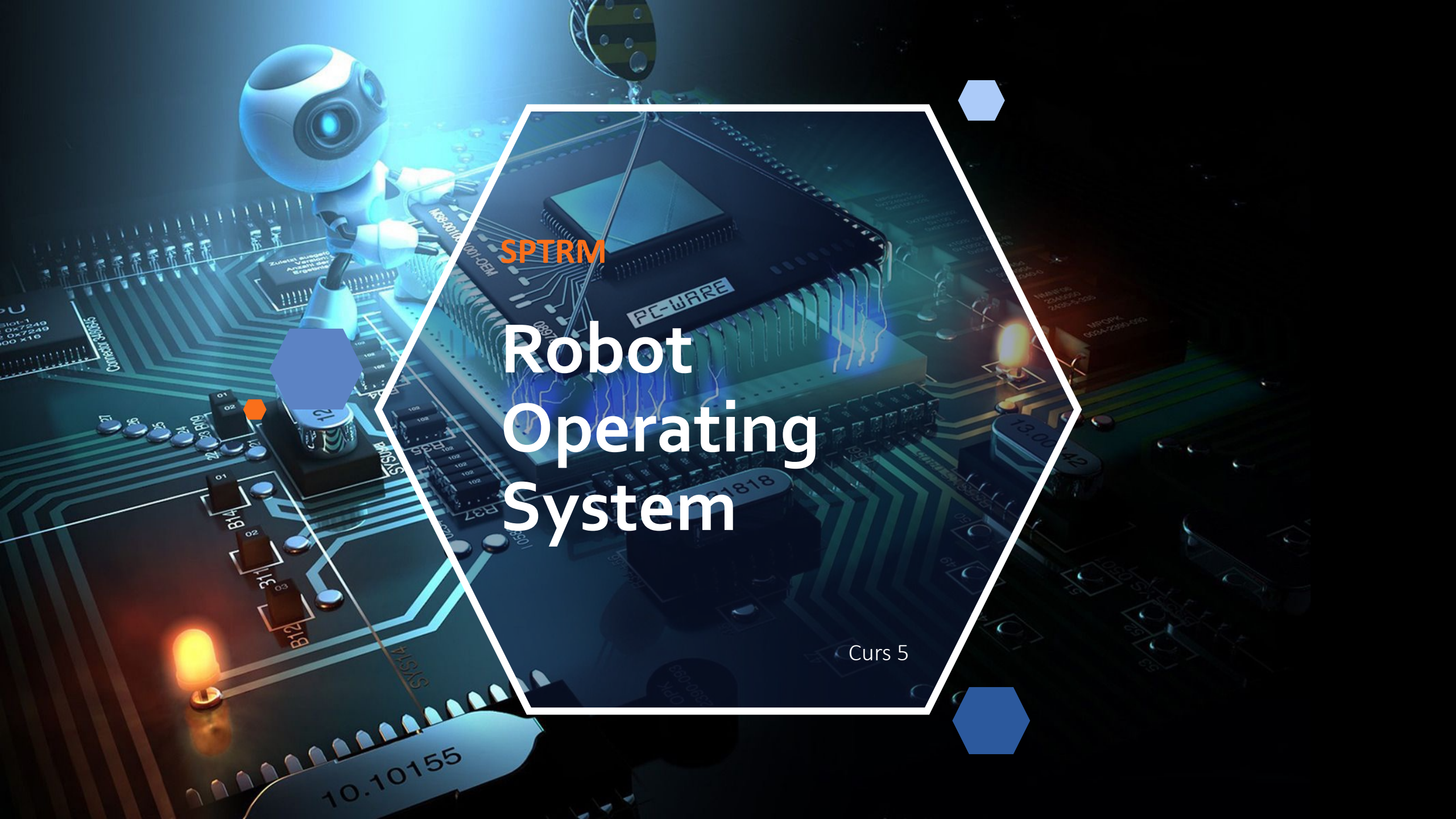
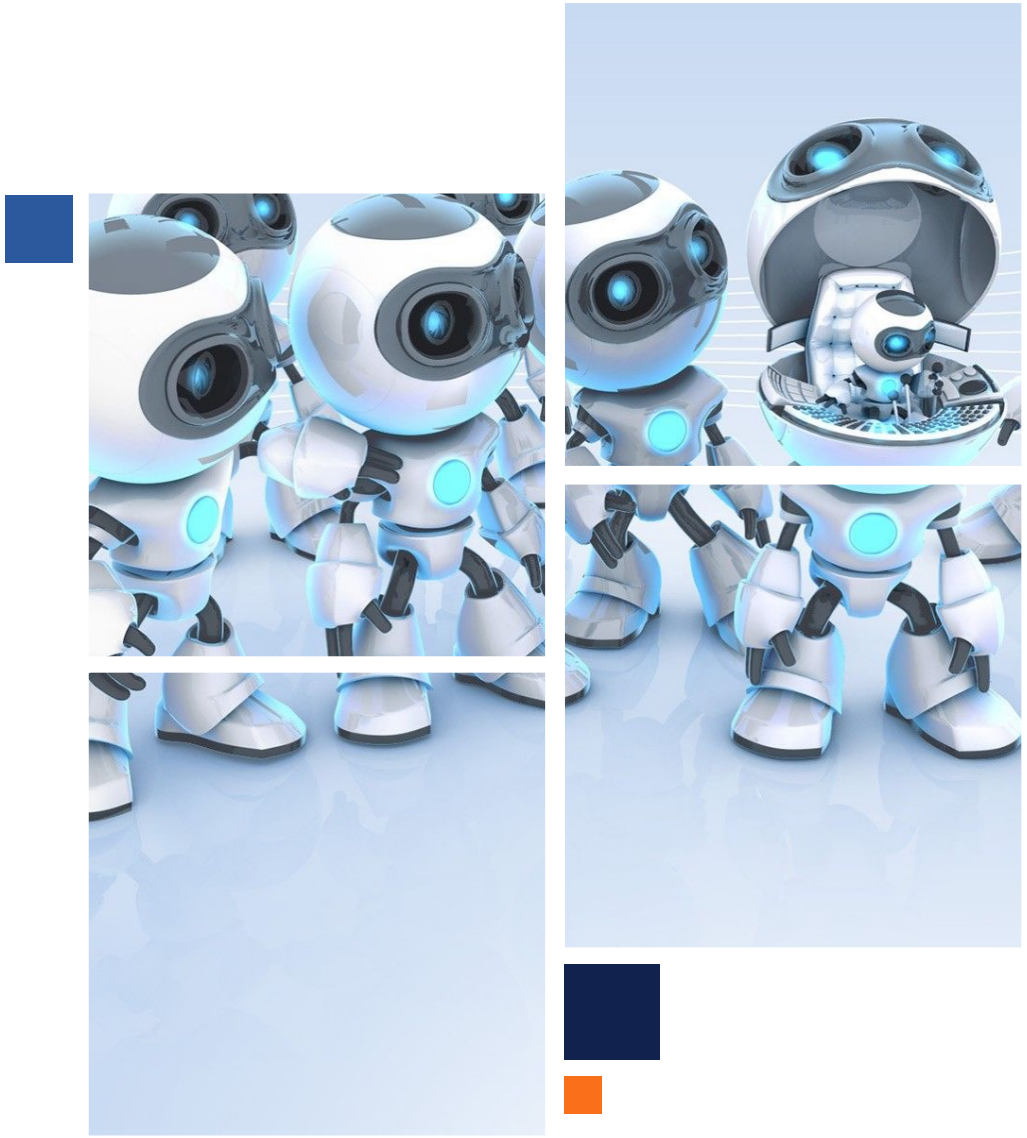
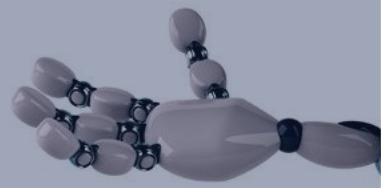

SPTRM
Robot Operating System
Curs 5

Agenda

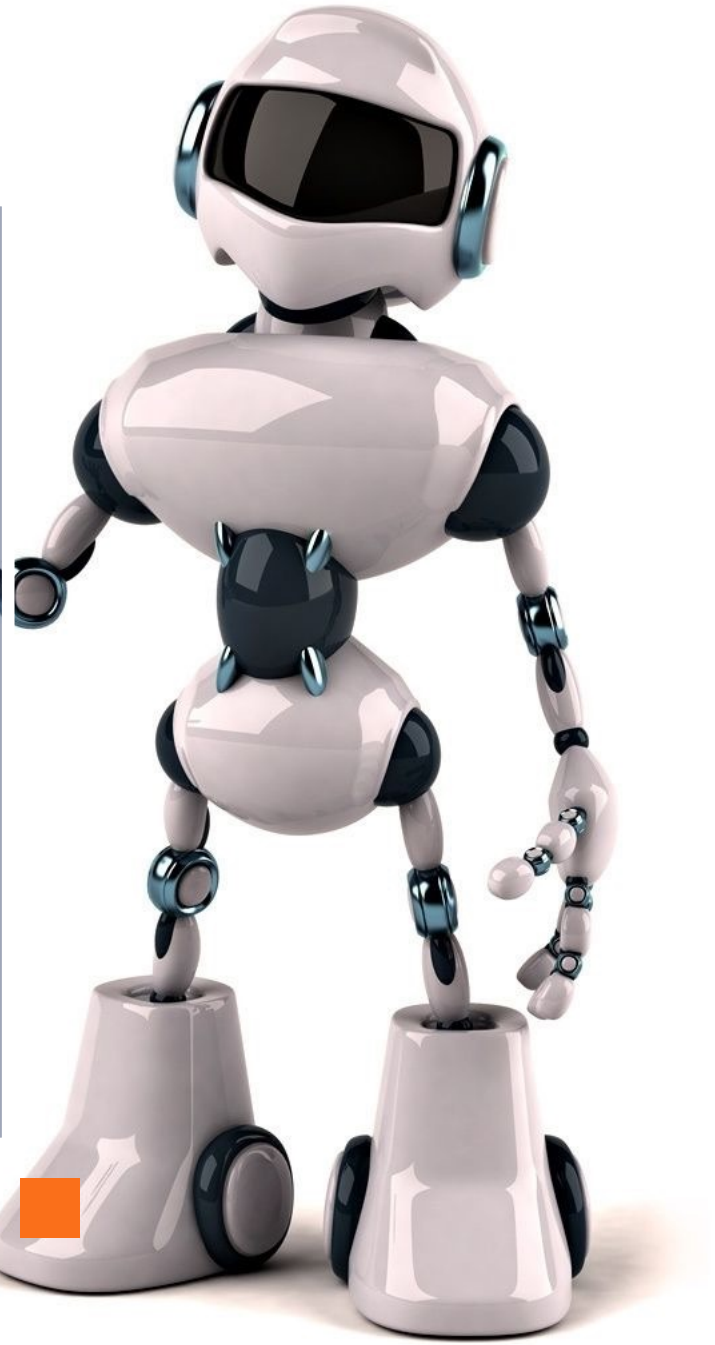
- Recap
- URDF vs xacro
- 7DOF Manipulator
  - Study case
  - Visualisation
  - Simulation
- Robot. Sensor. Motor
  - Robot packages
  - Sensor packages
  - Motor packages



