator.

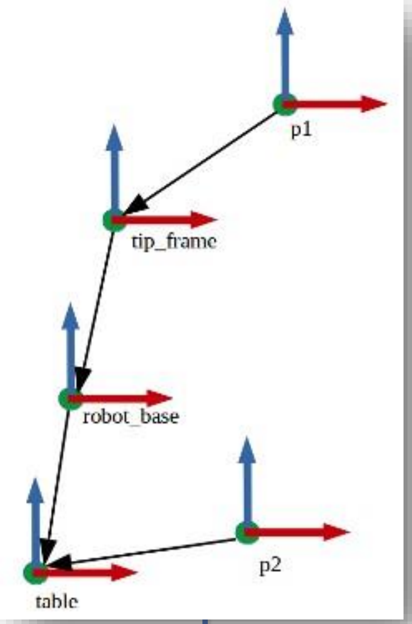
http://wiki.ros.org/urdf

Xacro is an XML macro language: enable construction of shorter and more readable XML files by using macros that expand to larger XML expressions.

http://wiki.ros.org/xacro

ROS provides parsing tools for reading and checking URDF files:

http://wiki.ros.org/urdf/Tutorials

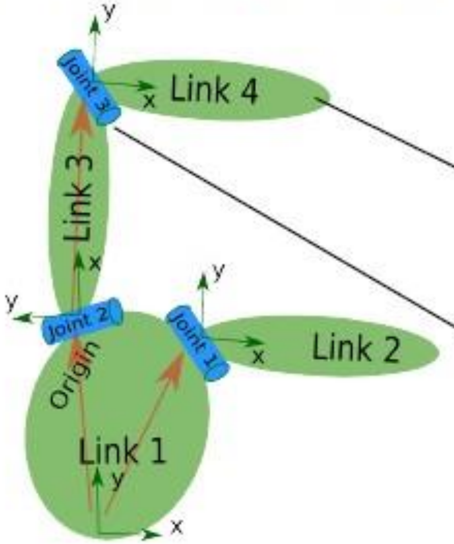



```
<robot name="test_robot">
  <link name="link1" />
  <link name="link2" />
  <link name="link3" />
  <link name="link4" />

  <joint name="joint1" type="continuous">
    <parent link="link1"/>
    <child link="link2"/>
  </joint>
  <joint name="joint2" type="continuous">
    <parent link="link1"/>
    <child link="link3"/>
  </joint>
  <joint name="joint3" type="continuous">
    <parent link="link3"/>
    <child link="link4"/>
  </joint>
</robot>
```

URDF Simple Example

- 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>
```

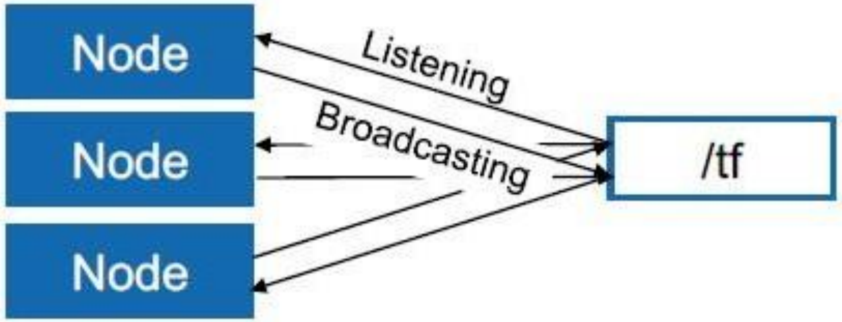
```
<link name="link_name">
  <visual>
    <geometry>
      <mesh filename="mesh.dae"/>
    </geometry>
  </visual>
  <collision>
    <geometry>
      <cylinder length="0.6" radius="0.2"/>
    </geometry>
  </collision>
  <inertial>
    <mass value="10"/>
    <inertia ixx="0.4" ixy="0.0" …/>
  </inertial>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

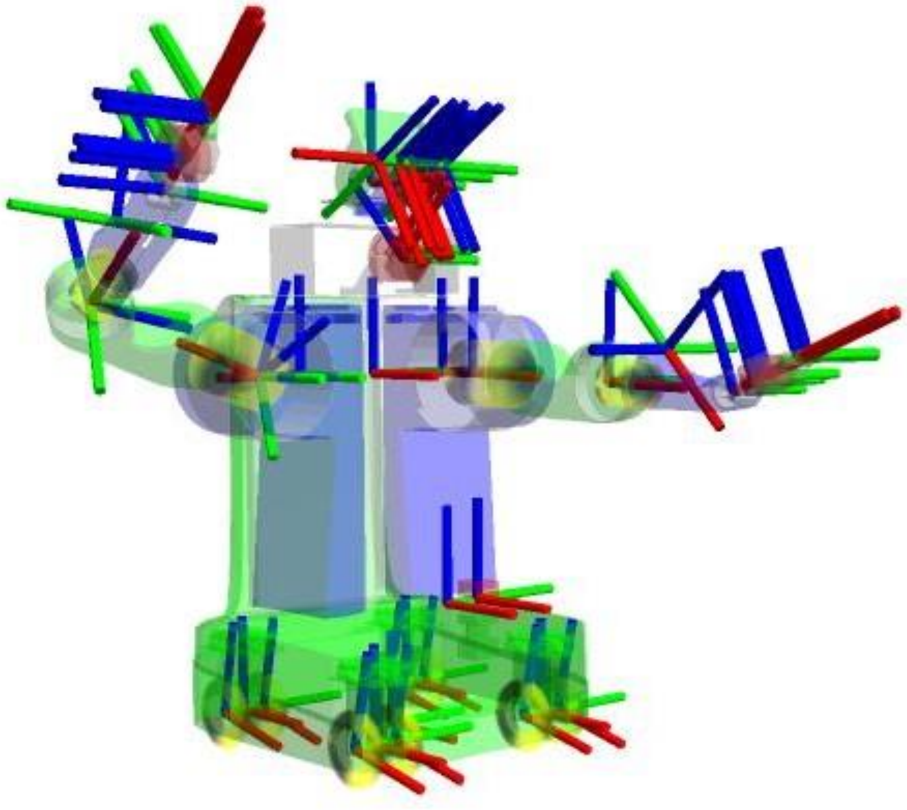
```
<joint name="joint_name" type="revolute">
  <axis xyz="0 0 1"/>
  <limit effort="1000.0" upper="0.548" … />
  <origin rpy="0 0 0" xyz="0.2 0.01 0"/>
  <parent link="parent_link_name"/>
  <child link="child_link_name"/>
</joint>
```

