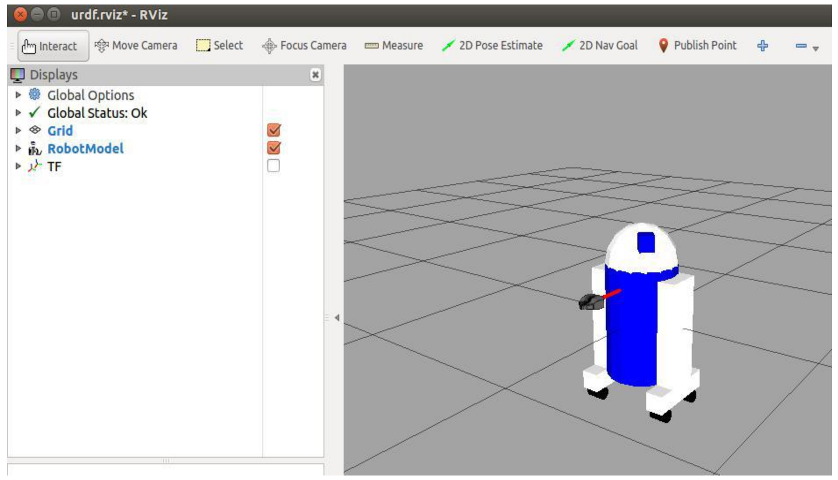
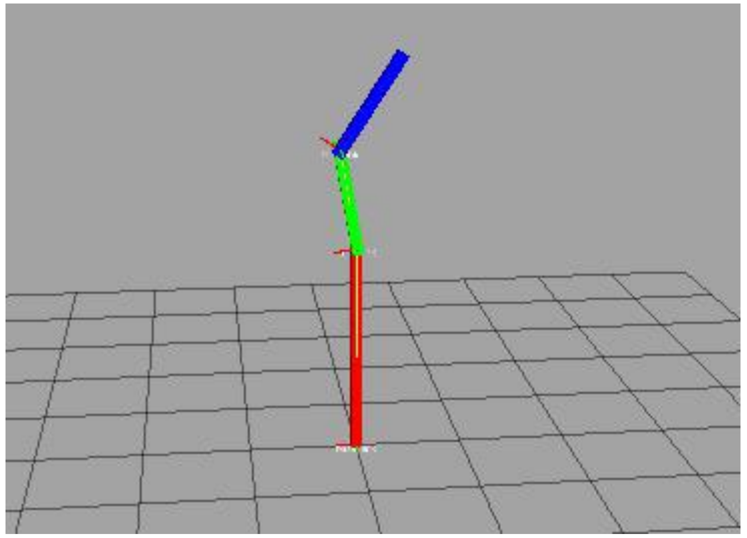
Learning ROS – Robot operating system | The easy way

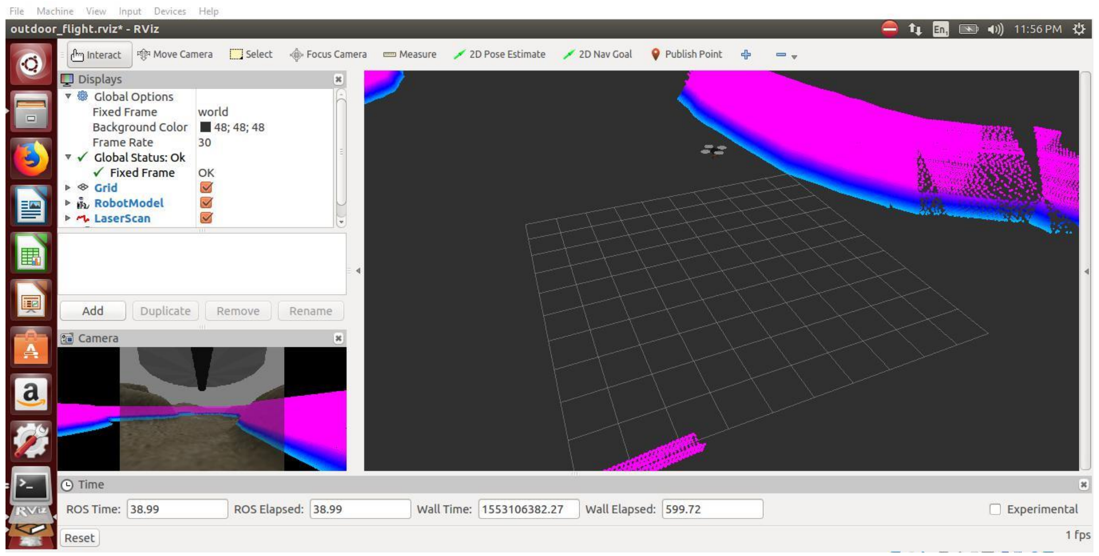
Anirban Ghatak, Founder MieRobot

Our Approach

- Learn by doing
- Use a *fast lane* towards *applying* ROS
- Avoid complex theory where it is not needed
- Simulate 7 robots in this course







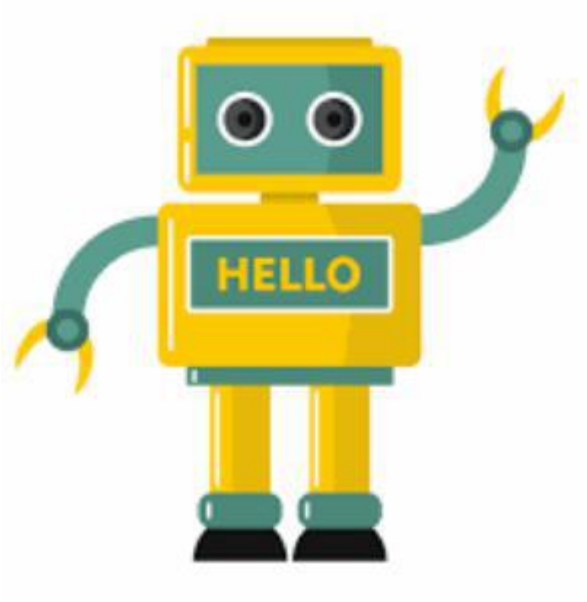
Global Options
Fixed Frame world
Background Color 48; 48; 48
Frame Rate 30
Global Status: Ok
Fixed Frame OK
Grid ✓
RobotModel ✓
LaserScan ✓

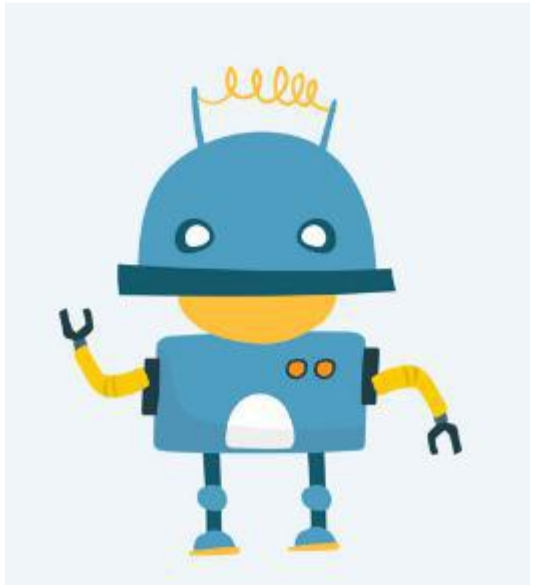
ROS Time: 38.99 ROS Elapsed: 38.99 Wall Time: 1553106382.27 Wall Elapsed: 599.72 Experimental

1 fps

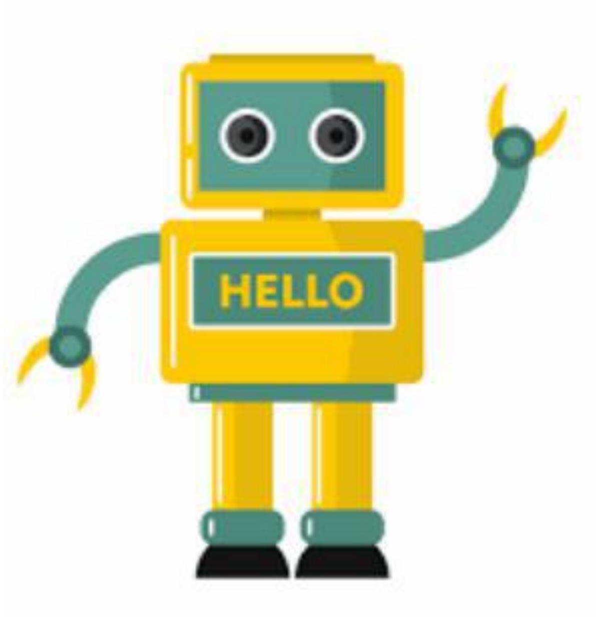
Dive straight into ROS
Installation can wait – See this video first

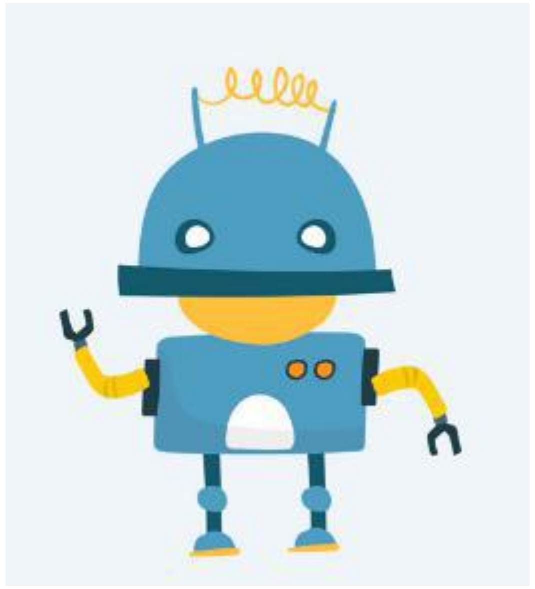
- **Talker chatter & listener application**
- **ROSCORE**
- **ROSNODE**
- **ROSRUN**
- **ROSTOPIC**
- **ROS MESSAGE**





Watch the demo – I will do slowly





What is ROS?