Recap

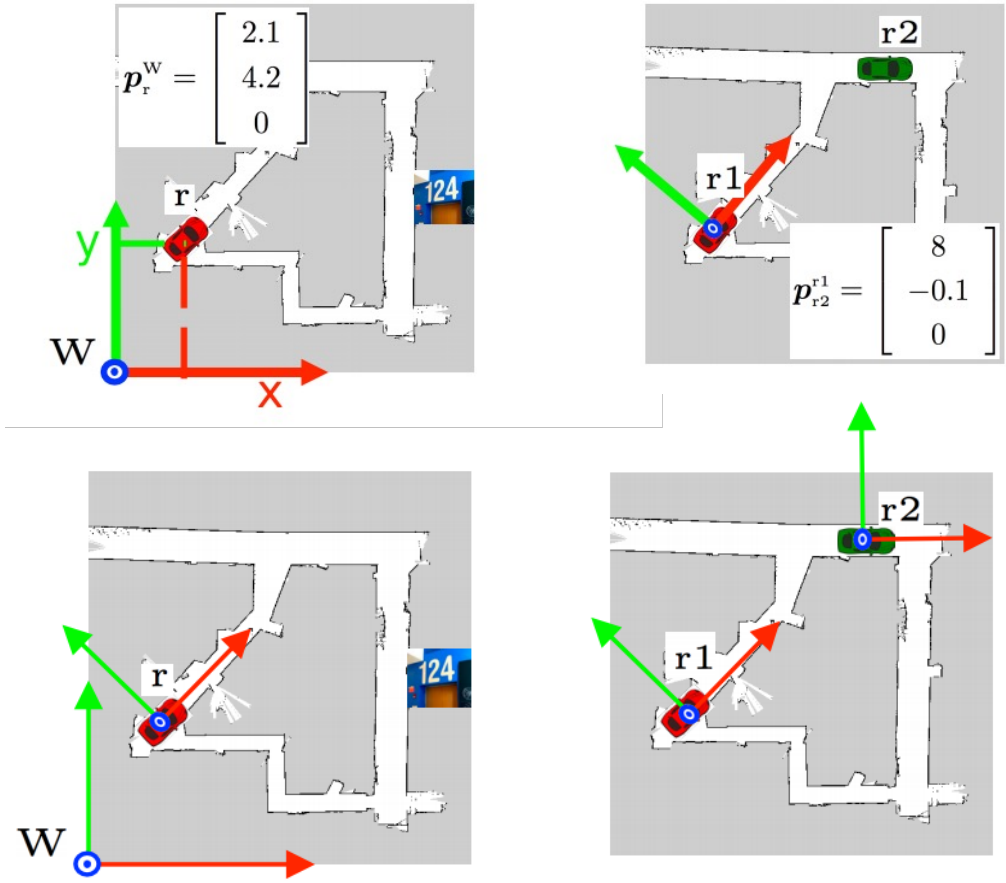


From the last episode…



Coordinate frames representation

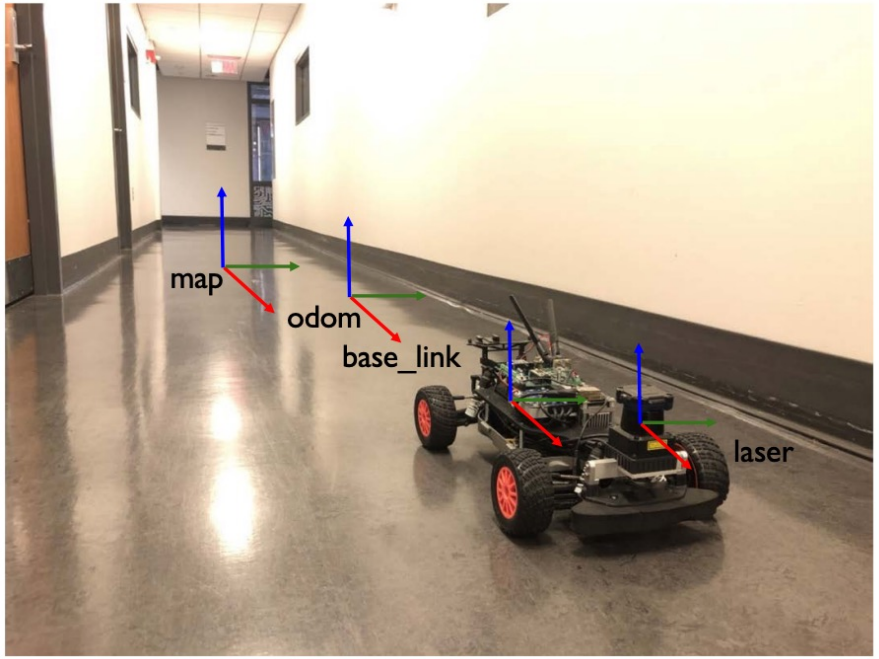
- Compact representation of points and orientations
- Coordinates are always considered in regard to a coordinate frame
- Coordinates have no meaning without specifying coordinate frame
- Pose = position + rotation



ROS Coordinate Frames

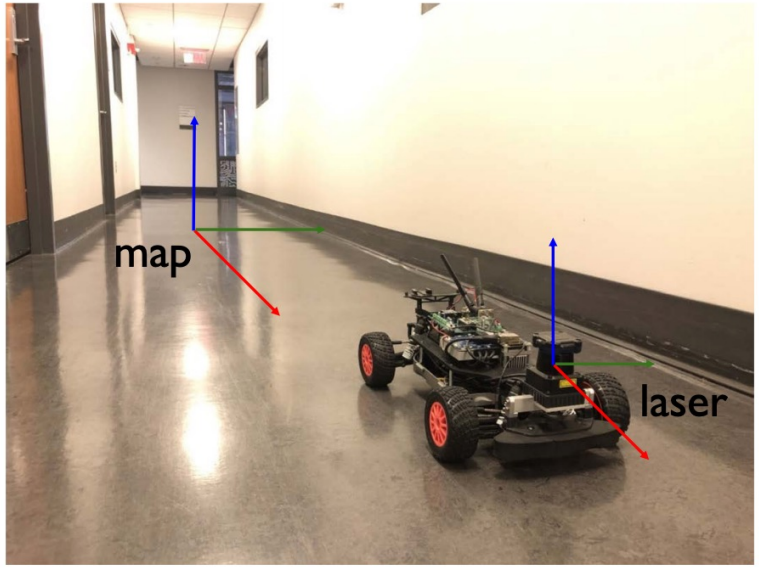
- Usually in ROS one can find 4 coordinate frames
  - Map frame
  - Robot Base (base_link) frame
  - Tool frame
  - Odometry frame
- Odometry = distance measurement technique for vehicles and pedestrians

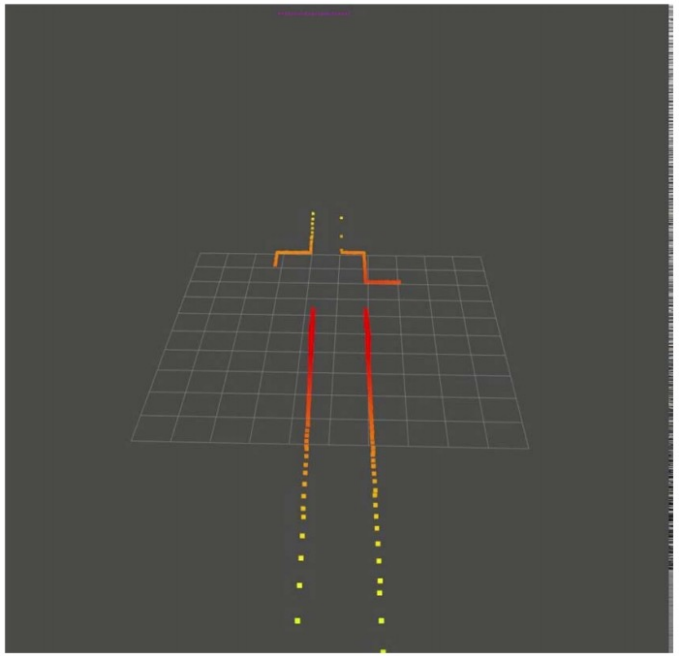
RGB → XYZ

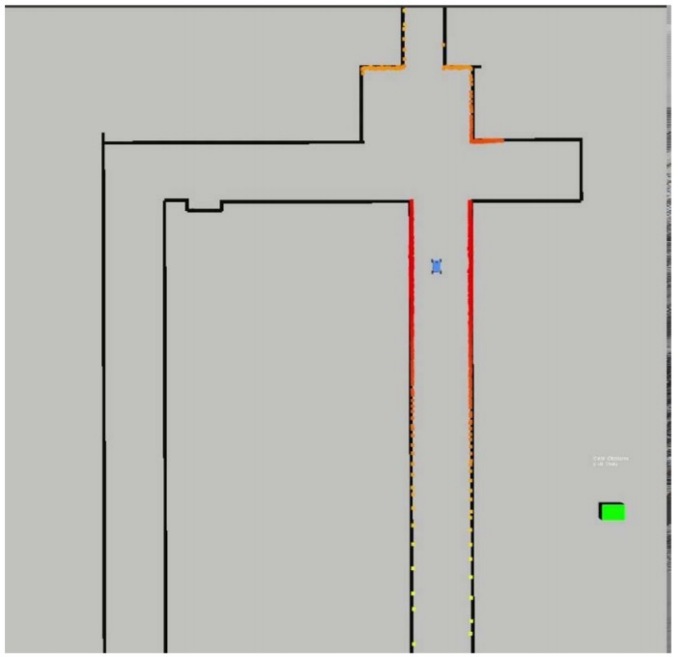


Transformations

- Giving sense to collected data (submaps)
- Left: 2 coordinate frames: map + laser
- Middle: Laser frame
- Right: Map frame

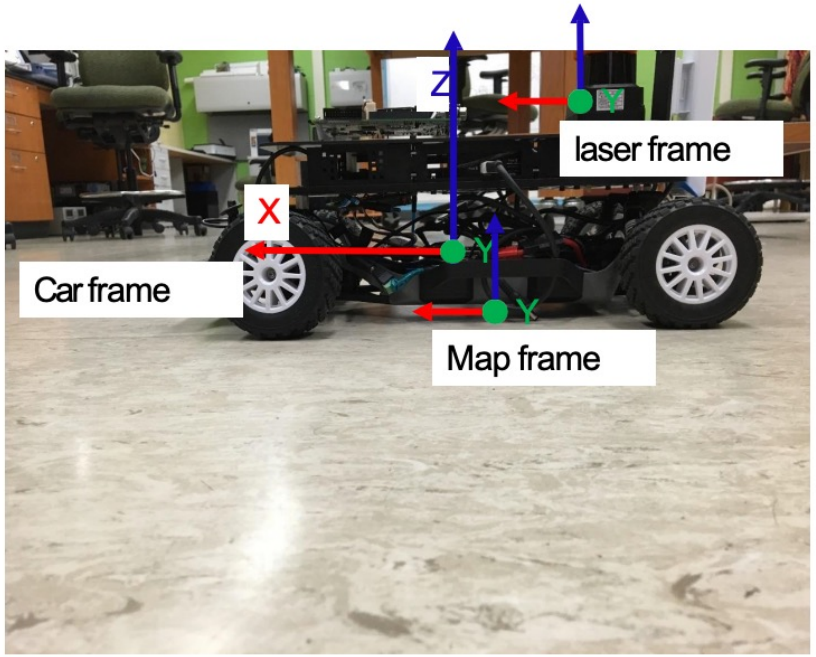


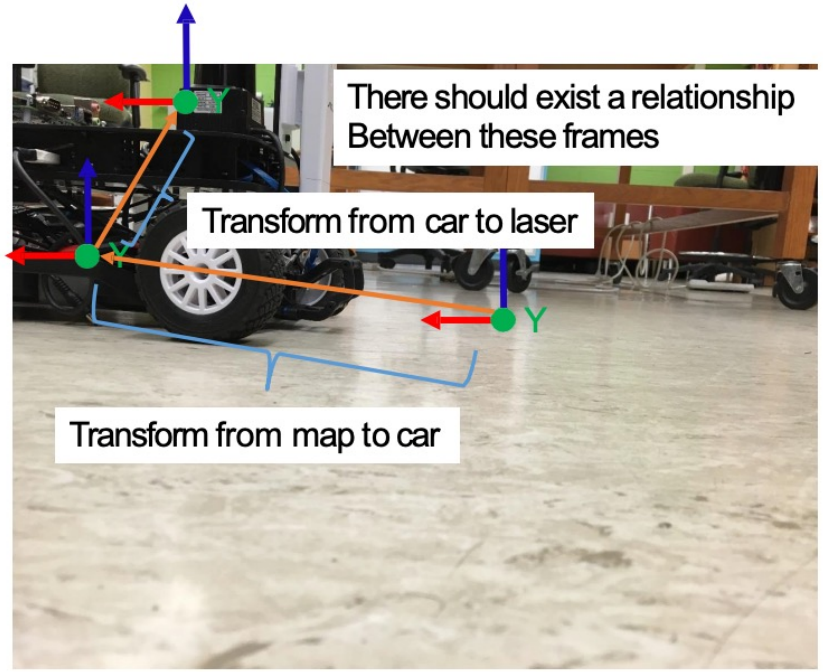




Transformations

- Define relations between frames
- Convert measurements from one frame to another
- Eg: Relationship between the measurement frame and the actuation frame