More info
http://wiki.ros.org/urdf/XML/model

TF Transformation System

- Tool for keeping track of coordinate frames over time
- Maintains relationship between coordinate frames in a tree structure buffered in time
- Lets the user transform points, vectors, etc. between coordinate frames at desired time
- Implemented as publisher/subscriber model on the topics `/tf` and `/tf_static`





More info
http://wiki.ros.org/tf2

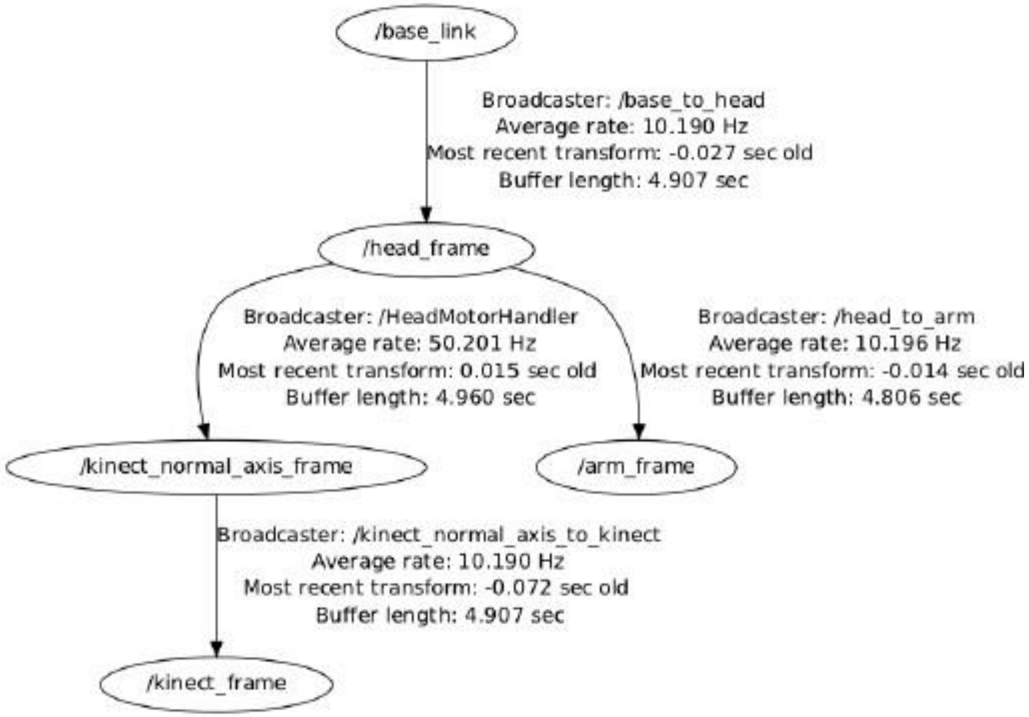
TF Transformation System

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
```



TF Transformation System

Transform Listener C++ API

- Create a TF listener to fill up a buffer

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

- Make sure, that the `listener` does not run out of scope!
- To lookup transformations, use

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

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

```
#include <ros/ros.h>
#include <tf2_ros/transform_listener.h>
#include <geometry_msgs/TransformStamped.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 transformStamped;
    try {
      transformStamped = tfBuffer.lookupTransform("base",
                                   "odom", ros::Time(0));
    } catch (tf2::TransformException &exception) {
      ROS_WARN("%s", exception.what());
      ros::Duration(1.0).sleep();
      continue;
    }
    rate.sleep();
  }
  return 0;
};
```

TF Transformation System: Tools

Command line

Print information about the current transform tree

```
> rosrun tf tf_monitor
```

Print information about the transform between two frames

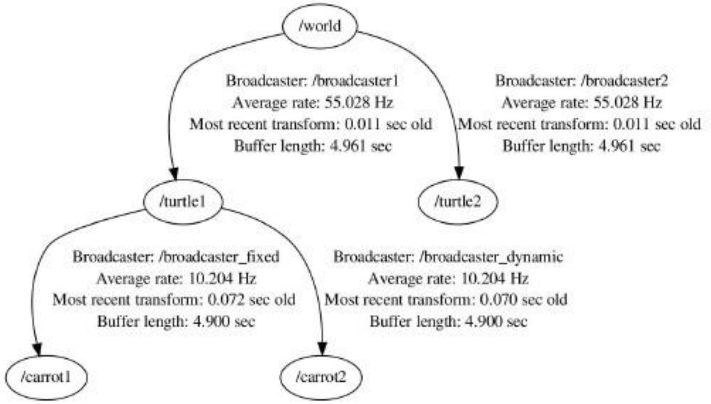
```
> rosrun tf tf_echo
    source_frame target_frame
```

View Frames

Creates a visual graph (PDF) of the transform tree. Broken at the moment!!!!!
https://github.com/ros/geometry/pull/222

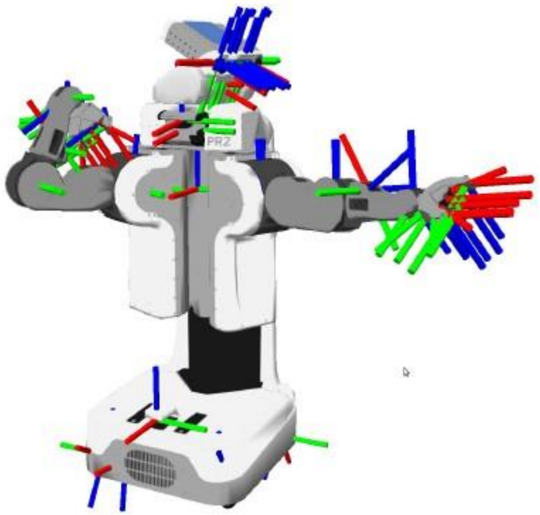
```
rosrun tf view_frames
```