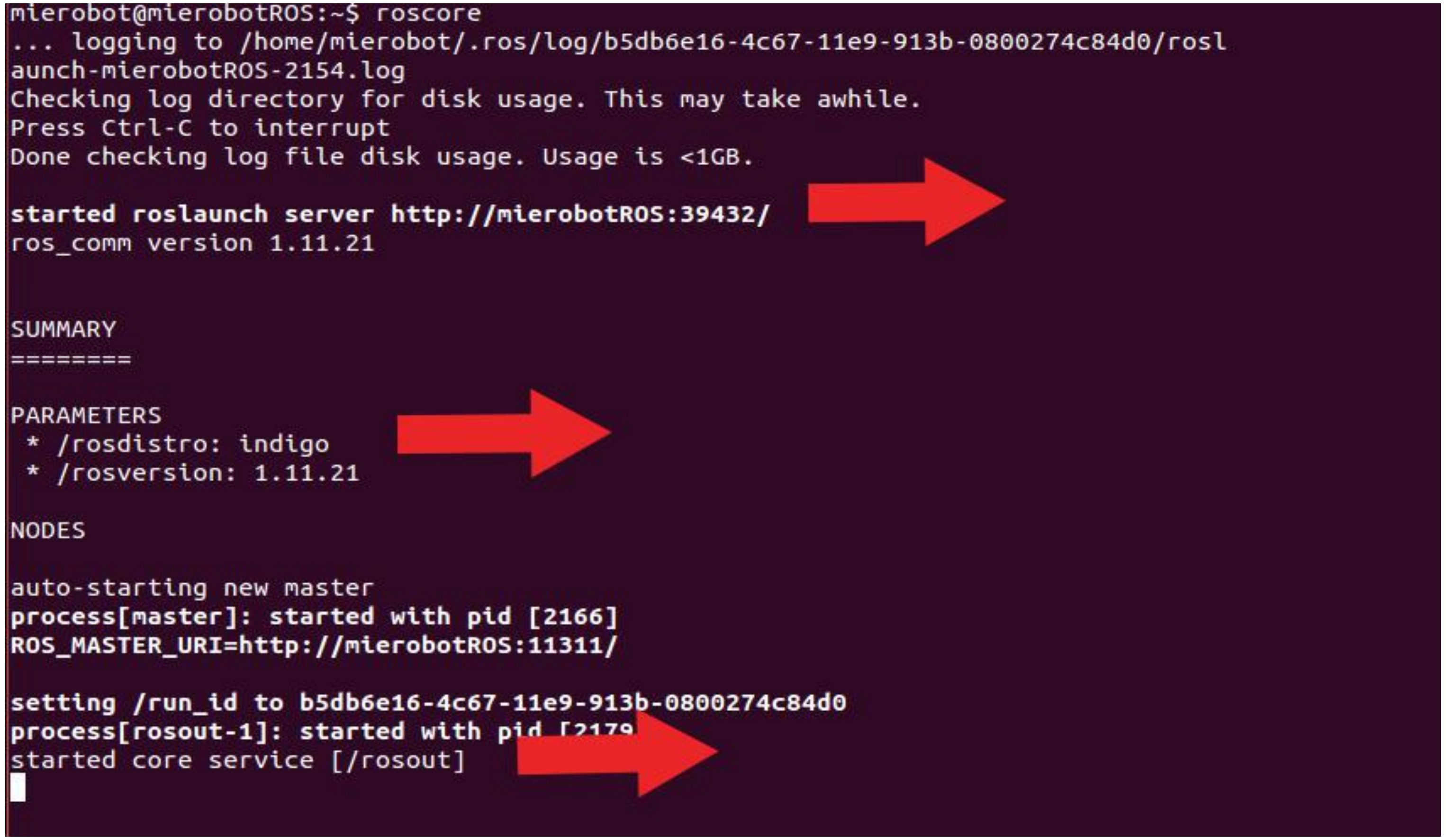
- Meta Operating system but needs a host OS
- ROS is multilingual, peer to peer, distributed, light weight
- ROS is Open Source and Free under BSD license
- ROS is maintained by OSRF (Open Source Robotics Foundation)
- Many famous robots use ROS like Hector, TurtleBot, Baxter, Crazy file and Bebop
- You can simulate real life robots in ROS

The Robots we will simulate

1. A custom 2 wheeled robot
2. Turtle SIM simulation
3. Running R2D2 Star wars robot
4. Running the TurtleBot robot
5. Simulation of a robotic arm
6. Use XBOX controlled Turtle Sim
7. Use XBOX controlled Hector Drone

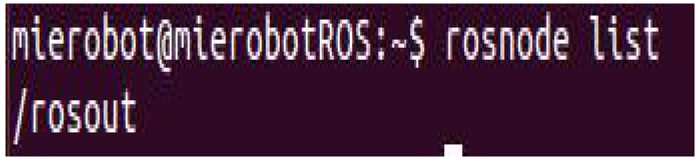
ROSCORE http://wiki.ros.org/roscore

- ✓ **ROSCORE** is the mandatory service to be running for ROS to work
- ✓ You can run by the command roscore
- ✓ It starts up: a ROS master, a ROS parameter server and a **rosout** log node
- ✓ Parameter server stores & retrives parameter at runtime
- ✓ **ROSCORE** manages communication between nodes
- ✓ Every node registers with the ROS core during start-up
- ✓ **ROSCORE** is the Master



```
mierobot@mierobotROS:~$ roscore
... logging to /home/mierobot/.ros/log/b5db6e16-4c67-11e9-913b-0800274c84d0/rosl
aunch-mierobotROS-2154.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://mierobotROS:39432/
ros_comm version 1.11.21


SUMMARY
========

PARAMETERS
 * /rosdistro: indigo
 * /rosversion: 1.11.21

NODES

auto-starting new master
process[master]: started with pid [2166]
ROS_MASTER_URI=http://mierobotROS:11311/

setting /run_id to b5db6e16-4c67-11e9-913b-0800274c84d0
process[rosout-1]: started with pid [2179
started core service [/rosout]
```

ROS nodes http://wiki.ros.org/Nodes



```
mierobot@mierobotROS:~$ rosnode list
/rosout
```

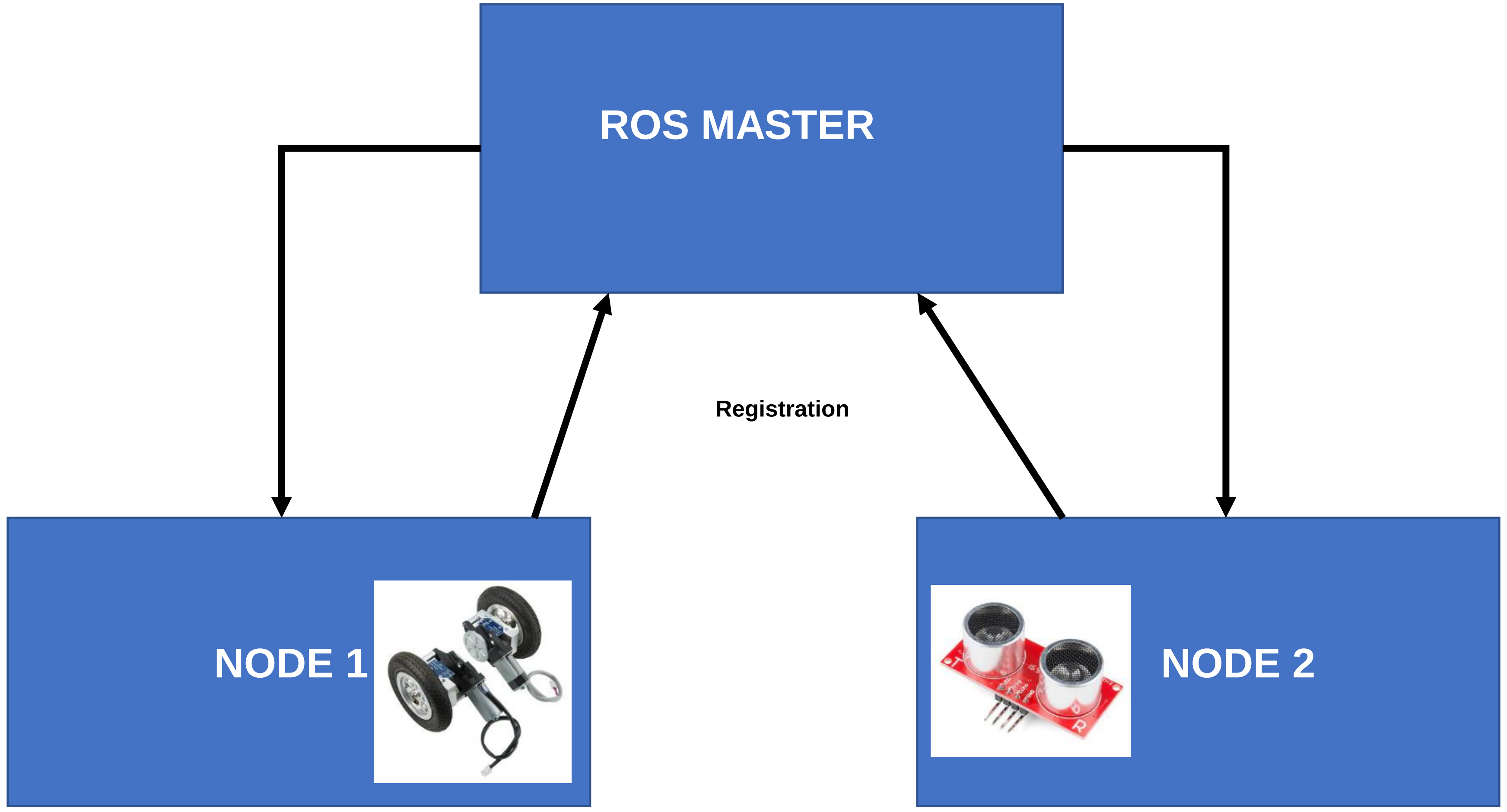
- ✓ One single use or purpose or program
- ✓ Node is an executable
- ✓ Node is compiled as an individual
- ✓ Node are organised in package