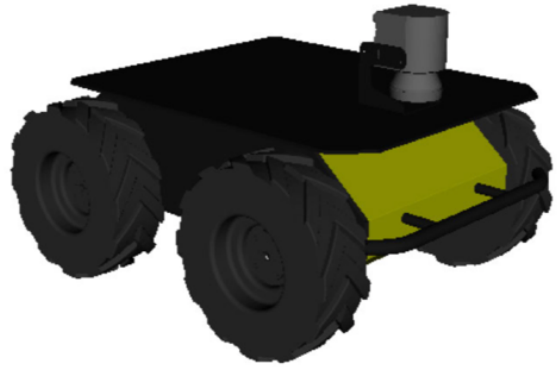


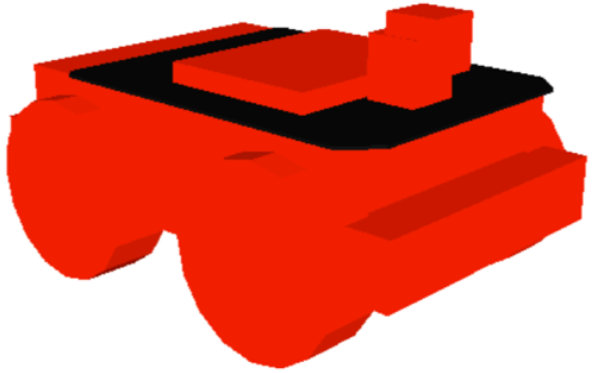
Robot models

- Unified Robot Description Format (URDF)

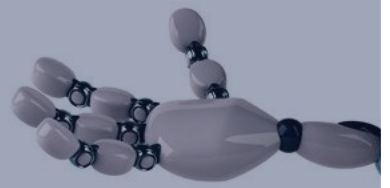
  http://wiki.ros.org/urdf
- XML file for representing a robot model
- Kinematic and dynamic description
- Visual representation (mesh)
- Collision model (primitives)
- URDF generation can be be scripted with XACRO

  http://wiki.ros.org/xacro



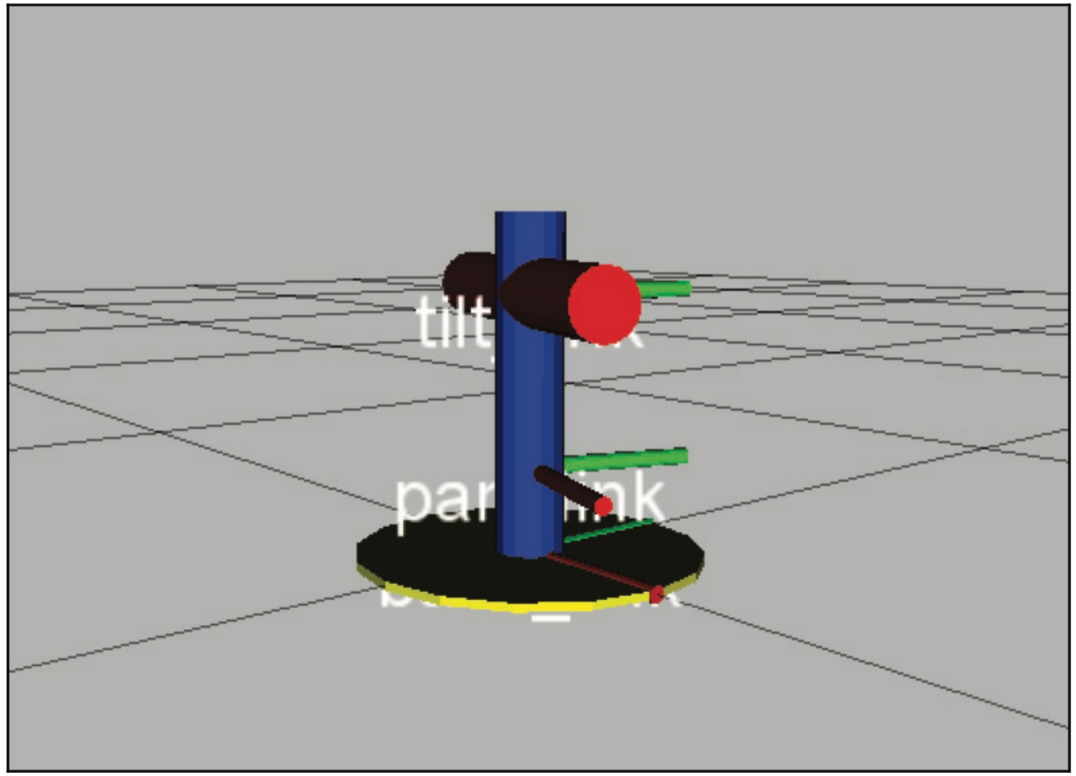


URDF vs xacro



Continued...

URDF model



- Pan-and-tilt mechanism
- There are three links and two joints in this mechanism.
- The base link is static, and all the other links are mounted on it.
- The first joint can pan on its axis, and the second link is mounted on the first link, and it can tilt on its axis.

URDF model

```
<?xml version="1.0"?>
<robot name="pan_tilt">
```

- Defines the name of the robot

```
<link name="base_link">
  <visual>
    <geometry>
    <cylinder length="0.01" radius="0.2"/>
    </geometry>
    <origin rpy="0 0 0" xyz="0 0 0"/>
    <material name="yellow">
      <color rgba="1 1 0 1"/>
    </material>
  </visual>
</link>
```

- Base link definition for the pan-and-tilt mechanism
  - geometry: cylinder / box / sphere / mesh
  - visual appearance of the link: color / texture

URDF model

- Define actuators using the **transmission** tag
  - Type
  - Joint
  - Parameters
  - Interface

```
<transmission name="tran1">
  <type>transmission_interface/SimpleTransmission</type>
  <joint name="shoulder_pan_joint">
    <hardwareInterface>hardware_interface/PositionJointInterface</hardwareInterface>
  </joint>
  <actuator name="motor1">
    <hardwareInterface>hardware_interface/PositionJointInterface</hardwareInterface>
    <mechanicalReduction>1</mechanicalReduction>
  </actuator>
</transmission>
```

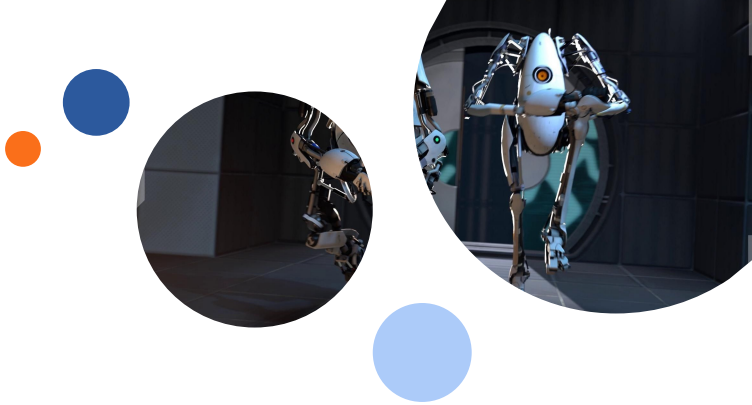
- gazebo_ros_control plugin

URDF model

```
<joint name="pan_joint" type="revolute">
  <parent link="base_link"/>
  <child link="pan_link"/>
  <origin xyz="0 0 0.1"/>
  <axis xyz="0 0 1" />
</joint>
```

- a joint with a unique name and its joint type
- supported join types
  - **revolute** - a hinge joint that rotates along the axis and has a limited range specified by the upper and lower limits.
  - **continuous** - a continuous hinge joint that rotates around the axis and has no upper and lower limits.
  - **prismatic** - a sliding joint that slides along the axis, and has a limited range specified by the upper and lower limits.
  - **fixed** - This is not really a joint because it cannot move. All degrees of freedom are locked. This type of joint does not require the axis, calibration, dynamics, limits or safety_controller.
  - **floating** - This joint allows motion for all 6 degrees of freedom.
  - **planar** - This joint allows motion in a plane perpendicular to the axis.

Tools



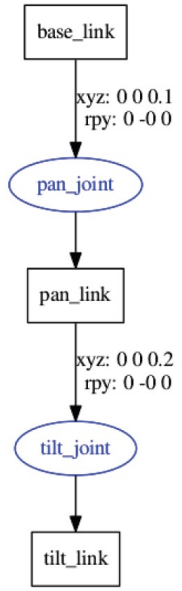
- Error checking

```
$ check_urdf pan_tilt.urdf
```

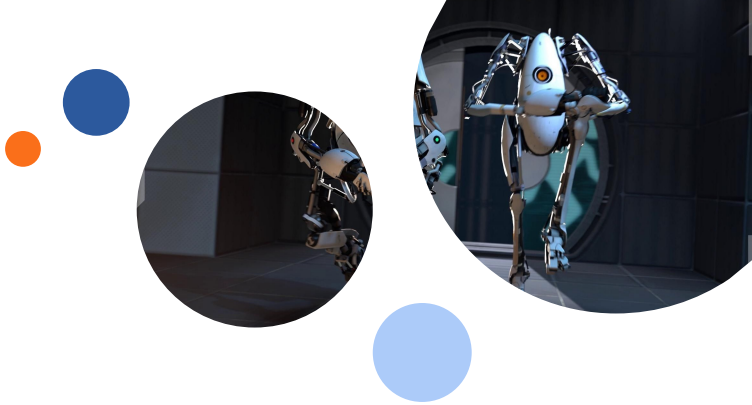
```
robot name is: pan_tilt
---------- Successfully Parsed XML ---------------
  root Link: base_link has 1 child(ren)
    child(1):  pan_link
      child(1):  tilt_link
```