```
rosrun tf2_tools view_frames.py
```



RViz

3D visualization of the transforms

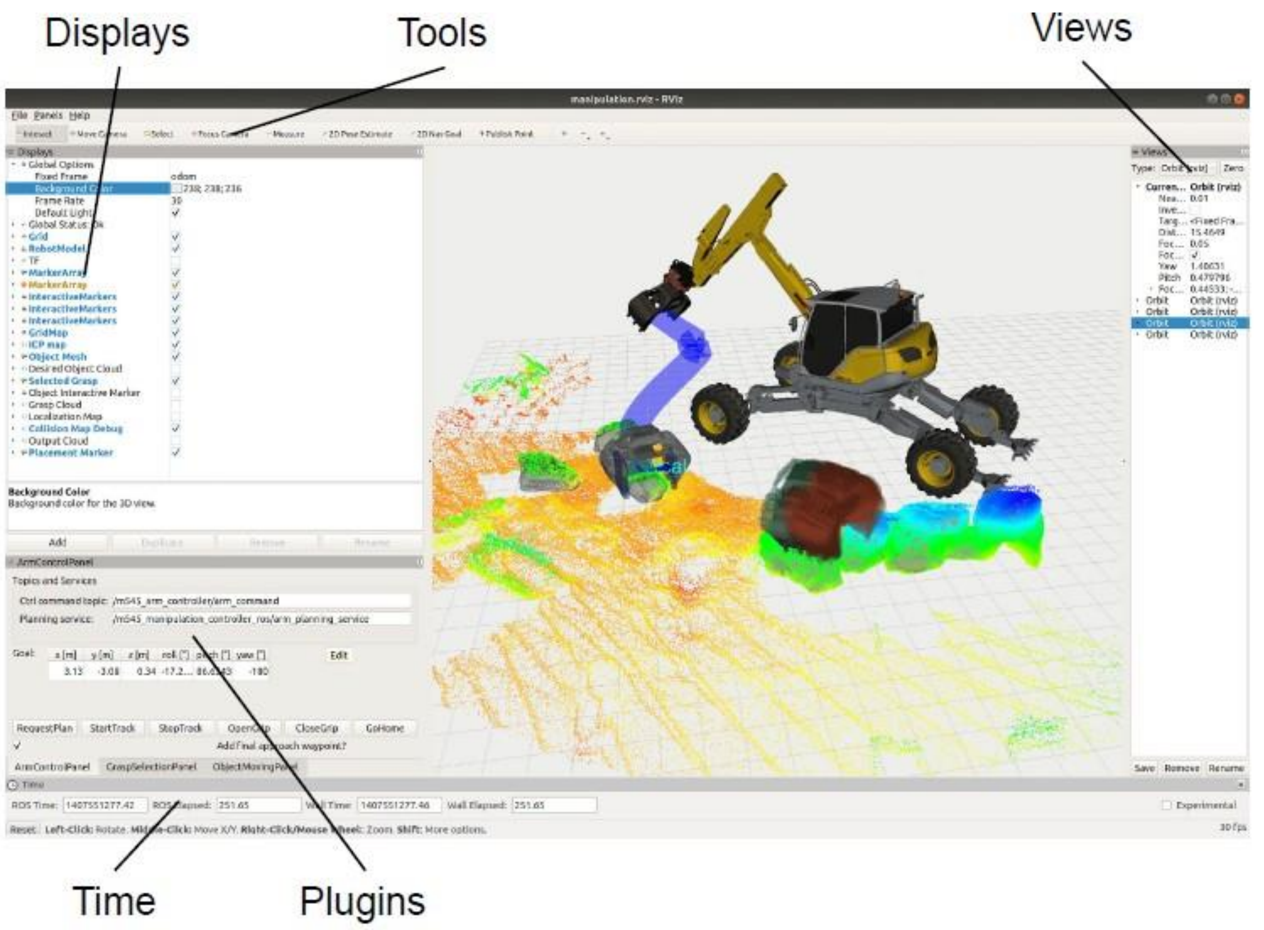


RViz

- 3D visualization tool for ROS
- Subscribes to topics and visualizes the message contents
- Different camera views (orthographic, top-down, etc.)
- Interactive tools to publish user information
- Save and load setup as RViz configuration
- Extensible with plugins

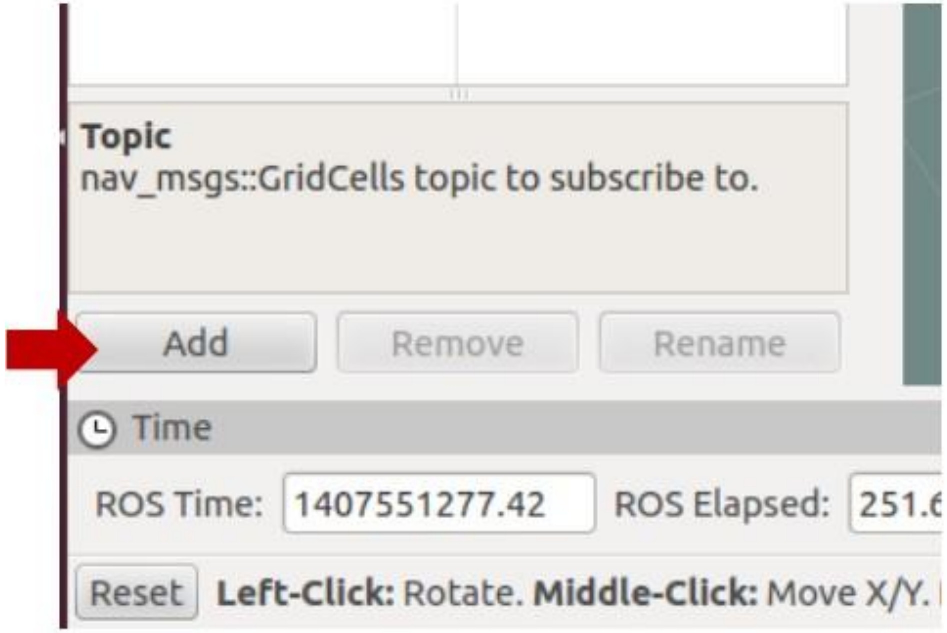
Run RViz with

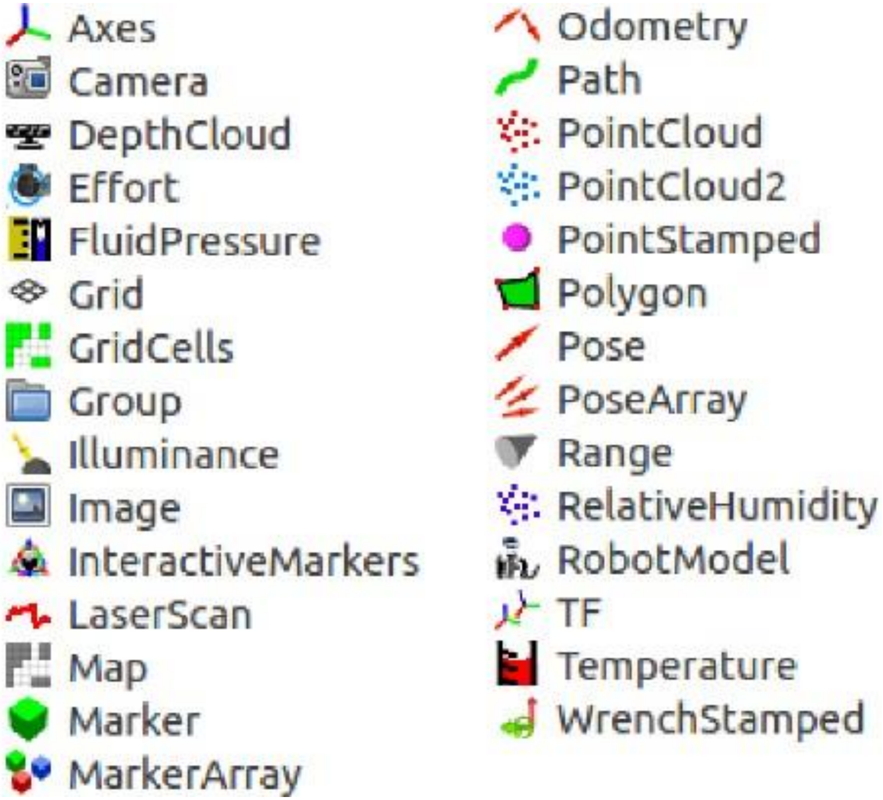
```
> rviz
```



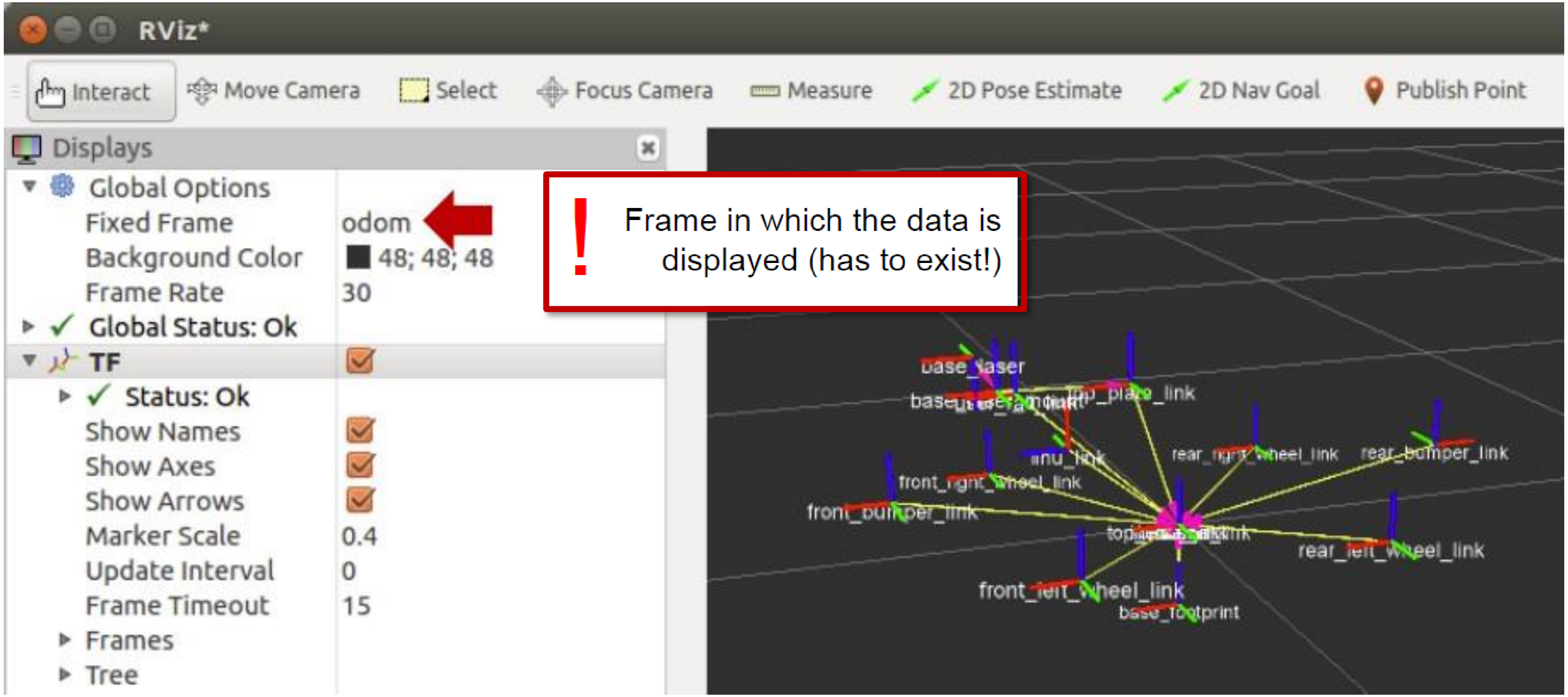
More info
wiki.ros.org/rviz

RViz Display plugin





Rviz: TF Transformation System