Advantages:

- ▪ Move away from a monolithic structure (Like single program MCU)
- ▪ Fault tolerant robots
- ▪ Reduce complexity
- ▪ Uses ROS client library like ROSCPP/ROSPY

Example: You have 1 node for running motors & another node for an ultrasonic sensor. The motor is coded in C++ & Sensor is coded in Python

Relation between ROSMASTER with ROSNODE



What is catkin build system

- ✓ Catkin is the ROS build system
- ✓ A build system generates interfaces, executables & libraries
- ✓ **Catkin_make** is the command used to make a catkin workspace
- ✓ Workspace the is root folder for our ROS work

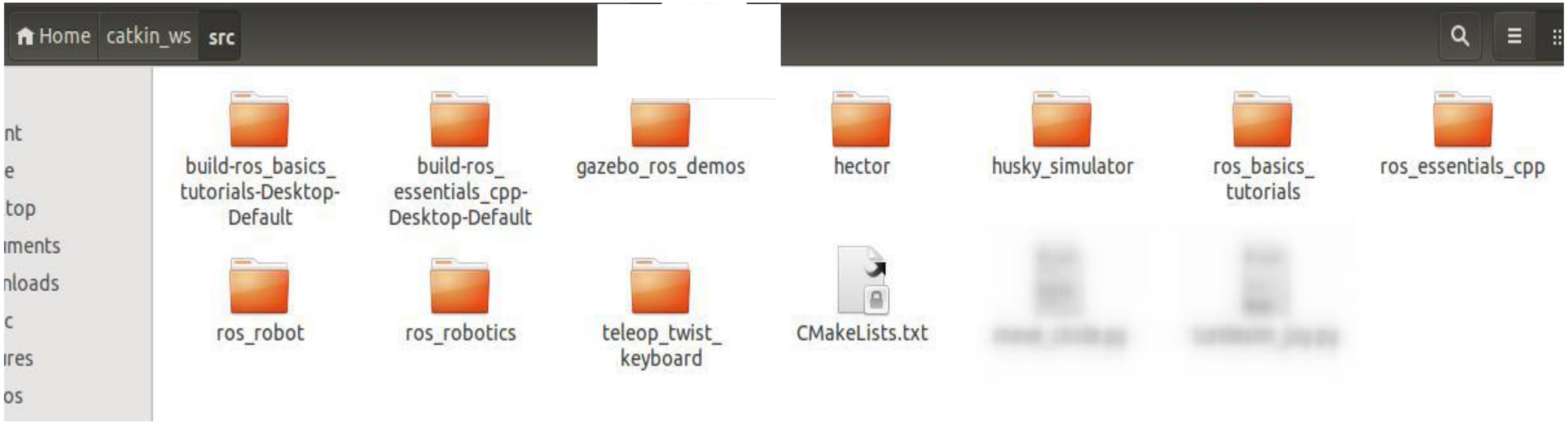
build
devel
install
src

build: cmake (manages build process – see https://cmake.org/overview/ for more) is invoked here (**Do not even touch it**)
devel: build targets are placed here (**Do not even touch it**)
install: has etc,lib and scripts (**Do not use it- avoid**)
src: Our folder to work on our source code (**Always use this)**

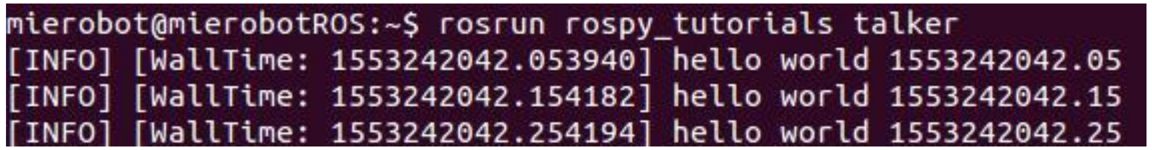
ROS Package & ROSRUN

http://wiki.ros.org/Packages http://wiki.ros.org/rosbash#rosrun

- ✓ **Packages** are directory created from ROS_PACKAGE_PATH
- ✓ Directory to store codes, configs, dataset, library
- ✓ Created under src for custom package
- ✓ **Command**: **catkin build MyPackage**



rosrun allows you to run an executable in an arbitrary package from anywhere without having to give its full path

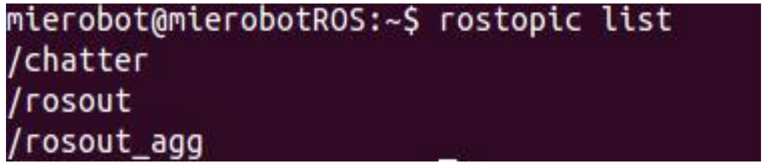


```
mierobot@mierobotROS:~$ rosrun rospy_tutorials talker
[INFO] [WallTime: 1553242042.053940] hello world 1553242042.05
[INFO] [WallTime: 1553242042.154182] hello world 1553242042.15
[INFO] [WallTime: 1553242042.254194] hello world 1553242042.25
```

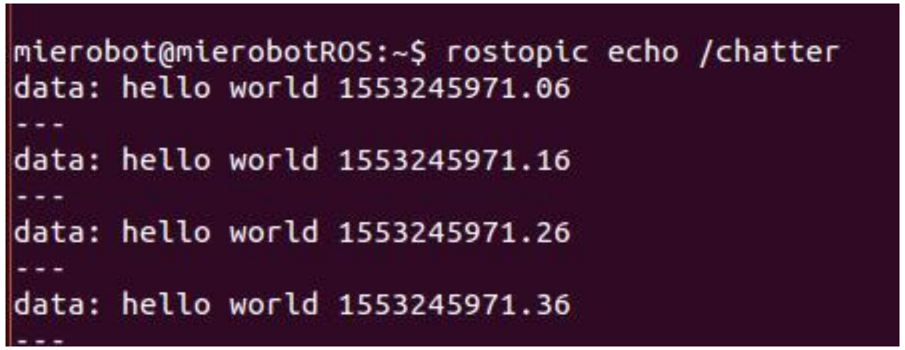
ROS Topic & message

ROS Topic:

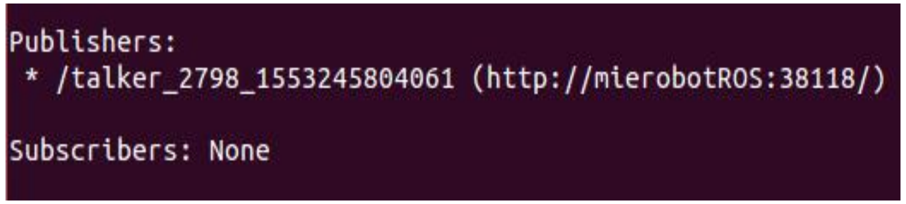
✓ A bus for messages
✓ Nodes communicate via topic
✓ 1:N mapping (Publisher: Subscriber)
✓ Command: **rostopic list**
✓ Command: **rostopic echo /chatter**



```
mierobot@mierobotROS:~$ rostopic list
/chatter
/rosout
/rosout_agg
```



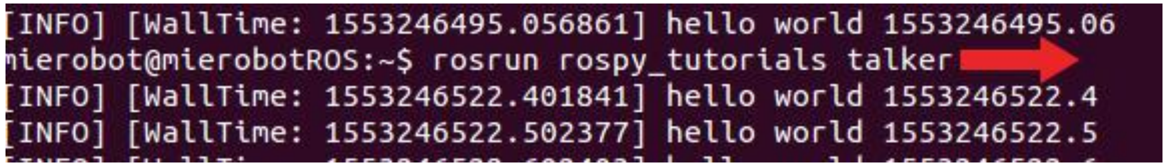
```
mierobot@mierobotROS:~$ rostopic echo /chatter
data: hello world 1553245971.06
---
data: hello world 1553245971.16
---
data: hello world 1553245971.26
---
data: hello world 1553245971.36
---
```



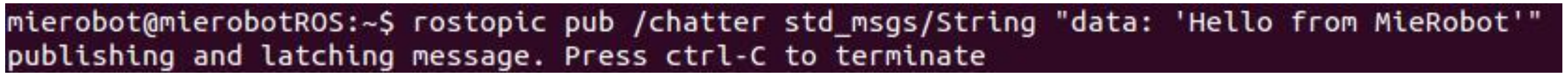
```
Publishers:
 * /talker_2798_1553245804061 (http://mierobotROS:38118/)

Subscribers: None
```

Message:

✓ Nodes communicate with each other by publishing messages to topics
✓ Nested structure of int,floats,Boolean,strings etc
✓ Defined in *.msg files