- View the structure of the robot links and joints graphically

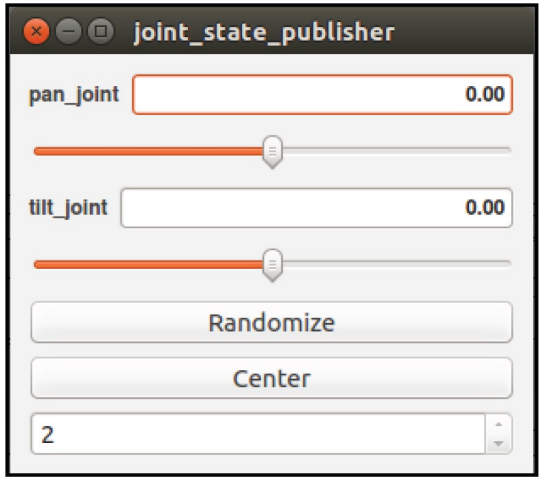
```
$ urdf_to_graphiz pan_tilt.urdf
```



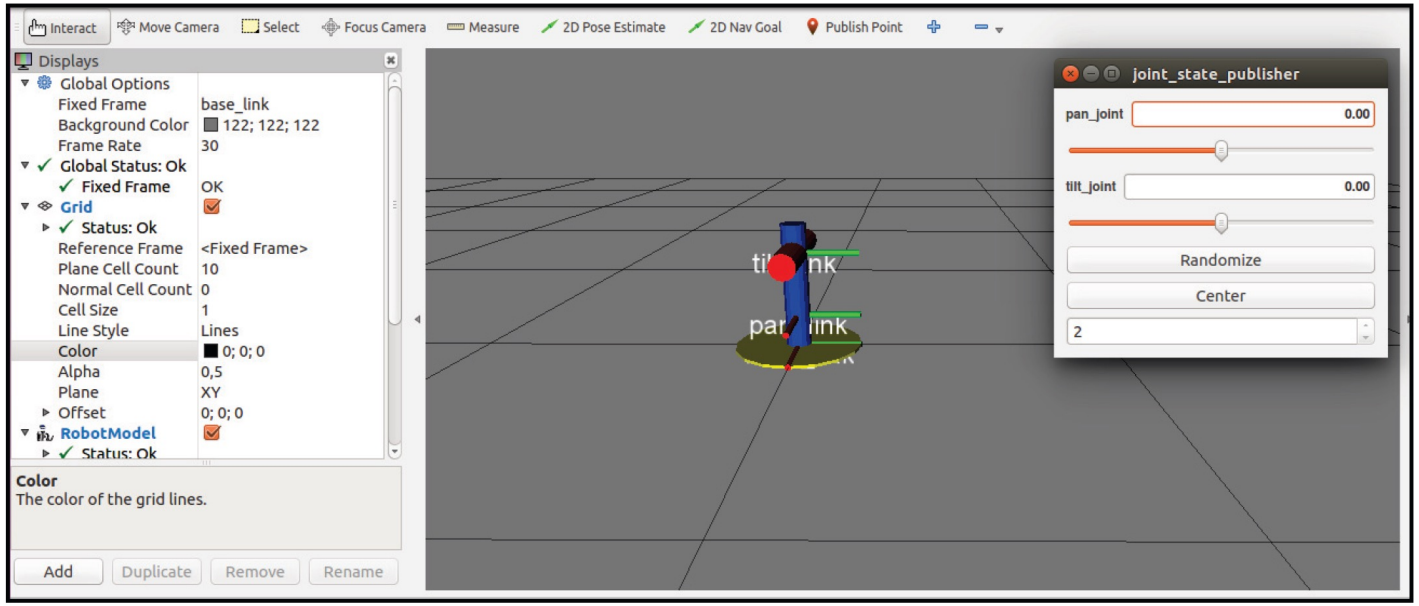
Launching RViz



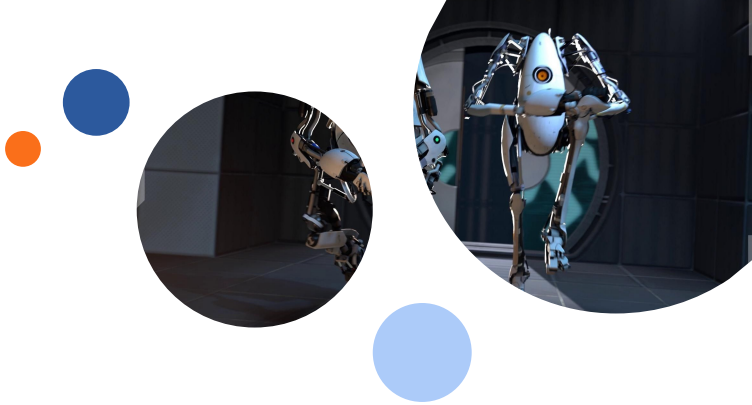
```
<launch>
      <arg name="model" />
      <param
            name="robot_description"
            textfile="$(find my_robot_description_pkg)/urdf/pan_tilt.urdf" />
      <param name="use_gui" value="true"/>
      <node name="joint_state_publisher" pkg="joint_state_publisher" type="joint_state_publisher" />
      <node name="robot_state_publisher" pkg="robot_state_publisher" type="state_publisher" />
      <node name="rviz" pkg="rviz" type="rviz" required="true" args="-d $(find my_robot_description_pkg)/urdf.rviz" />
</launch>
```



SPTRM



Adding physical and collision properties



```
<link>
......
<collision>
      <geometry>
      <cylinder length="0.03" radius="0.2"/>
      </geometry>
      <origin rpy="0 0 0" xyz="0 0 0"/>
    </collision>

    <inertial>
    <mass value="1"/>
    <inertia ixx="1.0" ixy="0.0" ixz="0.0" iyy="1.0" iyz="0.0" izz="1.0"/>
    </inertial>
...........
</link>
```

- define the robot link's physical properties, such as geometry, color, mass, and inertia, as well as the collision properties of the link
  - collision
  - inertia
- Required in each link to load correctly in a robot simulator

xacro

- URDF
  - Is a single file
  - Lacks variables, constants
  - Complicated non-modular approach

- xacro
  - Extends URDF by bringing
    - Flexibility
    - Reusability
    - Modularity
    - Programmability
  - Simplified version of URDF
  - Supports a simple programming statement
  - Brings variables, constants, mathematical expressions, conditional statements
  - Convert xacro to URDF

```
$ rosrun xacro xacro pan_tilt.xacro --inorder > pan_tilt_generated.urdf
```

xacro properties

- Declaration of constants and properties
- Easier to change than hardcoded values

```
<xacro:property name="base_link_length" value="0.01" />
<xacro:property name="base_link_radius" value="0.2" />
<xacro:property name="pan_link_length" value="0.4" />
<xacro:property name="pan_link_length" value="0.04" />
```

- Replace hardcoded value with the following definition

```
<cylinder length="${pan_link_length}" radius="${pan_link_radius}" />
```

- Using math expression

```
<cylinder length="${pan_link_length}" radius="${pan_link_radius+0.02}" />
```

xacro macro

- macro
  - Reduce the length of complex definitions
  - Easier to change than hardcoded values

```
<xacro:macro name="inertial_matrix" params="mass">
    <inertial>
        <mass value="${mass}" />
        <inertia ixx="0.5" ixy="0.0" ixz="0.0" iyy="0.5" iyz="0.5" izz="0.5" />
    </inertial>
</xacro:macro>
```

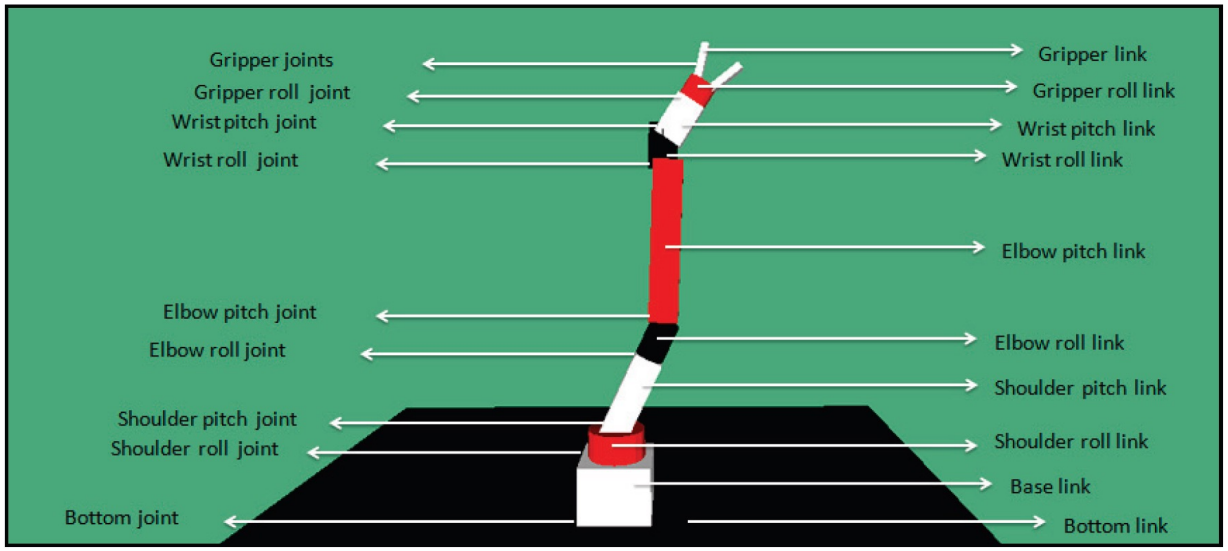
  - Replace each inertial code with a single line

```
<xacro:inertial_matrix mass="1">
```

7 DOF robot manipulator

Study Case

Robot



- Degrees of freedom: 7
- Length of the arm: 50 cm
- Reach of the arm: 35 cm
- Number of links: 12
- Number of joints: 11

Joints

Joint number	Joint name	Joint type	Angle limits
1	bottom_joint	Fixed	--
2	shoulder_pan_joint	Revolute	-150 to 114 degrees
3	shoulder_pitch_joint	Revolute	-67 to 109 degrees
4	elbow_roll_joint	Revolute	-150 to 41 degrees
5	elbow_pitch_joint	Revolute	-92 to 110 degrees
6	wrist_roll_joint	Revolute	-150 to 150 degrees
7	wrist_pitch_joint	Revolute	92 to 113 degrees
8	gripper_roll_joint	Revolute	-150 to 150 degrees
9	finger_joint1	Prismatic	0 to 3 cm degrees
10	finger_joint1	Prismatic	0 to 3 cm degrees

1. Define properties

- Conversions (degree to radian)
- Constants values: PI
- Dimensions for links:
  - Length
  - Width
  - Height

```
<xacro:property name="deg_to_rad" value="0.017453292519943295777"/>

<!-- Constants -->
<xacro:property name="M_SCALE" value="0.001 0.001 0.001"/>
<xacro:property name="M_PI" value="3.14159"/>

<!-- Shoulder pan link properties -->
<xacro:property name="shoulder_pan_width" value="0.04" />
<xacro:property name="shoulder_pan_len" value="0.08" />


<!-- Shoulder pitch link properties -->
<xacro:property name="shoulder_pitch_len" value="0.14" />
<xacro:property name="shoulder_pitch_width" value="0.04" />
<xacro:property name="shoulder_pitch_height" value="0.04" />


<!-- Elbow roll link properties -->
<xacro:property name="elbow_roll_width" value="0.02" />
<xacro:property name="elbow_roll_len" value="0.06" />


<!-- Elbow pitch link properties -->
<xacro:property name="elbow_pitch_len" value="0.22" />
<xacro:property name="elbow_pitch_width" value="0.04" />
<xacro:property name="elbow_pitch_height" value="0.04" />
```

```
<!-- Wrist roll link properties -->
<xacro:property name="wrist_roll_width" value="0.02" />
<xacro:property name="wrist_roll_len" value="0.04" />


<!-- wrist pitch link properties -->
<xacro:property name="wrist_pitch_len" value="0.06" />
<xacro:property name="wrist_pitch_width" value="0.04" />
<xacro:property name="wrist_pitch_height" value="0.04" />


<!-- Gripper roll link properties -->
<xacro:property name="gripper_roll_width" value="0.04" />
<xacro:property name="gripper_roll_len" value="0.02" />


<!-- Left gripper -->
<xacro:property name="left_gripper_len" value="0.08" />
<xacro:property name="left_gripper_width" value="0.01" />
<xacro:property name="left_gripper_height" value="0.01" />


<!-- Right gripper -->
<xacro:property name="right_gripper_len" value="0.08" />
<xacro:property name="right_gripper_width" value="0.01" />
<xacro:property name="right_gripper_height" value="0.01" />
```

2. Define macros

- Repetitive portions of URDF
- Easier to read

```
<xacro:macro name="inertial_matrix" params="mass">
   <inertial>
     <mass value="${mass}" />
     <inertia ixx="1.0" ixy="0.0" ixz="0.0" iyy="0.5" iyz="0.0" izz="1.0" />
   </inertial>
</xacro:macro>
```

```
<xacro:macro name="transmission_block" params="joint_name">
       <transmission name="tran1">
         <type>transmission_interface/SimpleTransmission</type>
         <joint name="${joint_name}">
           <hardwareInterface>hardware_interface/PositionJointInterface</hardwareInterface>
         </joint>
         <actuator name="motor1">
           <hardwareInterface>hardware_interface/PositionJointInterface</hardwareInterface>
           <mechanicalReduction>1</mechanicalReduction>
         </actuator>
       </transmission>
</xacro:macro>
```

3. Define the base of the robot

- The base on which the arm is attached

```
<joint name="bottom_joint" type="fixed">
  <origin xyz="0 0 0" rpy="0 0 0" />
  <parent link="base_link"/>
  <child link="bottom_link"/>
</joint>
```

```
<gazebo reference="bottom_link">
  <material>Gazebo/White</material>
</gazebo>
```

```
<link name="bottom_link">

  <visual>
    <origin xyz=" 0 0 -0.04"  rpy="0 0 0"/>
    <geometry>

            <box size="1 1 0.02" />

    </geometry>
    <material name="Brown" />
  </visual>

  <collision>
    <origin xyz=" 0 0 -0.04"  rpy="0 0 0"/>
    <geometry>
            <box size="1 1 0.02" />
    </geometry>
    </collision>>

</link>
```

4. Define links and joints

- Succession of **link** and **joint** elements
- Use **gazebo** tag for color definition in Gazebo

```
<link name="base_link">

  <visual>
    <origin xyz="0 0 0" rpy="${M_PI/2} 0 0" /> <!-- rotate PI/2 -->
    <geometry>

             <box size="0.1 0.1 0.1" />

    </geometry>
    <material name="White" />
  </visual>

  <collision>
    <origin xyz="0 0 0" rpy="${M_PI/2} 0 0" /> <!-- rotate PI/2 -->
    <geometry>

             <box size="0.1 0.1 0.1" />

    </geometry>
    </collision>>
      <xacro:inertial_matrix mass="1"/>

</link>
```

```
<gazebo reference="base_link">
  <material>Gazebo/White</material>
</gazebo>

<joint name="shoulder_pan_joint" type="revolute">
  <parent link="base_link"/>
  <child link="shoulder_pan_link"/>
  <origin xyz="0 0 0.05" rpy="0 ${M_PI/2} ${M_PI*0}" />
  <axis xyz="-1 0 0" />
  <limit effort="300" velocity="1" lower="-2.61799387799" upper="1.98394848567"/>
  <dynamics damping="50" friction="1"/>
</joint>
```