Simulation environments in ROS

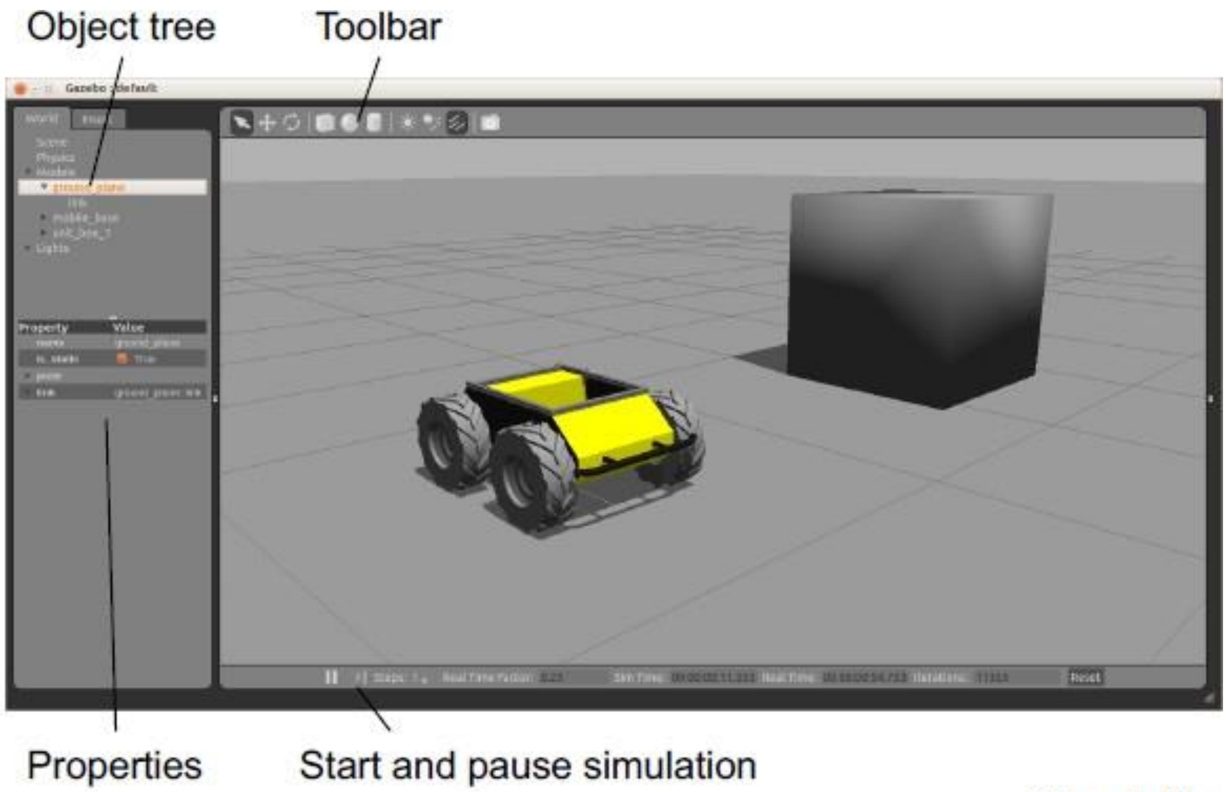
- Rviz a complex 3D visualizer, fundamental for debugging and better understanding
- It could also «animate» robotic kinematic chain (URDF models)
- Sometimes a more complete simulation is needed, including the behaviour of robots
- Gazebo is the default simulator used in ROS framework, maintained as a separate project from OSRF.
- V-REP is a robotic simulators developed by Coppelia Robotics
- It is a commercial software, that can be obtained for free in its educational version.

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

Run Gazebo with