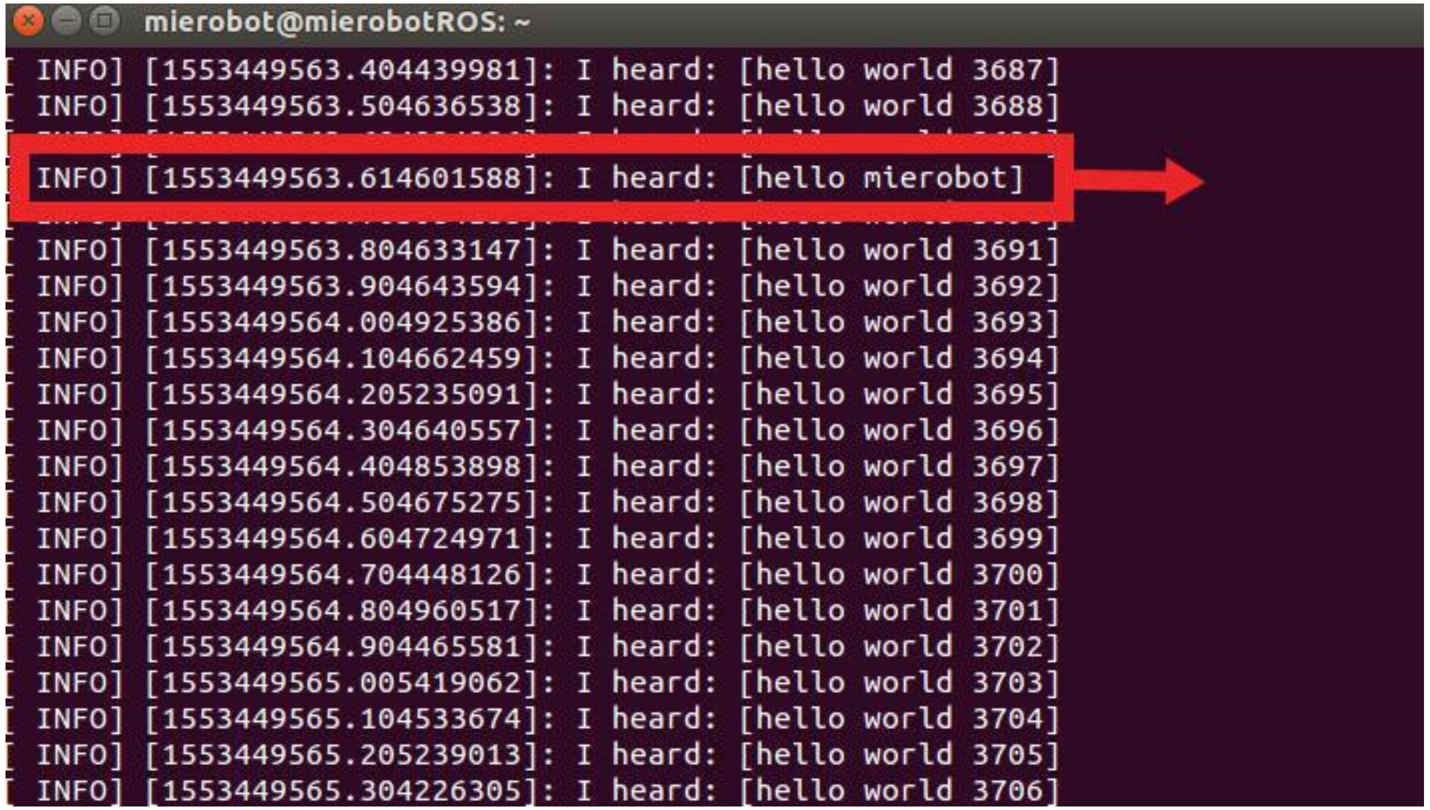


```
[INFO] [WallTime: 1553246495.056861] hello world 1553246495.06
mierobot@mierobotROS:~$ rosrun rospy_tutorials talker
[INFO] [WallTime: 1553246522.401841] hello world 1553246522.4
[INFO] [WallTime: 1553246522.502377] hello world 1553246522.5
```



```
mierobot@mierobotROS:~$ rostopic pub /chatter std_msgs/String "data: 'Hello from MieRobot'"
publishing and latching message. Press ctrl-C to terminate
```

Publishing own message to a chatter topic



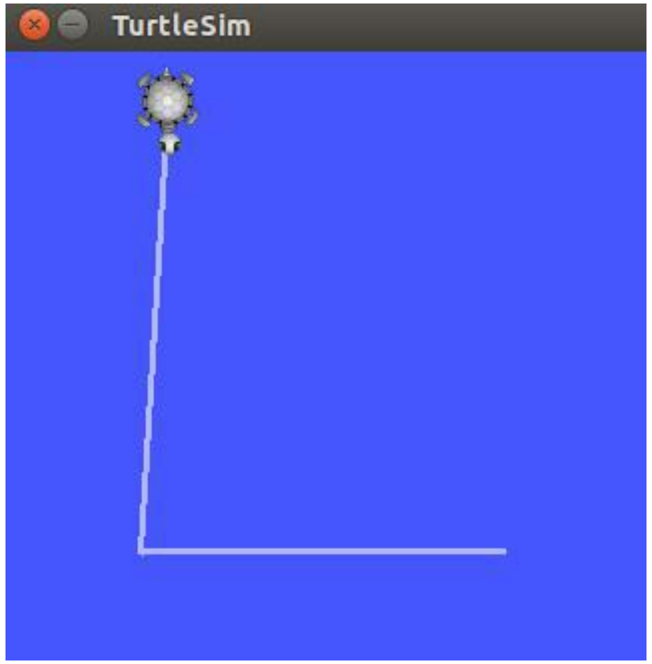
```
mierobot@mierobotROS: ~
[ INFO] [1553449563.404439981]: I heard: [hello world 3687]
[ INFO] [1553449563.504636538]: I heard: [hello world 3688]
[illegible]
[ INFO] [1553449563.614601588]: I heard: [hello mierobot]
[illegible]
[ INFO] [1553449563.804633147]: I heard: [hello world 3691]
[ INFO] [1553449563.904643594]: I heard: [hello world 3692]
[ INFO] [1553449564.004925386]: I heard: [hello world 3693]
[ INFO] [1553449564.104662459]: I heard: [hello world 3694]
[ INFO] [1553449564.205235091]: I heard: [hello world 3695]
[ INFO] [1553449564.304640557]: I heard: [hello world 3696]
[ INFO] [1553449564.404853898]: I heard: [hello world 3697]
[ INFO] [1553449564.504675275]: I heard: [hello world 3698]
[ INFO] [1553449564.604724971]: I heard: [hello world 3699]
[ INFO] [1553449564.704448126]: I heard: [hello world 3700]
[ INFO] [1553449564.804960517]: I heard: [hello world 3701]
[ INFO] [1553449564.904465581]: I heard: [hello world 3702]
[ INFO] [1553449565.005419062]: I heard: [hello world 3703]
[ INFO] [1553449565.104533674]: I heard: [hello world 3704]
[ INFO] [1553449565.205239013]: I heard: [hello world 3705]
[ INFO] [1553449565.304226305]: I heard: [hello world 3706]
```

Section: Running the TurtleSim with keyboard

Anirban Ghatak, MieRobot

What is Turtle Robot & TurtleSim?

Installation command: sudo apt-get install ros-indigo-turtlesim

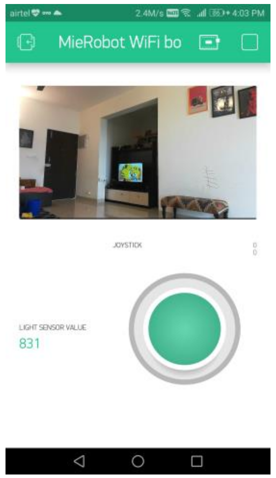


Source: https://www.turtlebot.com/

What is Teleop in robotics?

Teleoperation (or remote operation) indicates operation of a system or machine at a distance. It is similar in meaning to the phrase "remote control" but is usually encountered in research, academic and technical environments.

It is most commonly associated with robotics and mobile robots but can be applied to a whole range of circumstances in which a device or machine is operated by a person from a distance.



What is rqt?
http://wiki.ros.org/rqt

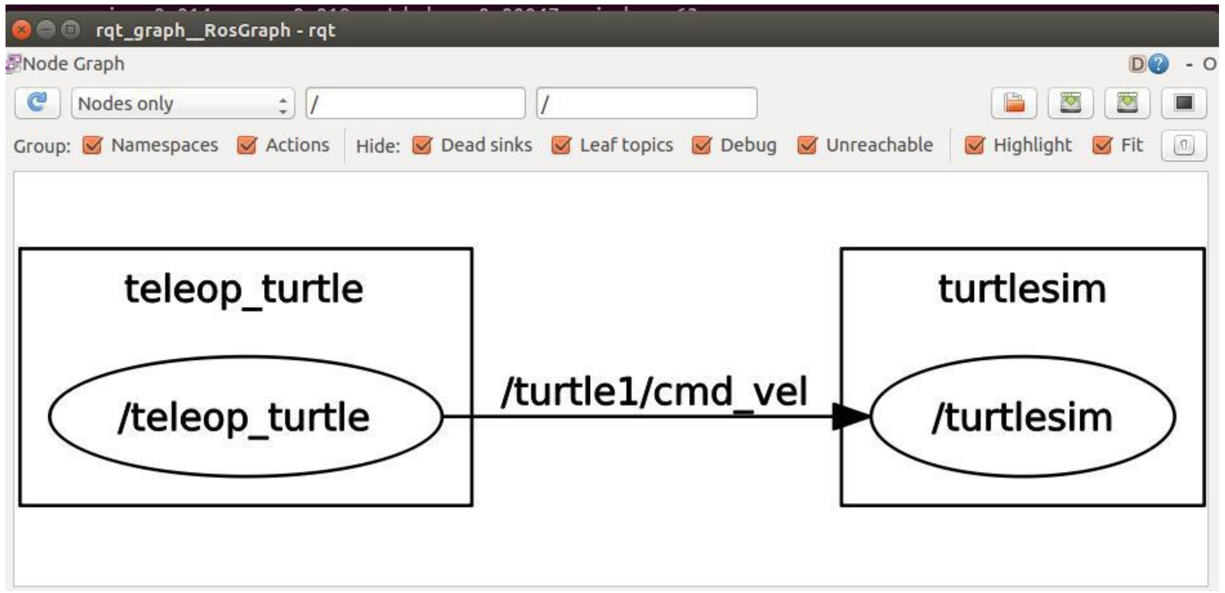
rqt is a Qt-based framework for GUI development for ROS.