4. Define links and joints

- Succession of **link** and **joint** elements
- Use **gazebo** tag for color definition in Gazebo

```
<link name="shoulder_pan_link" >

  <visual>
    <origin xyz="0 0 0" rpy="0 ${M_PI/2} 0" />
    <geometry>

          <cylinder radius="${shoulder_pan_width}" length="${shoulder_pan_len}"/>

    </geometry>
    <material name="Red" />
  </visual>

  <collision>
    <origin xyz="0 0 0" rpy="0 ${M_PI/2} 0" />
    <geometry>
          <cylinder radius="${shoulder_pan_width}" length="${shoulder_pan_len}"/>
    </geometry>
  </collision>
      <xacro:inertial_matrix mass="1"/>
</link>
```

```
<gazebo reference="shoulder_pan_link">
  <material>Gazebo/Red</material>
</gazebo>

<joint name="shoulder_pitch_joint" type="revolute">
  <parent link="shoulder_pan_link"/>
  <child link="shoulder_pitch_link"/>
  <origin xyz="-0.041 0.0021 0.0" rpy="-${M_PI/2} 0 ${M_PI/2}" />
  <axis xyz="1 0 0" />
  <limit effort="300" velocity="1" lower="-1.19962513147" upper="1.89994105047" />
  <dynamics damping="50" friction="1"/>
</joint>
```

4. Define links and joints

- Succession of **link** and **joint** elements
- Use **gazebo** tag for color definition in Gazebo

```
<link name="shoulder_pitch_link" >

  <visual>
    <origin xyz="-0.002 0 0.04" rpy="0 ${M_PI/2} 0" />
    <geometry>

         <box size="${shoulder_pitch_len} ${shoulder_pitch_width} ${shoulder_pitch_height}" />

    </geometry>
    <material name="White" />
  </visual>

  <collision>
    <origin xyz="-0.002 0 0.04" rpy="0 ${M_PI/2} 0" />
    <geometry>
         <box size="${shoulder_pitch_len} ${shoulder_pitch_width} ${shoulder_pitch_height}" />
    </geometry>
  </collision>
      <xacro:inertial_matrix mass="1"/>
</link>
```

```
<gazebo reference="shoulder_pitch_link">
  <material>Gazebo/White</material>
</gazebo>

<joint name="elbow_roll_joint" type="revolute">
  <parent link="shoulder_pitch_link"/>
  <child link="elbow_roll_link"/>
  <origin xyz="-0.002 0 0.1206" rpy="${M_PI} ${M_PI/2} 0" />
  <axis xyz="-1 0 0" />
  <limit effort="300" velocity="1" lower="-2.61799387799" upper="0.705631162427" />
  <dynamics damping="50" friction="1"/>
</joint>
```

5. Add transmission information and sensors

- Make of the transmission_block macro
- Include other xacro files
  - a xacro definition of a sensor called **Asus Xtion pro**

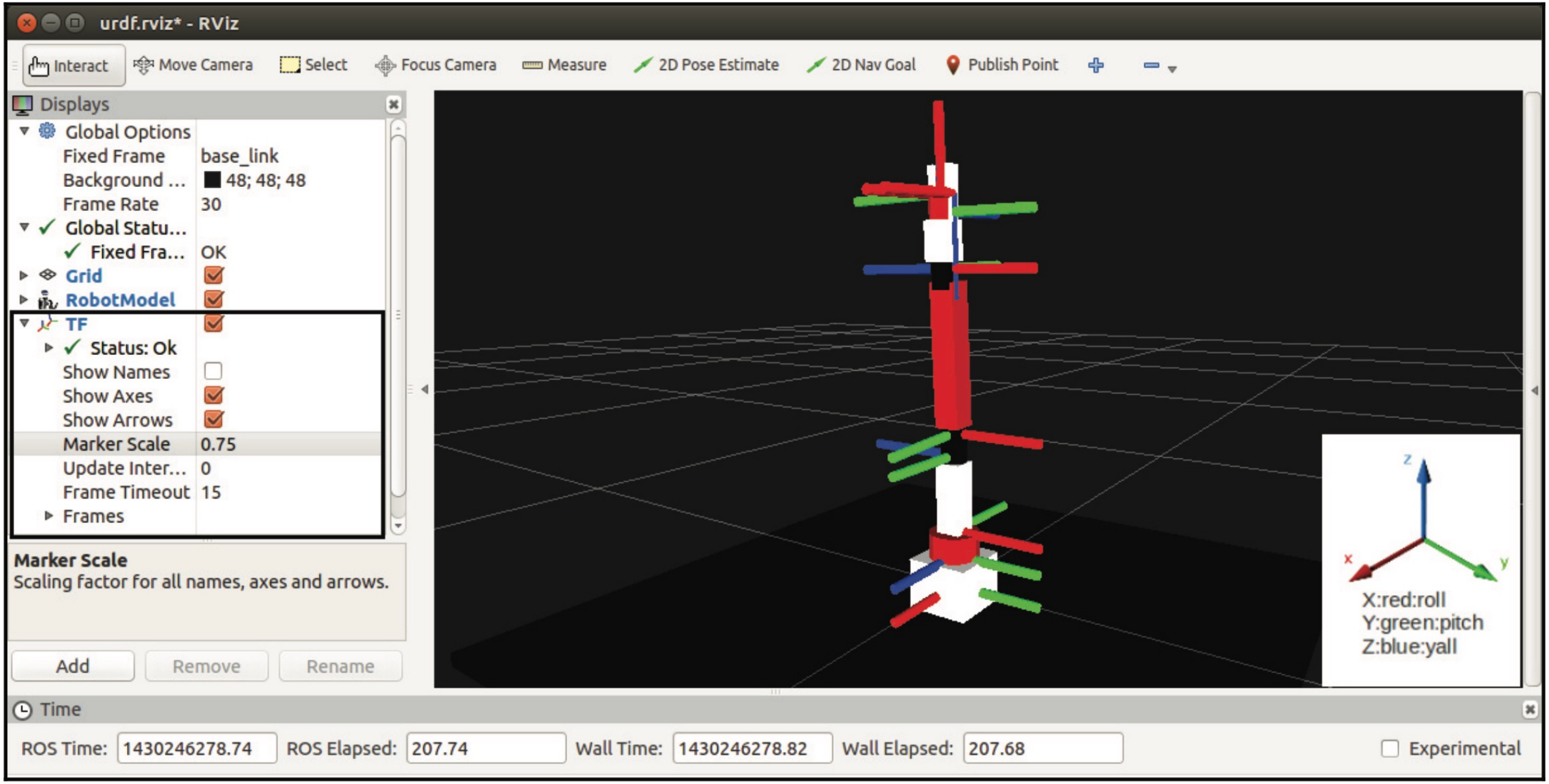
```
<xacro:transmission_block joint_name="shoulder_pan_joint"/>
<xacro:transmission_block joint_name="shoulder_pitch_joint"/>
<xacro:transmission_block joint_name="elbow_roll_joint"/>
<xacro:transmission_block joint_name="elbow_pitch_joint"/>
<xacro:transmission_block joint_name="wrist_roll_joint"/>
<xacro:transmission_block joint_name="wrist_pitch_joint"/>
<xacro:transmission_block joint_name="gripper_roll_joint"/>
<xacro:transmission_block joint_name="finger_joint1"/>
<xacro:transmission_block joint_name="finger_joint2"/>
```

```
<xacro:include filename="$(find mastering_ros_robot_description_pkg)/urdf/sensors/xtion_pro_live.urdf.xacro"/>
```

6. View robot in RViz

- Create a launch file
- Control each joint using the joint state publisher

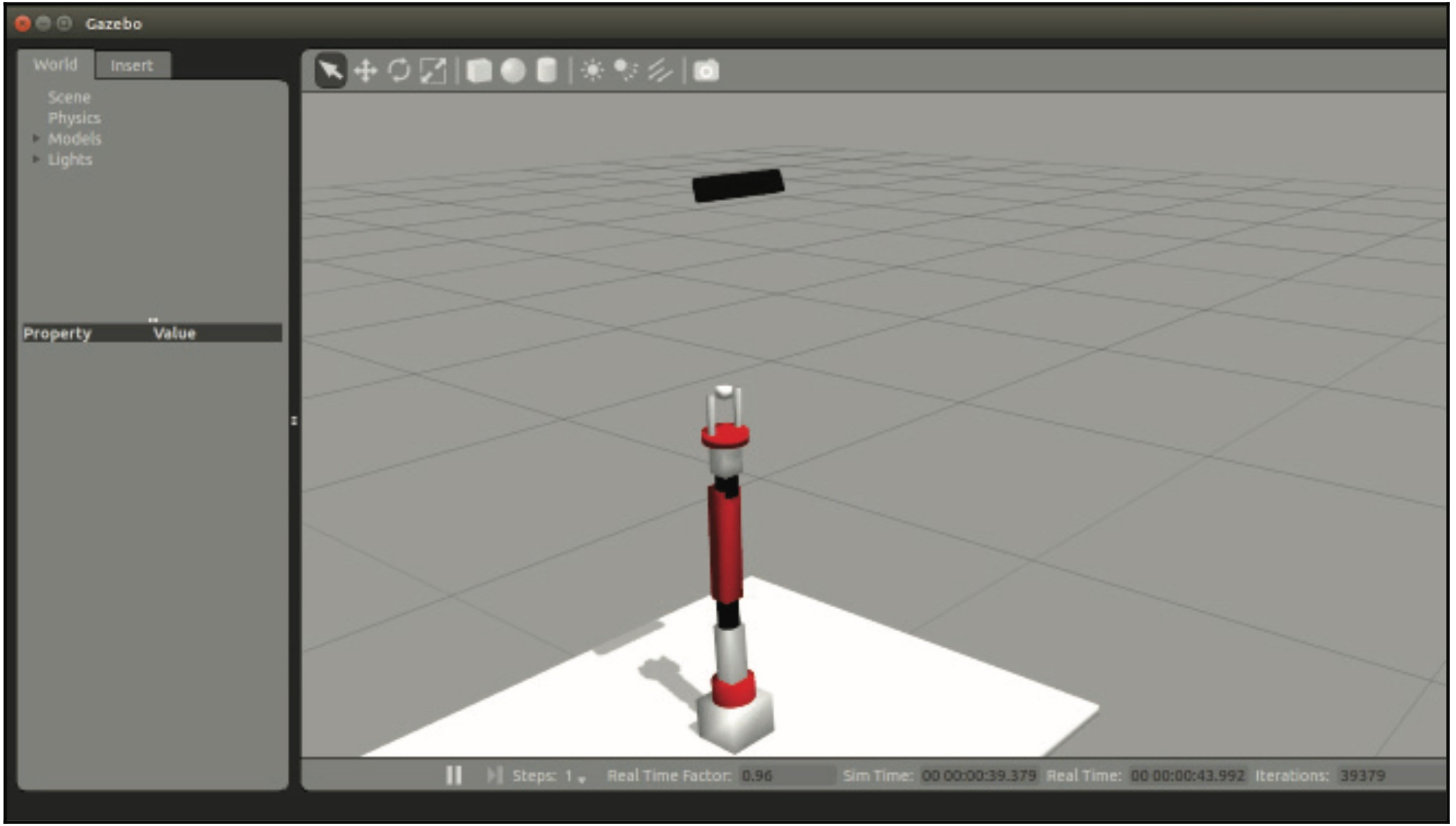
```
<launch>
        <arg name="model" />
        <!-- Parsing xacro and setting robot_description parameter -->

        <param name="robot_description" command="$(find xacro)/xacro --inorder $(find mastering_ros_robot_description_pkg)/urdf/seven_dof_arm.xacro" />


        <!-- Setting gui parameter to true for display joint slider -->
        <param name="use_gui" value="true"/>

        <!-- Starting Joint state publisher node which will publish the joint values -->
        <node name="joint_state_publisher" pkg="joint_state_publisher" type="joint_state_publisher" />

        <!-- Starting robot state publish which will publish tf -->
        <node name="robot_state_publisher" pkg="robot_state_publisher" type="state_publisher" />

        <!-- Launch visualization in rviz -->
        <node name="rviz" pkg="rviz" type="rviz" args="-d $(find mastering_ros_robot_description_pkg)/urdf.rviz" required="true" />

</launch>
```


urdf.rviz* - RViz
Interact
Move Camera
Select
Focus Camera
Measure
2D Pose Estimate
2D Nav Goal
Publish Point
Displays
Global Options
Fixed Frame
base_link
Background ...
48; 48; 48
Frame Rate
30
Global Statu...
Fixed Fra...
OK
Grid
RobotModel
TF
Status: Ok
Show Names
Show Axes
Show Arrows
Marker Scale
0.75
Update Inter...
0
Frame Timeout
15
Frames
Marker Scale
Scaling factor for all names, axes and arrows.
Add
Remove
Rename
Z
x
y
X:red:roll
Y:green:pitch
Z:blue:yall
Time
ROS Time:
1430246278.74
ROS Elapsed:
207.74
Wall Time:
1430246278.82
Wall Elapsed:
207.68
Experimental