```
> rosrun gazebo_ros gazebo
```



More info

http://gazebosim.org/

http://gazebosim.org/tutorials

~~V-rep Simulator~~ CoppeliaSim

- V-REP has support for Windows, Linux and Mac operating systems.
- It is possible to use 7 different programming languages with V-REP, the default language being Lua.
- V-REP doesn’t have a native ROS node for it.
- This means that it is not yet possible to run it as a part of a ROS system in a single launchfile, but instead alongside it, in another Linux terminal.
- On the other hand, V-REP does offer a default ROS plugin that can be used in VREP Lua scripts for creating ROS publishers and subscribers..

Comparison of (main) Simulation Environments for ROS

	V-REP	GAZEBO	ARGoS Large-scale robot simulations
Physics Engines	Bullet Physics Library, Open Dynamics Engine, Newton Dynamics, Vortex by CM Labs	Open Dynamics Engine	Custom 2D and 3D engines
Languages	Lua, C++, ROS, RemoteAPI	C++, ROS	Lua, C++, ROS
Threads	Spawned automatically	Two (simulator + interface)	Set by user
3D meshes	Importing, manipulation, materials	Importing, but no editing	No importing, OpenGL only
Object library	A lot of robots and other objects	A fair number of robots and other objects	A limited number of robots
Documentation	Extensive, a lot of code examples	Fairly comprehensive, some non-working code examples	Good quality but rather limited

Rich Neutral Poor simulator characteristics

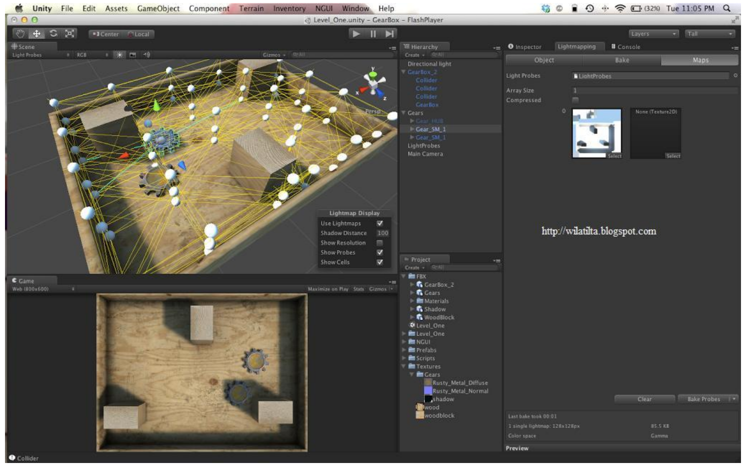
Simulation Scene description example: Simulation Description format (SDF)

- Defines an XML format to describe
  - Environments (lighting, gravity etc.)
  - Objects (static and dynamic)
  - Sensors
  - Robots
- SDF is the standard format for Gazebo
- Gazebo converts a URDF to SDF automatically

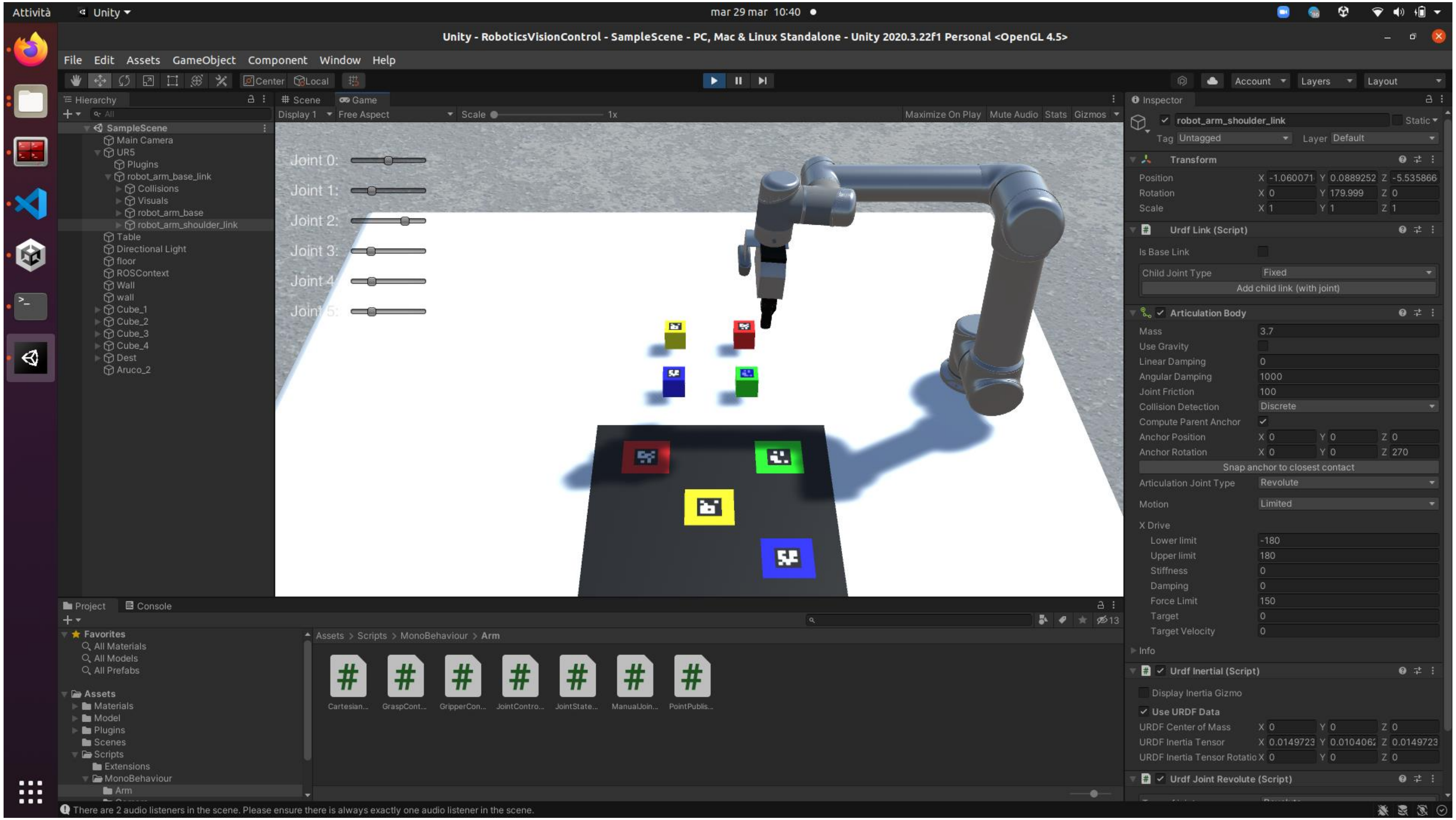
More info
http://sdformat.org

Unity3D