It consists of three parts/metapackages

- ✓ rqt
- ✓ rqt_common_plugins - ROS backend tools suite that can be used on/off of robot runtime.
- ✓ rqt_robot_plugins - Tools for interacting with robots during their runtime.

Commands:

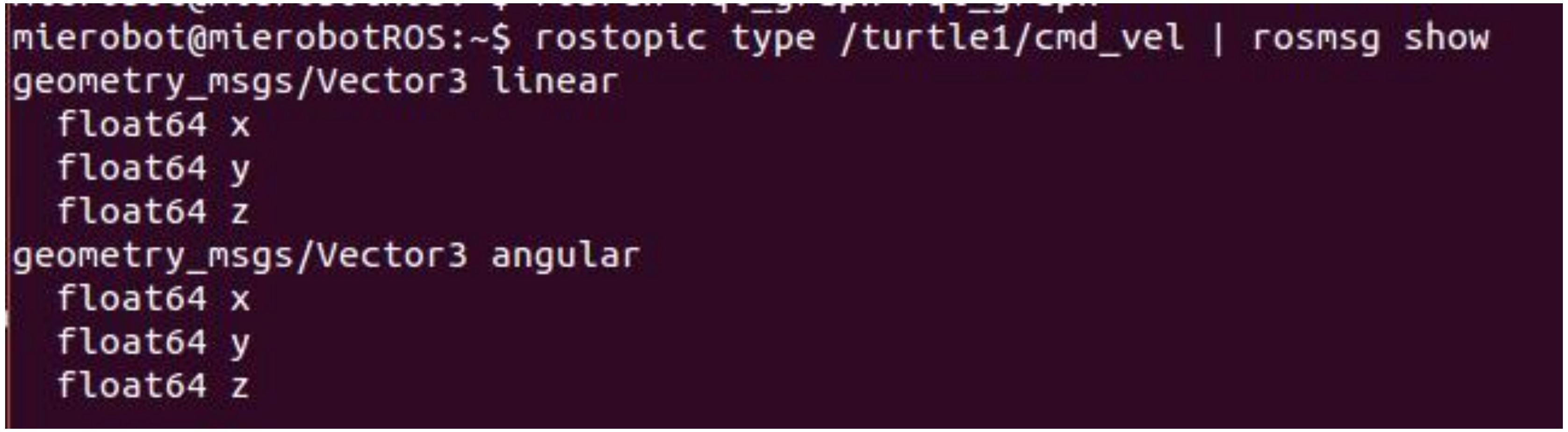
```
sudo apt-get install ros-indigo-rqt
rosrun rqt_graph rqt_graph
```



rqt_graph__RosGraph - rqt
Node Graph
Nodes only
/
/
Group: Namespaces Actions Hide: Dead sinks Leaf topics Debug Unreachable Highlight Fit
teleop_turtle
/teleop_turtle
/turtle1/cmd_vel
turtlesim
/turtlesim

What is Twist & Geometry msg? http://wiki.ros.org/geometry_msgs

Geometry msg:
geometry_msgs provides messages for common geometric primitives such as points, vectors, and poses.



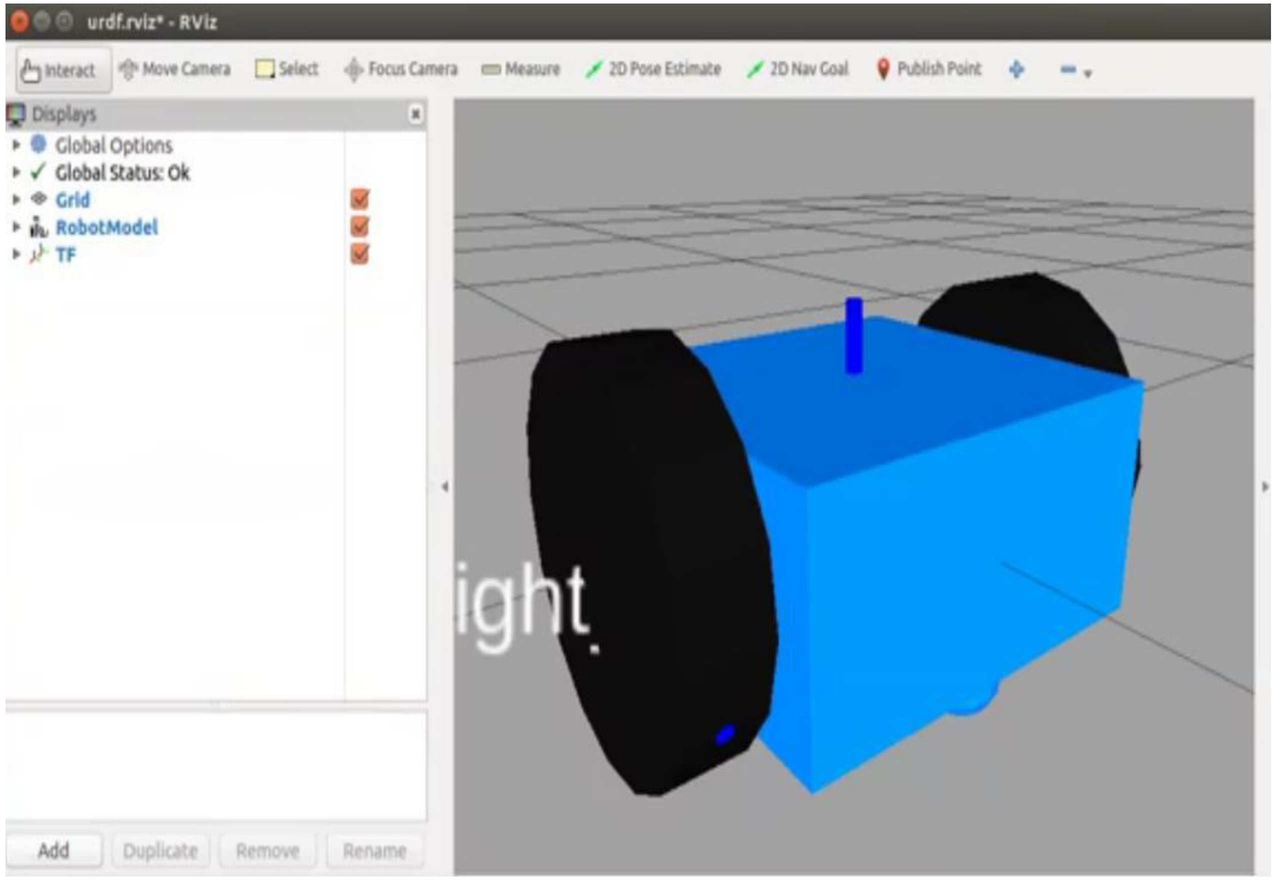
```
mierobot@mierobotROS:~$ rostopic type /turtle1/cmd_vel | rosmsg show
geometry_msgs/Vector3 linear
  float64 x
  float64 y
  float64 z
geometry_msgs/Vector3 angular
  float64 x
  float64 y
  float64 z
```

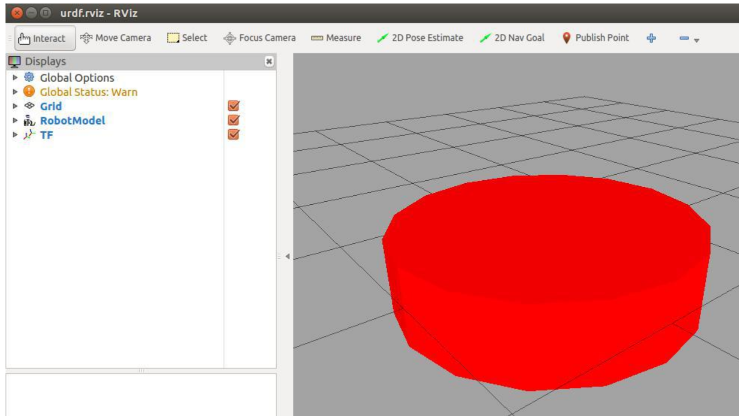
Twist:
This expresses velocity in free space broken into its linear and angular parts.

Section : Creating a 2 Wheeled robot in ROS

Anirban Ghatak, MieRobot

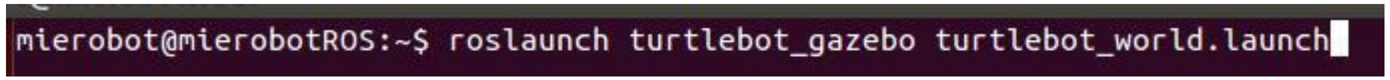
Our section goal





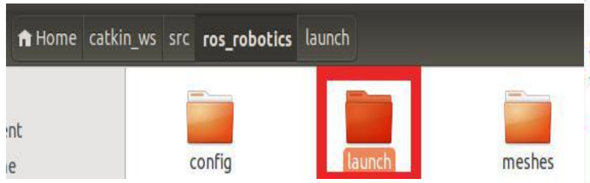
What is a Launch file
http://wiki.ros.org/roslaunch

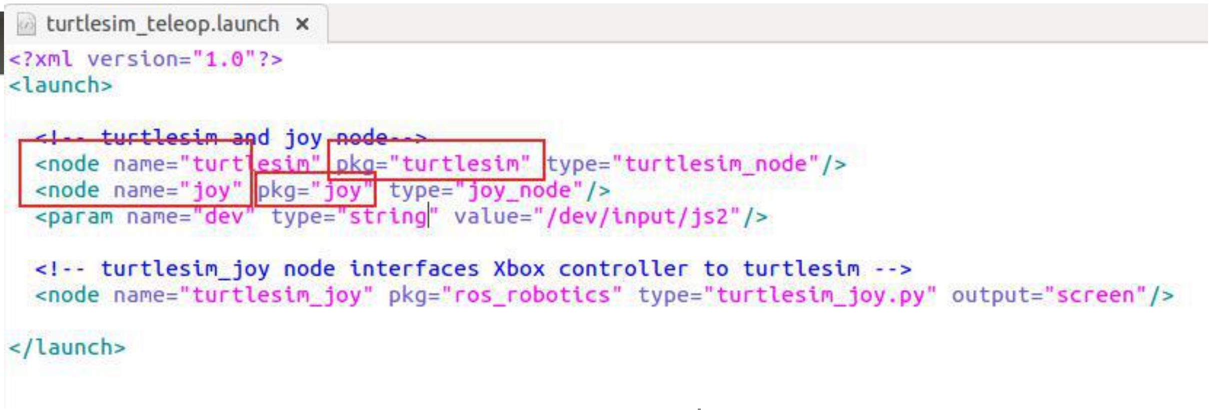
- **roslaunch** is a tool for easily launching multiple ROS nodes locally and remotely with ssh
- **roslaunch** is an xml file
- **Command:** **roslaunch package_name file.launch**



```
mierobot@mierobotROS:~$ roslaunch turtlebot_gazebo turtlebot_world.launch
```