6. Simulation using Gazebo

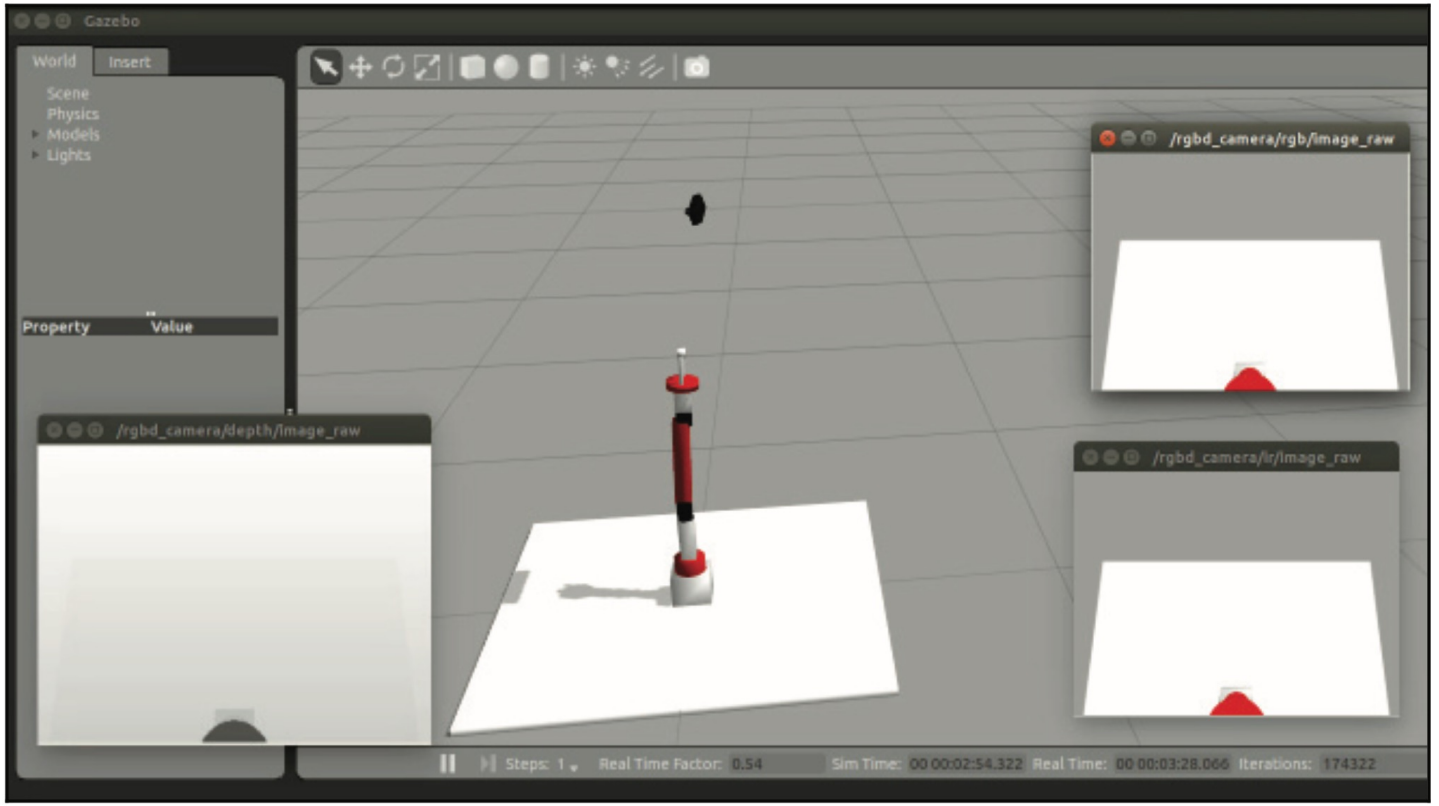
- Create a launch file

```
<launch>

  <!-- these are the arguments you can pass this launch file, for example paused:=true -->
  <arg name="paused" default="false"/>
  <arg name="use_sim_time" default="true"/>
  <arg name="gui" default="true"/>
  <arg name="headless" default="false"/>
  <arg name="debug" default="false"/>

  <!-- We resume the logic in empty_world.launch -->
  <include file="$(find gazebo_ros)/launch/empty_world.launch">
    <arg name="debug" value="$(arg debug)" />
    <arg name="gui" value="$(arg gui)" />
    <arg name="paused" value="$(arg paused)"/>
    <arg name="use_sim_time" value="$(arg use_sim_time)"/>
    <arg name="headless" value="$(arg headless)"/>
  </include>

  <!-- Load the URDF into the ROS Parameter Server -->
  <param name="robot_description" command="$(find xacro)/xacro --inorder '$(find mastering_ros_robot_description_pkg)/urdf/seven_dof_arm.xacro'" />


  <!-- Run a python script to the send a service call to gazebo_ros to spawn a URDF robot -->
  <node name="urdf_spawner" pkg="gazebo_ros" type="spawn_model" respawn="false" output="screen"
        args="-urdf -model seven_dof_arm -param robot_description"/>


</launch>
```

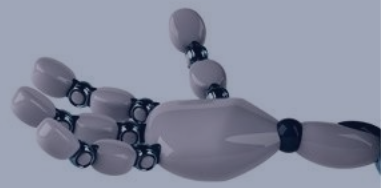

Gazebo
World
Insert
Scene
Physics
Models
Lights
Property
Value
Steps: 1
Real Time Factor: 0.96
Sim Time: 00 00:00:39.379
Real Time: 00 00:00:43.992
Iterations: 39379

7. Start RViz for sensor data

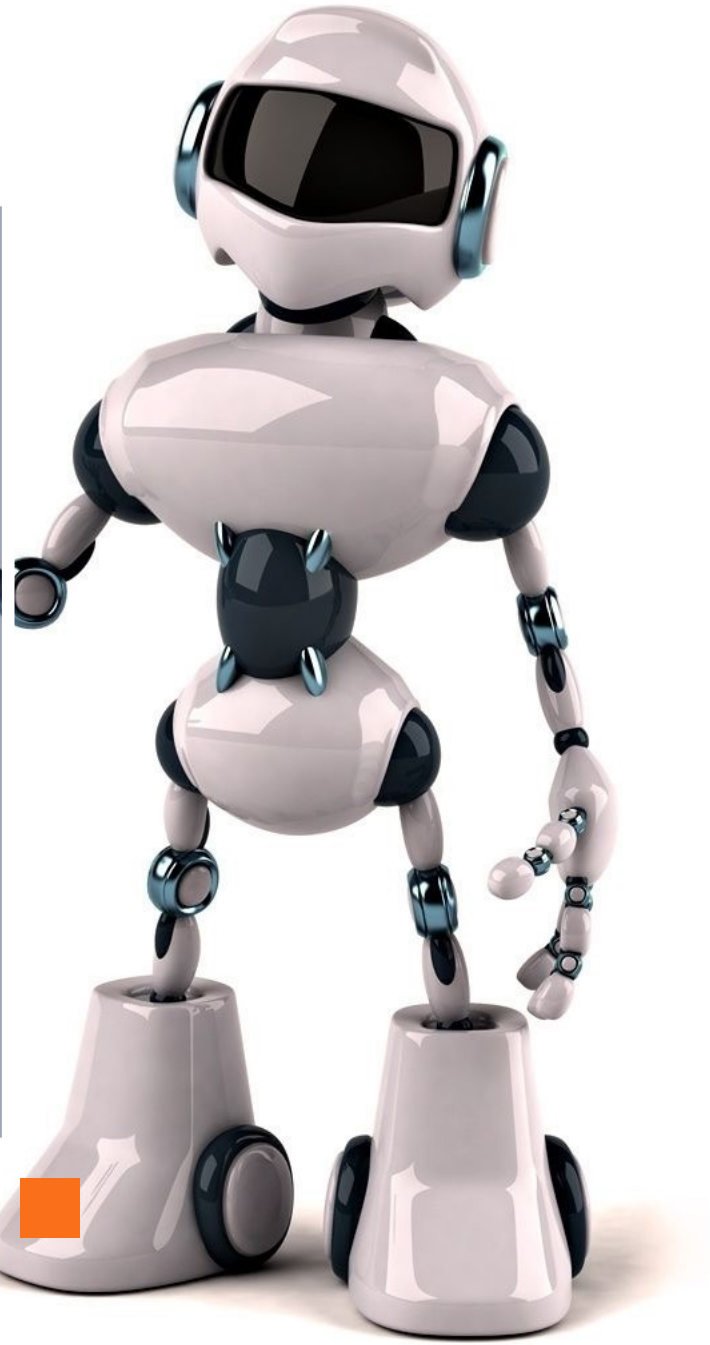
- rosrun image_view image_view image:=/rgbd_camera/rgb/image_raw
- rosrun image_view image_view image:=/rgbd_camera/ir/image_raw
- rosrun image_view image_view image:=/rgbd_camera/depth/image_raw



Robot. Sensor. Motor



Components. Hardware. Software



Robot. Motor. Sensor

- A robot consists of hardware and software components
- Hardware: mechanism, motors, gears, circuits, sensors
- Software
  - micro-controller firmware
  - application software: map building, planning, navigation, and environment perception => ROS
- ROS contains specialised software applications:
  - Robot packages (http://robots.ros.org/, http://wiki.ros.org/Robots)
  - Sensor packages (http://wiki.ros.org/Sensors)
  - Motor packages (http://wiki.ros.org/Motor%20Controller%20Drivers)

Robot packages

- Robot companies supporting ROS: Willow Garage, ROBOTIS, Yujin Robot, Fetch Robotics, Clearpath Robotics, Open Robotics (formerly OSRF)
- Around 200 robots supported by ROS (https://robots.ros.org/)
- Most known robots: PR2 and TurtleBot
- New additions:
  - for land: Husky, Warthog and Jackal
  - for water: Heron

PR2

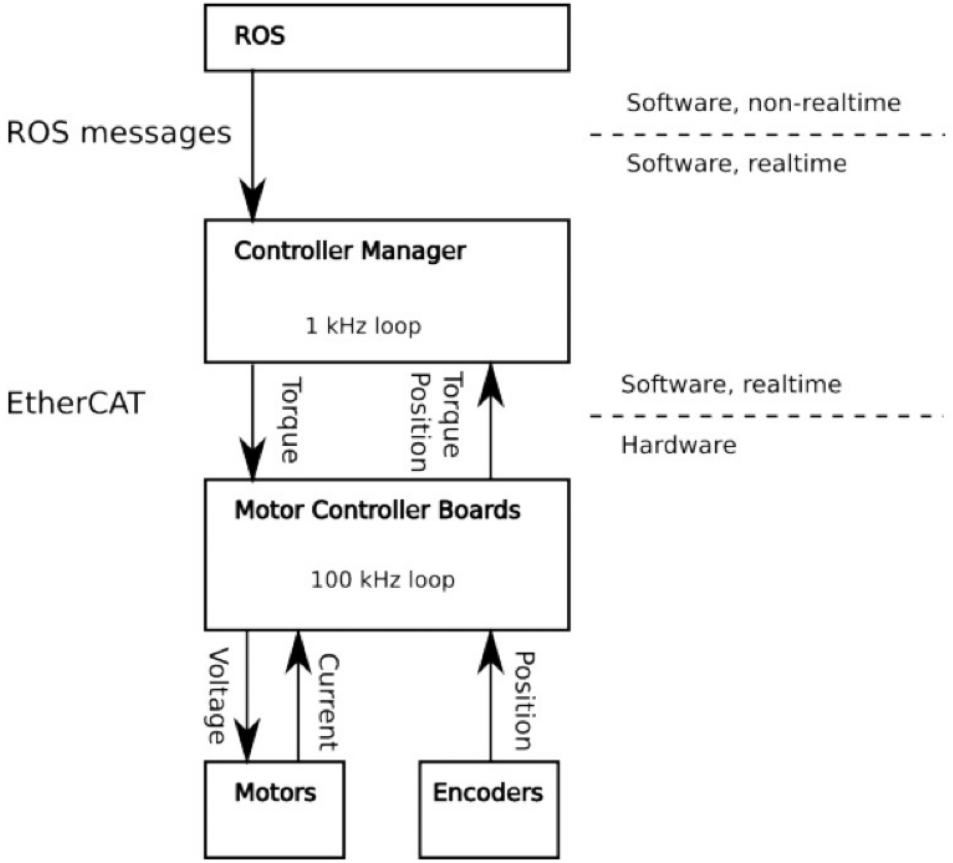
- PR2 (Personal Robot)
- created by Willow Garage
- maintained by Clearpath Robotics