- The Unity game engine launched in 2005, aiming to "democratize" game development by making it accessible to more developers.
- Unity gives users the ability to create games and experiences in both 2D and 3D,
- the engine offers a primary scripting API in C#, for both the Unity editor in the form of plugins, and games themselves, as well as drag and drop functionality.
- Unity is a cross-platform engine
- The Unity editor is supported on Windows and macOS, with a version of the editor available for the Linux platform, albeit in an experimental stage
- While the engine itself currently supports building games for more than 25 different platforms, including mobile, desktop, consoles, and virtual reality
- As of 2018, Unity has been used to create approximately half of the new mobile games on the market and 60 percent of augmented reality and virtual reality content.



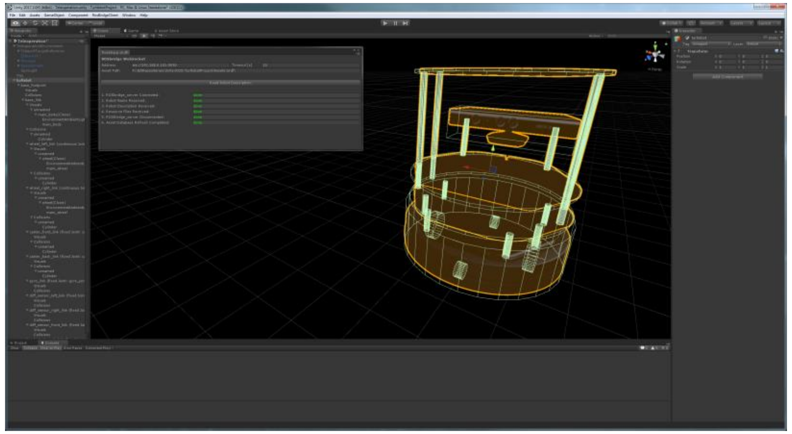
Unity3D Editor Interface

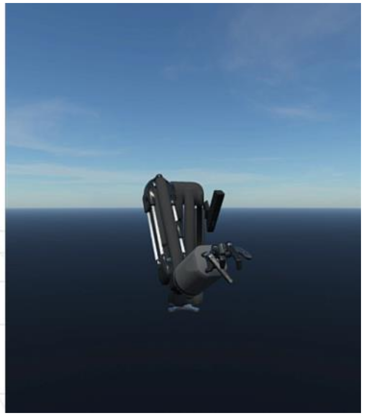


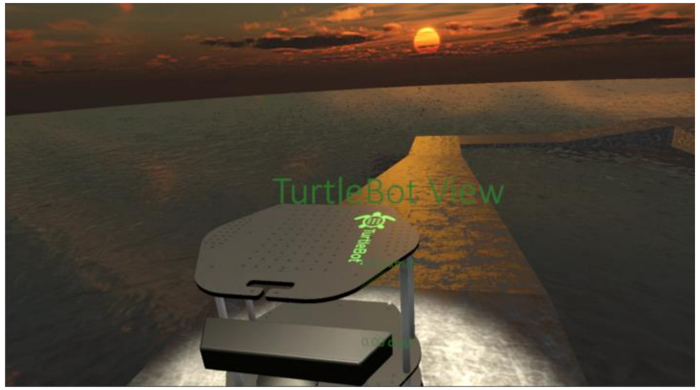
Unity + ROS

- communicate with ROS from within your Unity Application, including subscribe and publish topics, call and advertise services, set and get parameters
- import your robot's URDF model
- control your real Robot via Unity
- visualize your robot's actual state and sensor data in Unity
- simulate your robot in Unity with the data provided by the URDF and without using a connection to ROS. Beside visual components as meshes and textures, also joint parameters and masses, centers of mass, inertia and collider specifications of rigid bodies are imported and used for the physical simulation in Unity
- train neural networks e.g. with Unity's ML Agents

https://github.com/Unity-Technologies/Unity-Robotics-Hub



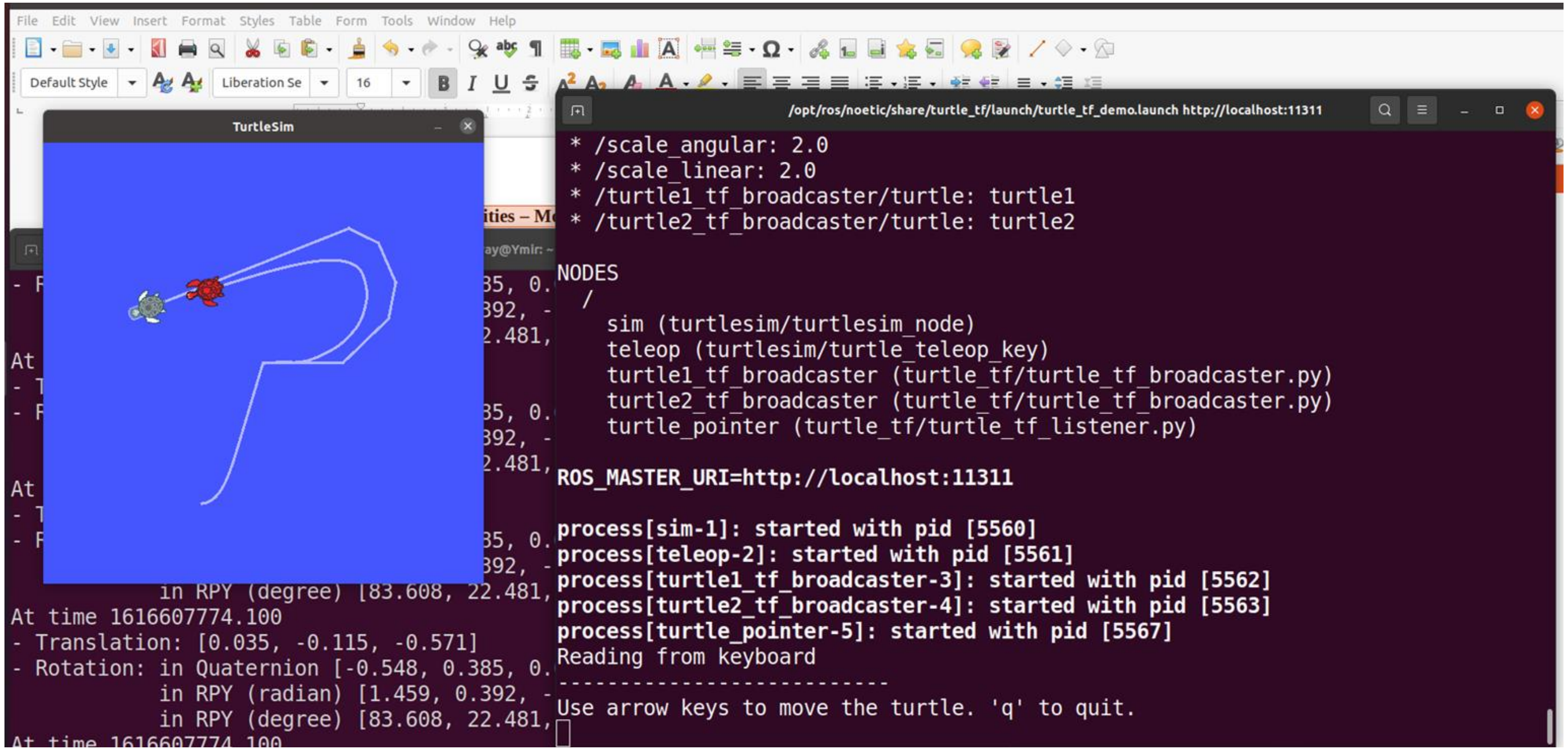




Exercise 1 - Working with Transformation in ROS: Tf (1)

- Use turtle_tf example to better understand how Tf are working:

```
roslaunch turtle_tf turtle_tf_demo.launch
```



Exercise 1 - Working with Transformation in ROS: Tf (2)

1. Visualize the Tf tree: rosrun tf2_tools view_frames.py
2. Print the absolute position and orientation of the first turtle
3. Print the relative transformation between the two turtles
4. Understand how the demo is implemented, how Tf are used in the listener side
    - modify the turtle2 behavior to keep a fixed distance from the turtle1
5. Use RViz to obtain a 3D visualization of the map, in particular familiarize with:
    - general interface and input mapping
    - the camera settings,
    - customization of the scene view
    - Tf plugin

Exercise 2 – A real URDF example: e.Do model (1)

- e.DO is a 6-axis articulated robot based on open-source hardware architecture and software.
- It incorporates DC motor movement, composite plastic casings and a base unit with integrated control logic and memory.
- The robot's flexible, modular structure even permits personalised configurations
- Each motorised unit has its own autonomous mechanical and electronic controller that can be configured to meet the operator's needs.

Get more info and check specs on:

https://edo.cloud/en/edo-robot/edo-robot-specs/

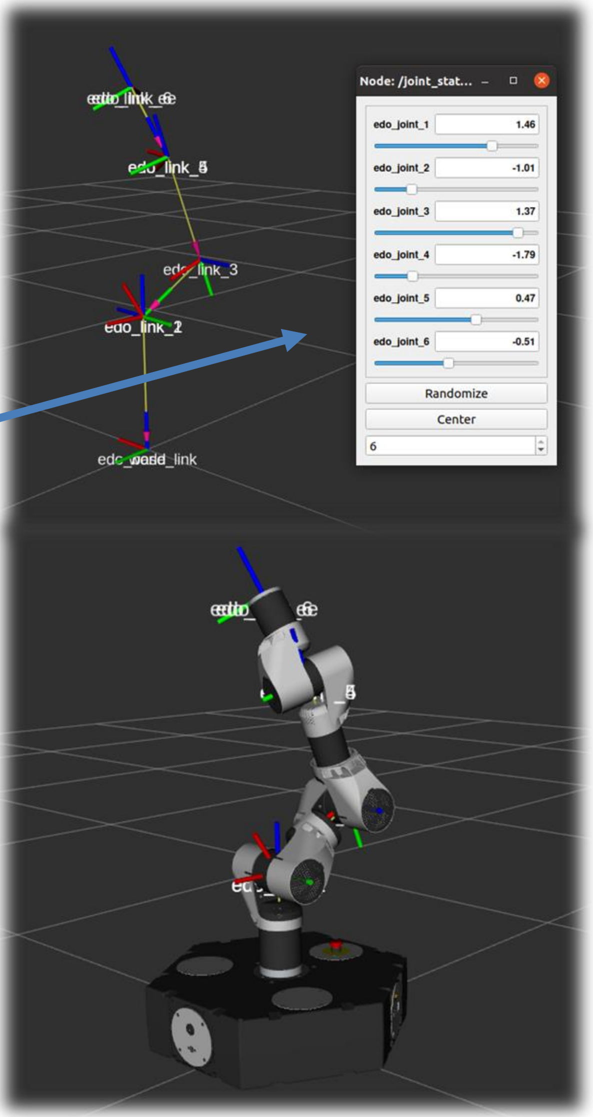
Exercise 2 – A real URDF example: e.Do model (2)

- Add the following package to your workspace: https://github.com/Pro/eDO_description
- Visualize the URDF in Rviz