Package name **Launch file name**



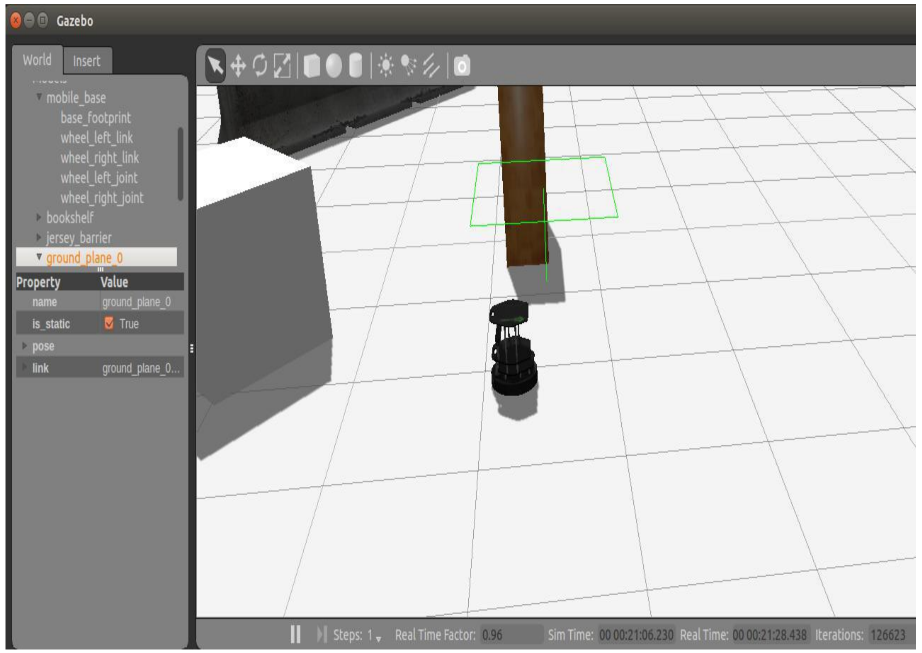


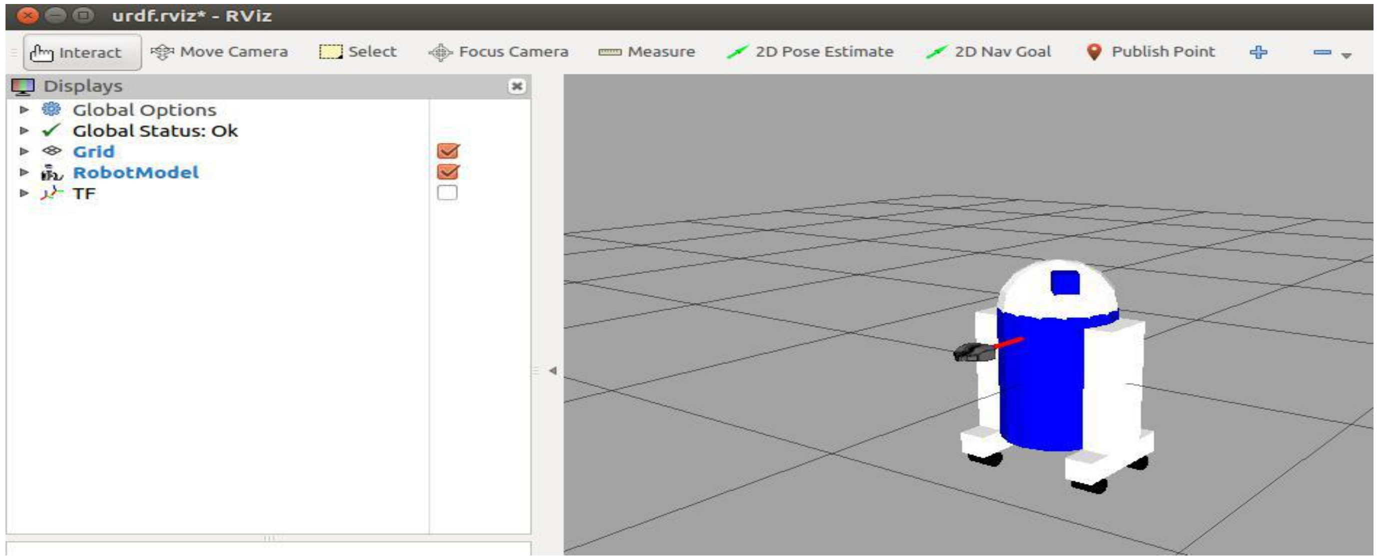
turtlesim_teleop.launch

```
<?xml version="1.0"?>
<launch>

  <!-- turtlesim and joy node-->
  <node name="turtlesim" pkg="turtlesim" type="turtlesim_node"/>
  <node name="joy" pkg="joy" type="joy_node"/>
  <param name="dev" type="string" value="/dev/input/js2"/>

  <!-- turtlesim_joy node interfaces Xbox controller to turtlesim -->
  <node name="turtlesim_joy" pkg="ros_robotics" type="turtlesim_joy.py" output="screen"/>

</launch>
```

What is Rviz?

- ✓ **Rviz is a 3D visualizer for displaying sensor data and state information from ROS**
- ✓ **Subscribe to topic and visualise message contents**
- ✓ **Camera angles**
- ✓ **Interactive tools for data publishing**
- ✓ **Save and load Rviz configs**
- ✓ **Command: rosrun rviz rviz**



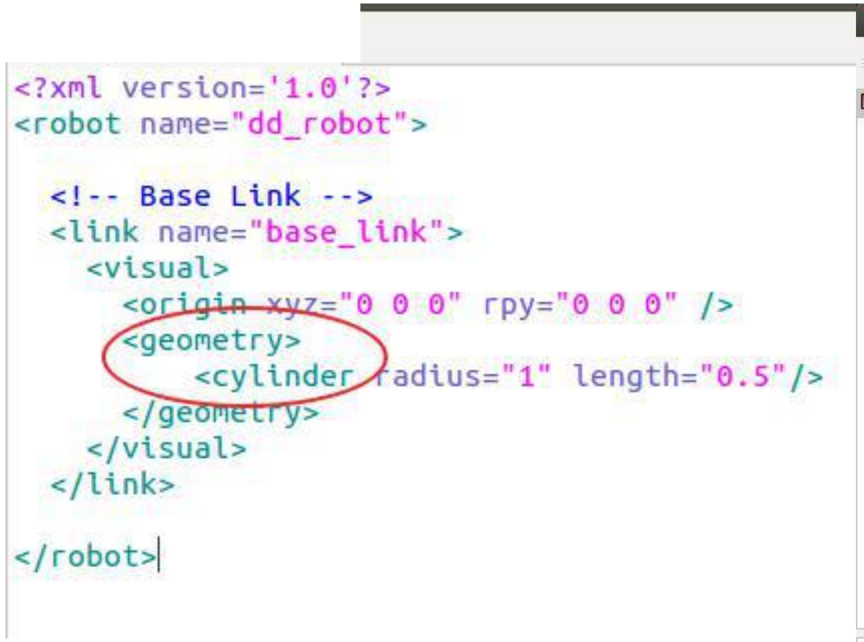


What is URDF?

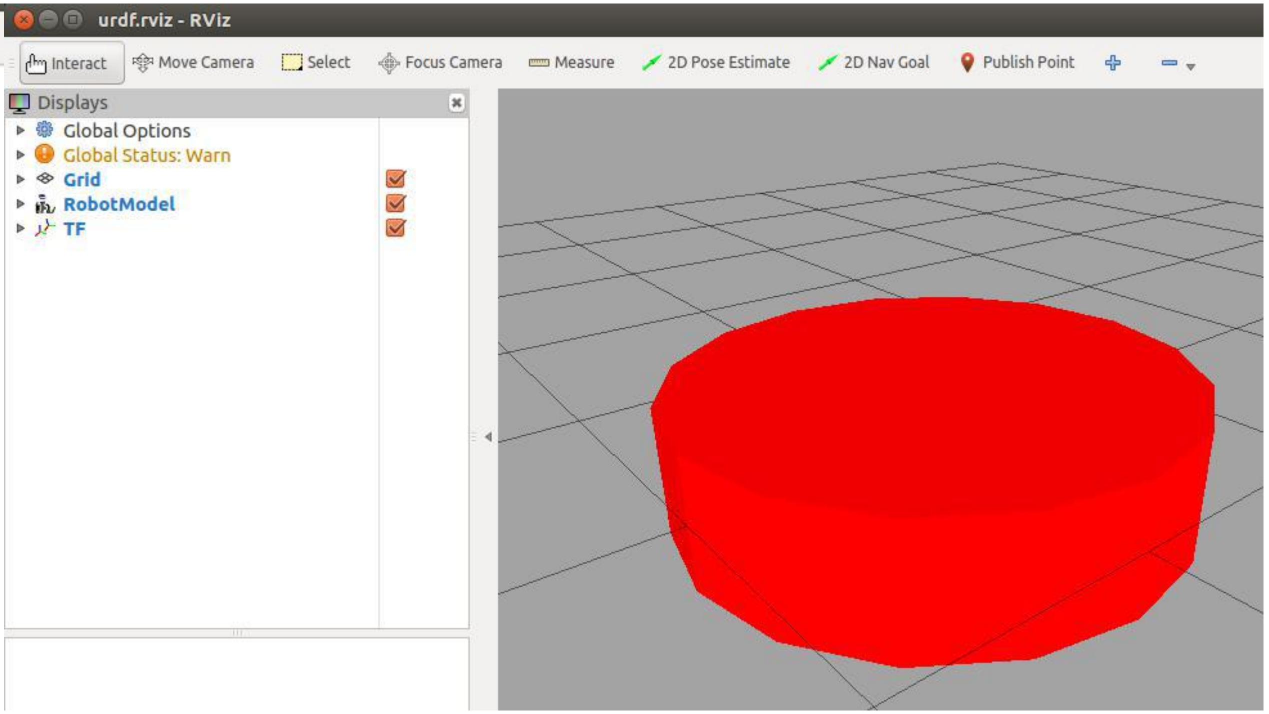
http://gazebosim.org/tutorials/?tut=ros_urdf