- https://robots.ieee.org/robots/pr2/
- http://wiki.ros.org/Robots/PR2

PR2

- **Sensors**: Head with wide-angle and narrow-angle stereo cameras, Microsoft Kinect, and 5-megapixel camera. Torso with tilting Hokuyo UTM-30LX laser scanner and Microstrain 3DM-GX2 IMU. Arms with Ethernet cameras, three-axis accelerometer, and fingertip pressure sensors. Mobile base with Hokuyo UTM-30LX laser scanner.
- **Actuators**: 32 brushed DC motors
- **Power**: 1.3-kWh lithium-ion battery pack, 2 hours of operation
- **Computing**: Two Intel i7 Xeon quad-core processors, 24 GB of memory, 500 GB internal hard drive, and 1.5 TB removable hard drive.
- **Software**: ROS (Robot Operating System) and other open-source packages, including OpenCV vision libraries and PCL 3D point cloud processing libraries.
- **Degrees of Freedom**: 20 (Arm: 4 DoF x 2; Wrist: 3 DoF x 2; Gripper: 1 DoF x 2; Head pan/tilt: 2 DoF; Head laser tilt: 1 DoF; Telescoping spine: 1 DoF; mobile base actuators not included)
- **Cost**: 400.000 $

PR2 – Hardware to Software Integration



- EtherCAT = Ethernet for Control Automation Technology
- the realtime loop -> single process (pr2_ethercat package)
- https://www.clearpathrobotics.com/assets/downloads/pr2/pr2_manual_r321.pdf

TurtleBot

- created by Willow Garage
- low-cost personal robot
- based on Create (iRobot's cleaning robot platform)

- https://robots.ieee.org/robots/turtlebot/

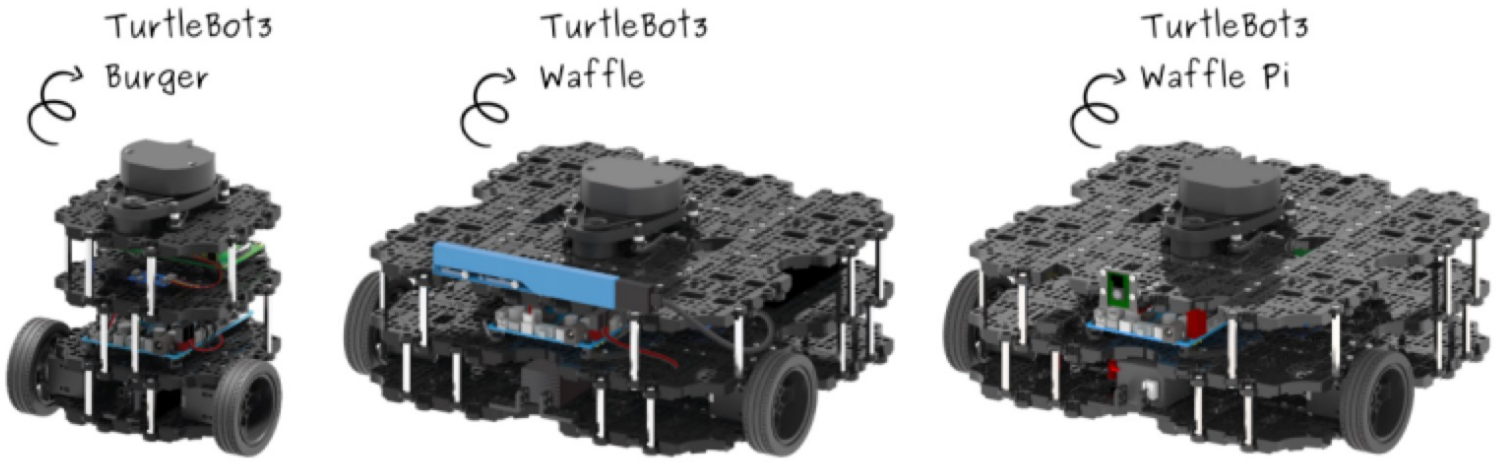
TurtleBot

- **Sensors**: Microsoft Kinect, gyro, and iRobot Create built-in cliff and bump sensors.
- **Actuators**: Two motors
- **Power**: 14.4-V nickel-metal hydride battery and laptop internal battery, 2 to 3 hours of operation
- **Computing**: Asus Eee PC 1215N
- **Software**: ROS (Robot Operating System)
- **Degrees of Freedom**: 2
- **Cost**: 1500 $

TurtleBot3

- Latest version of TurtleBot
- Multiple configurations (Burger, Waffle and Waffle Pi)

- https://robots.ieee.org/robots/turtlebot3/



TurtleBot3

- **Sensors**: HLDS 360-degrees laser distance sensor
- **Actuators**: Dynamixel XL430-W250 servo motors
- **Power**: 11.1-V 1800-mAh lithium-polymer battery, 2.5 hours of operation
- **Computing**: Compute module with Raspberry Pi 3 Model B. OpenCR board with 32-bit ARM Cortex-M7
- **Software**: ROS (Robot Operating System)
- **Degrees of Freedom**: 2
- **Cost**: 500 - 1500 $

Husky

- Created by Clearpath Robotics
- Tough unmanned ground vehicle

- https://robots.ieee.org/robots/husky/

Husky

- **Sensors**: On-board: IMU, battery voltage, motor currents, and wheel odometry sensors
- **Actuators**: DC motors
- **Power**: 24-V 20-Ah sealed lead-acid battery, 8 hours of operation. Onboard: 5-V, 12-V, and 24-V power supply for user payload
- **Computing**: Computing and sensor payloads according to customer requests.
- **Software**: ROS, LabVIEW, Player/Stage, custom software in C++ and Python
- **Degrees of Freedom**: 2
- **Cost**: N/A

- **Official Website**: http://www.clearpathrobotics.com

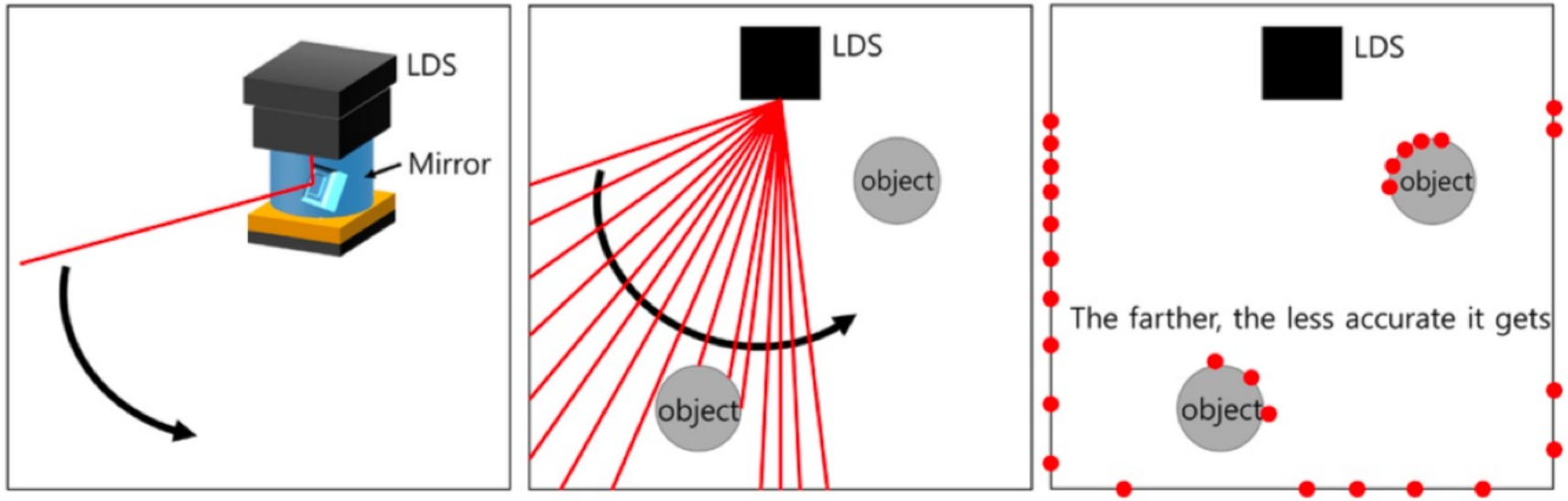
Sensor packages

- extract meaningful information from environment: location, space, weather, sound, vibrations, gas, etc.
- used to close the control loop
- Most common types:
  - Range finders (1D and 2D) - Laser Distance Sensor (LDS)
  - Cameras
  - 3D sensors - RealSense, Kinect, Xction

- ROS message package: http://wiki.ros.org/sensor_msgs
- http://wiki.ros.org/Sensors

Range finders (LDS)

- also known as Light Detection And Ranging (LiDAR), Laser Range Finder (LRF) and Laser Scanner
- measures distance to an object using a laser



Range finders (LDS)

- Packages: hls_lfcd_lds_driver
- Messages: sensor_msgs/LaserScan.msg
- Parameters: port and frame_id

- http://wiki.ros.org/hls_lfcd_lds_driver

```
# Single scan from a planar laser range-finder
#
# If you have another ranging device with different behavior (e.g. a sonar
# array), please find or create a different message, since applications
# will make fairly laser-specific assumptions about this data

Header header            # timestamp in the header is the acquisition time of
                         # the first ray in the scan.
                         #
                         # in frame frame_id, angles are measured around
                         # the positive Z axis (counterclockwise, if Z is up)
                         # with zero angle being forward along the x axis

float32 angle_min        # start angle of the scan [rad]
float32 angle_max        # end angle of the scan [rad]
float32 angle_increment  # angular distance between measurements [rad]

float32 time_increment   # time between measurements [seconds] - if your scanner
                         # is moving, this will be used in interpolating position
                         # of 3d points
float32 scan_time        # time between scans [seconds]

float32 range_min        # minimum range value [m]
float32 range_max        # maximum range value [m]

float32[] ranges         # range data [m] (Note: values < range_min or > range_max should be discarded)
float32[] intensities    # intensity data [device-specific units].  If your
                         # device does not provide intensities, please leave
                         # the array empty.
```

Camera

- USB Video Device Class (UVC)
- Packages: libuvc-camera, uvc-camera, usb-cam, camera1394, …
- Messages: sensor_msgs/Image.msg

- http://wiki.ros.org/Sensors/Cameras

```
# This message contains an uncompressed image
# (0, 0) is at top-left corner of image
#

Header header        # Header timestamp should be acquisition time of image
                     # Header frame_id should be optical frame of camera
                     # origin of frame should be optical center of camera
                     # +x should point to the right in the image
                     # +y should point down in the image
                     # +z should point into to plane of the image
                     # If the frame_id here and the frame_id of the CameraInfo
                     # message associated with the image conflict
                     # the behavior is undefined

uint32 height         # image height, that is, number of rows
uint32 width          # image width, that is, number of columns

# The legal values for encoding are in file src/image_encodings.cpp
# If you want to standardize a new string format, join
# ros-users@lists.sourceforge.net and send an email proposing a new encoding.

string encoding       # Encoding of pixels -- channel meaning, ordering, size
                      # taken from the list of strings in include/sensor_msgs/image_encodings.h

uint8 is_bigendian    # is this data bigendian?
uint32 step           # Full row length in bytes
uint8[] data          # actual matrix data, size is (step * rows)
```