```
roslaunch edo_description edo_upload.launch
roslaunch edo_description test.launch
```

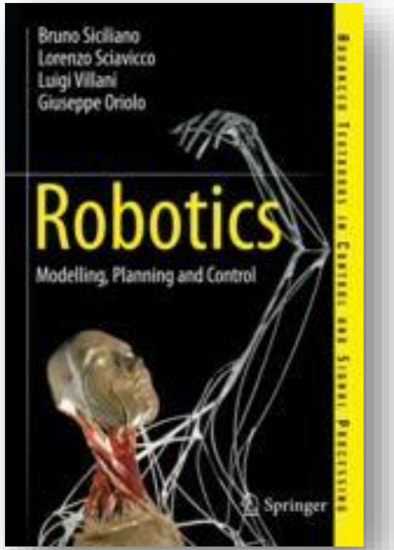
- Move around the manipulator with the joint slide panel
- Understand how the URDF is implemented and how Xacro is used (open the urdf file with a text editor)
- Visualize the Tf tree and understand the different frames relationship

Homework: Create a URDF with the e.Do manipulator positioned in the middle of a table, modeled as a cube with side dimension 1 meter. Add a cylindrical EE with radius 20mm and length 100mm.



Questions?

The contents of these slides are partially based on:



Other online sources

Programming for Robotics - Introduction to ROS

February 2017

DOI: 10.13140/RG.2.2.14140.44161

Affiliation: Robotics Systems Lab, ETH Zurich

Péter Fankhauser · Dominic Jud · Martin Wermelinger · Marco Hutter