✓ The **Universal Robotic Description Format** (URDF) is an XML file format used in ROS to describe all elements of a robot

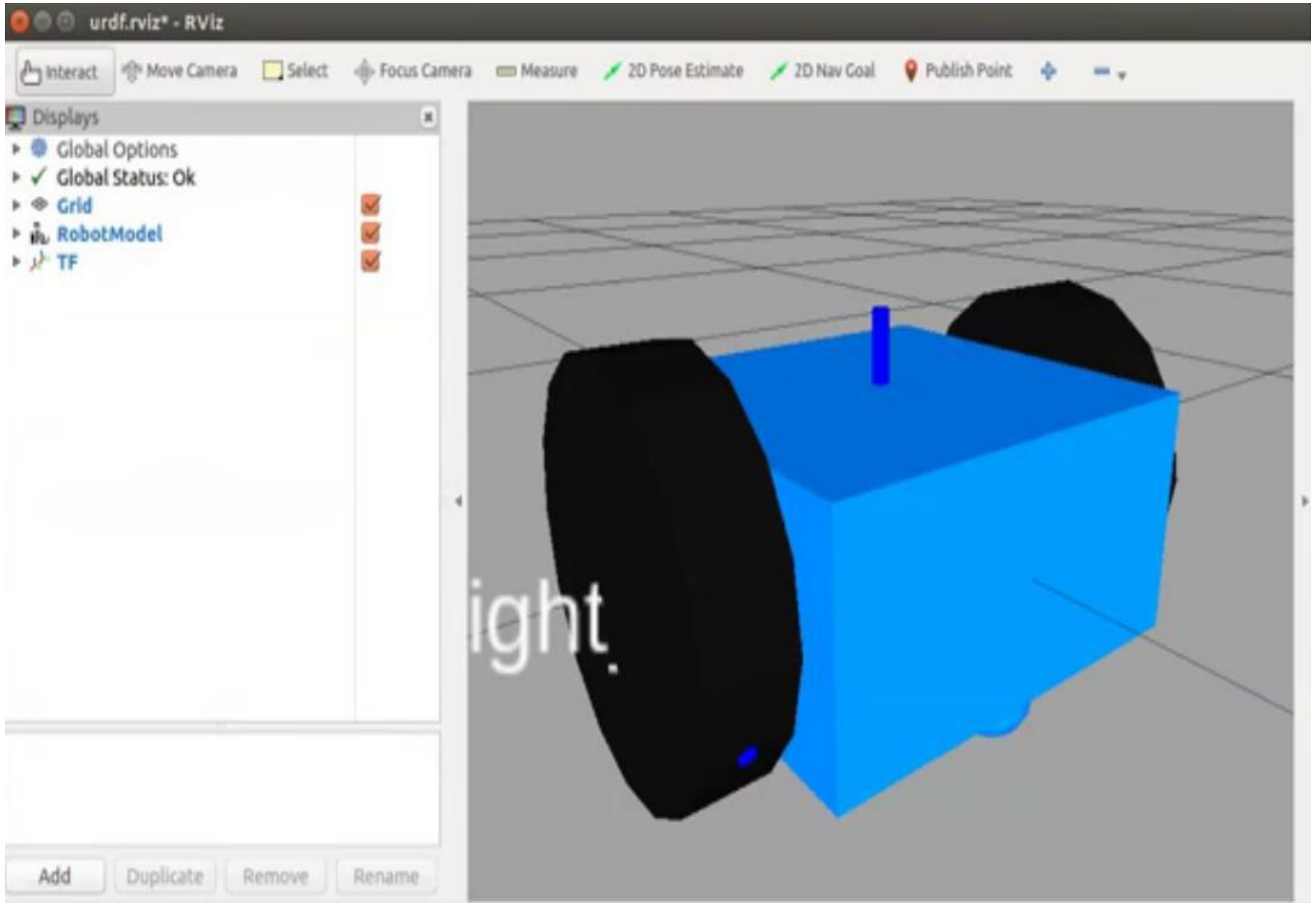
✓ URDF are xml files



```xml
<?xml version='1.0'?>
<robot name="dd_robot">

  <!-- Base Link -->
  <link name="base_link">
    <visual>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
          <cylinder radius="1" length="0.5"/>
      </geometry>
    </visual>
  </link>

</robot>
```



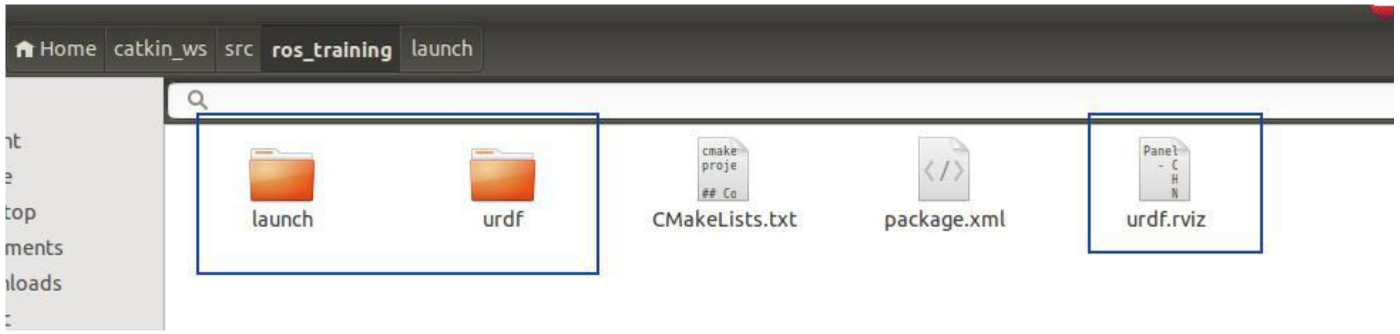
Lab# Making our own 2 wheel robot



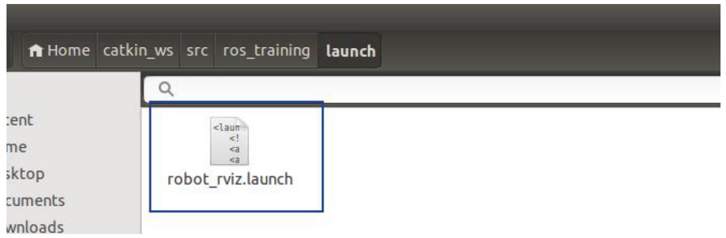
- ✓ **Create body**
- ✓ **Create wheels**
- ✓ **Create castor**
- ✓ **Run wheels using GUI = True**

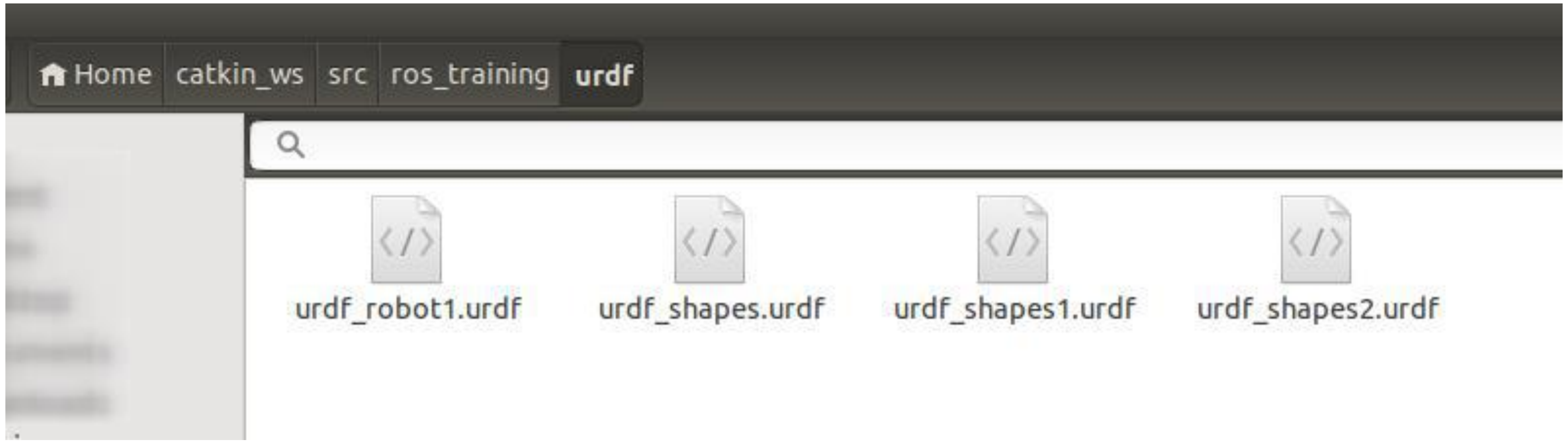
Steps to check before you launch URDF

1# Make sure you have the Launch and URDF folder under ros_training/src
2# Copy/Replace the urdf.rviz from GitHub folder