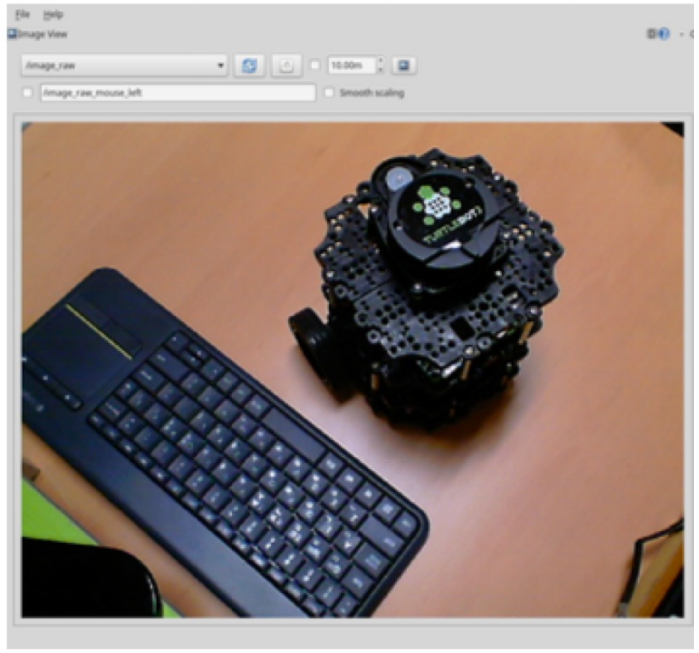
Camera tools

- Using **image_view** node

rosrun image_view image_view image:=/image_raw

- Using **rqt_image_view** node

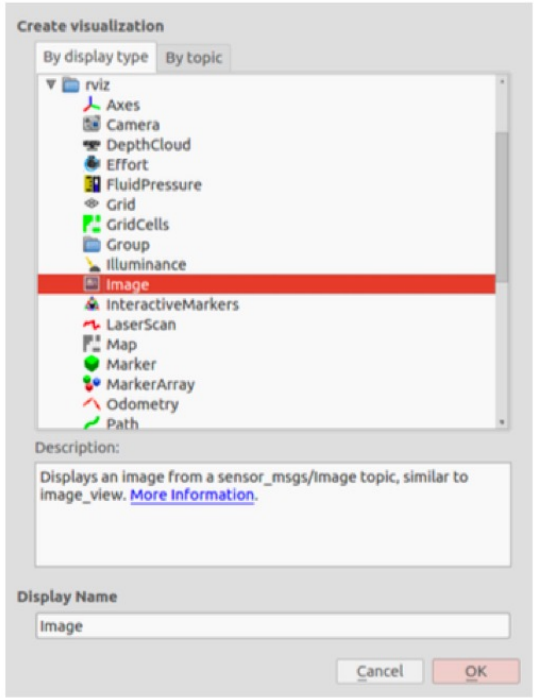
rqt_image_view image:=/image_raw

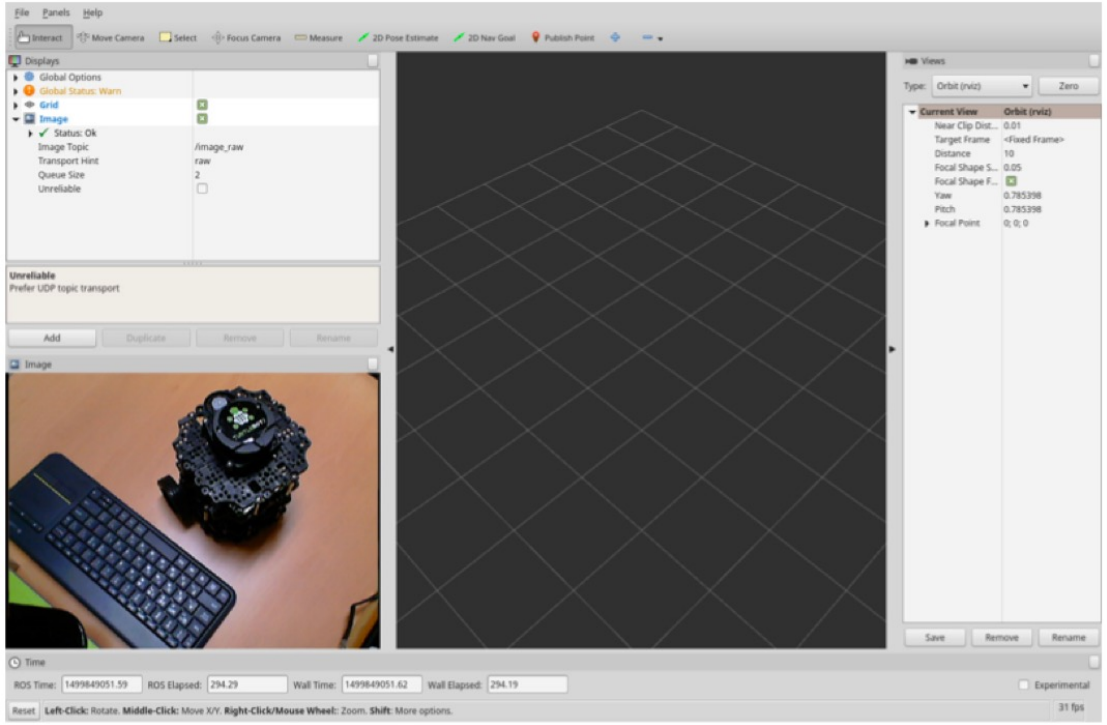


Camera tools

- Using **RViz** node

  rviz





Camera calibration

- process used to calculate each camera’s unique parameters
- Package: ros-kinetic-camera-calibration
- Usually performed using a chessboard image

Depth Cameras

- RGB-D camera - adds additional information to a point (the depth)
- Three types of camera
  - Time of Flight (ToF)
  - Structured Light
  - Stereo

ToF Cameras

- Calculate the distance based on the time required for a radiated infrared ray to return
- Panasonic D-IMager, MESA Imaging SwissRanger, PMDtechnologies CamBoard, Microsoft Kinect 2 (left to right)

Structured Light Cameras

- Uses coherent radiation pattern
- Projects a known pattern (grid or horizontal bars) on to a scene.
- Microsoft Kinect, Asus Xtion, Primesense Carmine, Occipital Structure Sensor (left to right)

Stereo Cameras

- Distance calculated by binocular parallax (human eyes)
- Bumblebee, OjOcamStereo, RealSense (left to right)

3D Sensors

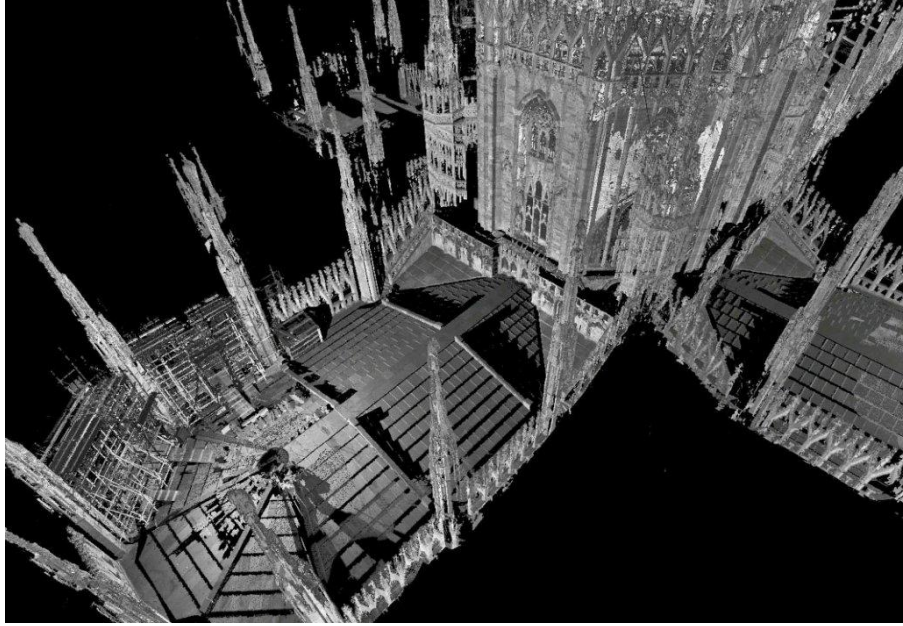
- Packages: ros-kinect, librealsense
- Messages: sensor_msgs/PointCloud2.msg

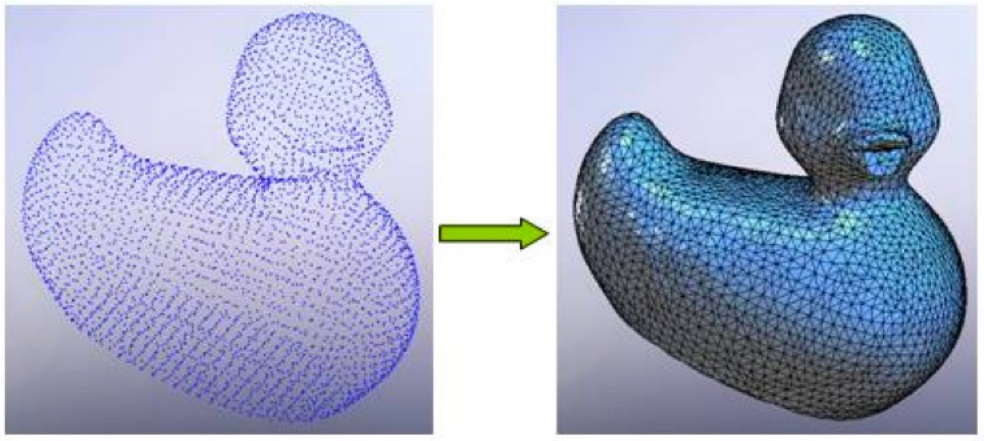
- http://wiki.ros.org/Sensors/3D%20Sensors

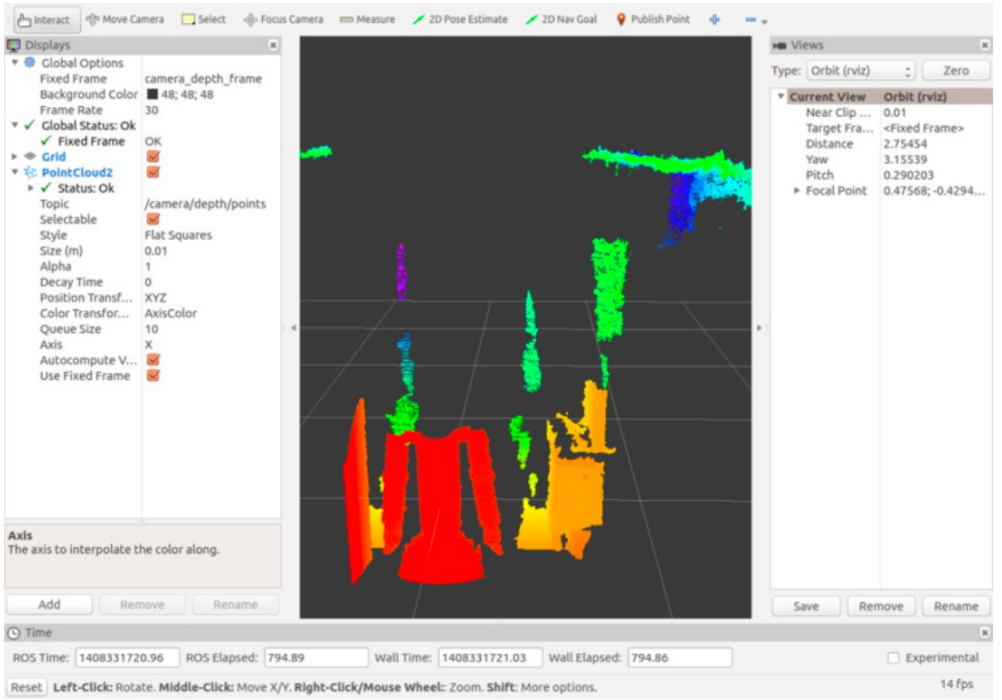
```
# This message holds a collection of N-dimensional points, which may
# contain additional information such as normals, intensity, etc. The
# point data is stored as a binary blob, its layout described by the
# contents of the "fields" array.

# The point cloud data may be organized 2d (image-like) or 1d
# (unordered). Point clouds organized as 2d images may be produced by
# camera depth sensors such as stereo or time-of-flight.

# Time of sensor data acquisition, and the coordinate frame ID (for 3d
# points).
Header header

# 2D structure of the point cloud. If the cloud is unordered, height is
# 1 and width is the length of the point cloud.
uint32 height
uint32 width

# Describes the channels and their layout in the binary data blob.
PointField[] fields

bool    is_bigendian # Is this data bigendian?
uint32  point_step   # Length of a point in bytes
uint32  row_step     # Length of a row in bytes
uint8[] data         # Actual point data, size is (row_step*height)

bool is_dense        # True if there are no invalid points
```

Point clouds

- Point Cloud Library (PCL) - a standalone, large scale, open project for 2D/3D image and point cloud processing.
- OpenNI library
- usually produced by 3D scanners
- can also be used to generate the mesh representation of an object







Motor packages

- interface to a motor controller or servo controller
- most used: Dynamixel
- packages: dynamixel_motor, arbotix, dynamixel_workbench

- http://wiki.ros.org/Motor%20Controller%20Drivers

Thank you

Question Time