3# Copy the robot_rviz.launch file under launch folder
4# Copy these four urdf files under folder URDF



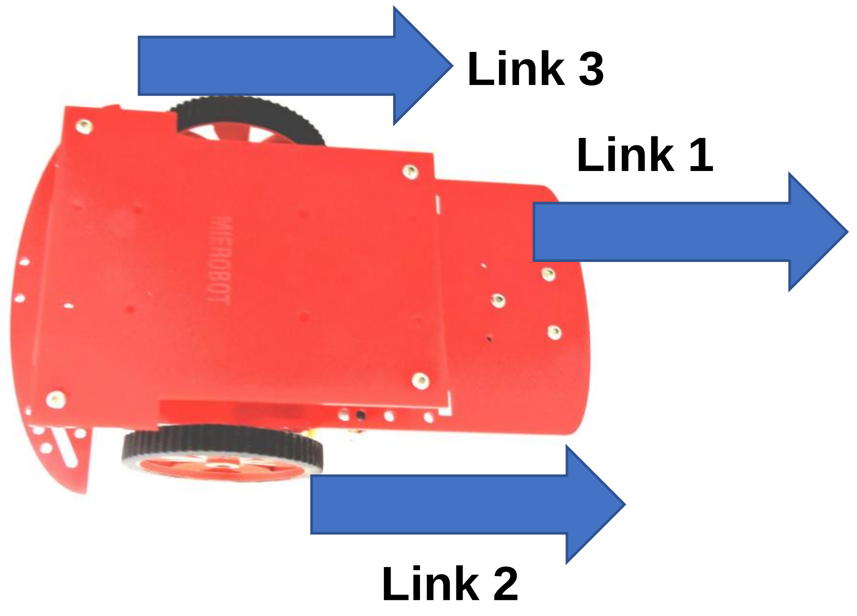


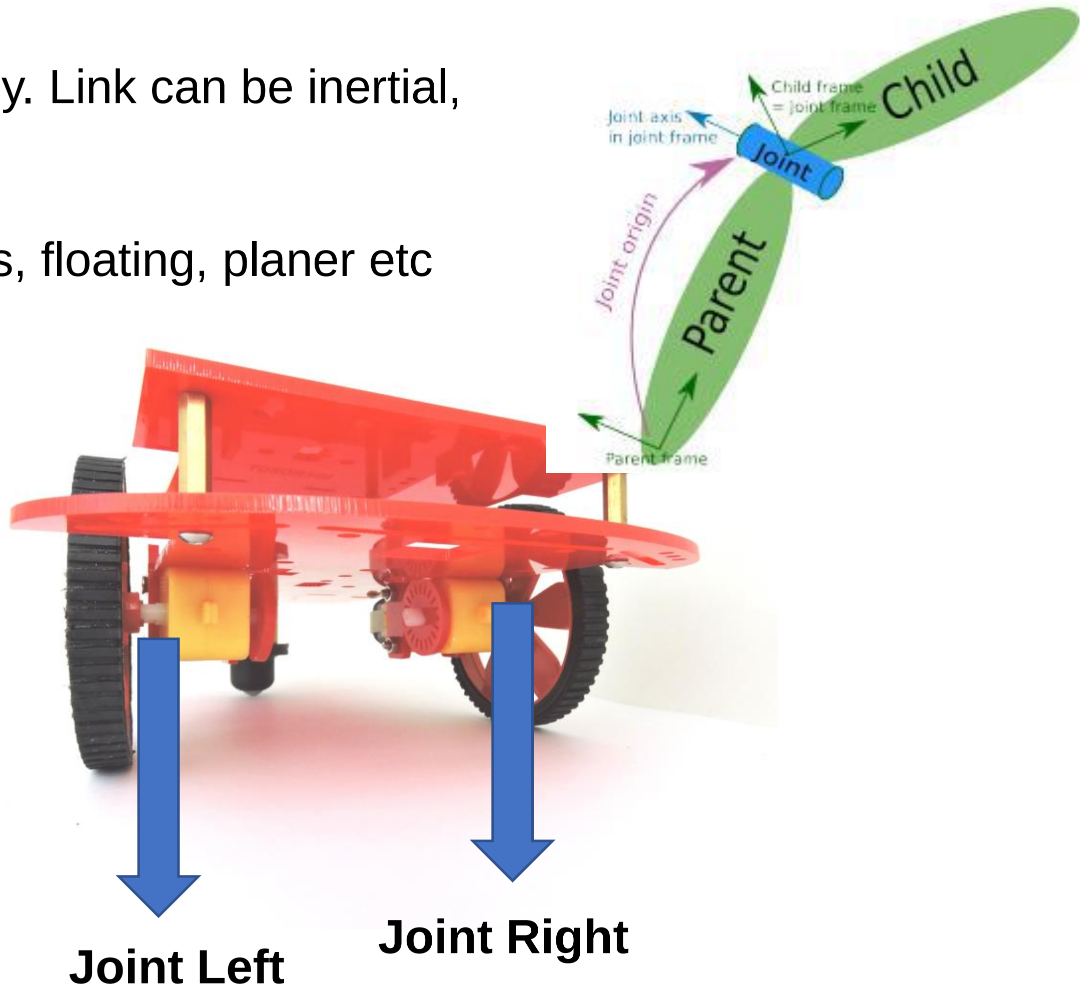
Understanding links, joints & collision

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

Link in the URDF XML file describes the rigid body. Link can be inertial, visual features or collision features

Joint connect two links & can be fixed, continuous, floating, planer etc

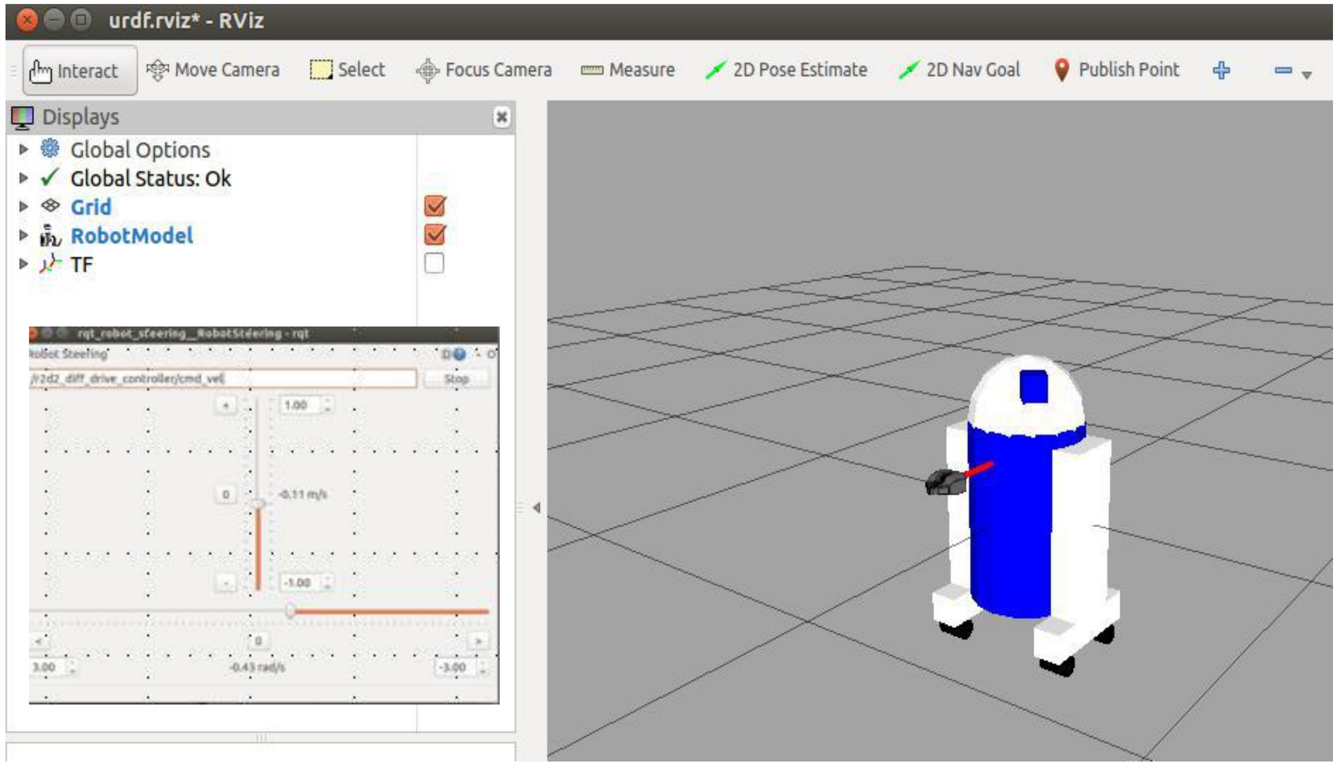




Collision tag is for boundary identification in udrf files

Section: Running a Star war R2D2 robot with head & gripper in ROS Simulation

Lab# Running a R2D2 Robot

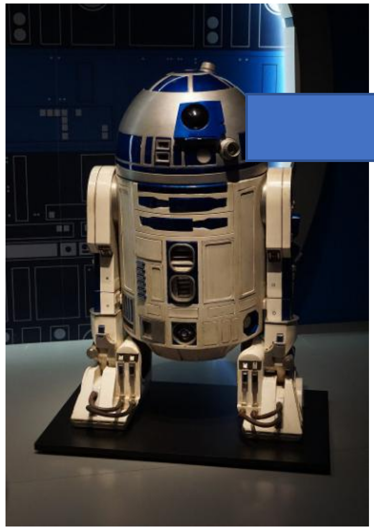


Source: http://wiki.ros.org/urdf/Tutorials/Building%20a%20Movable%20Robot%20Model%20with%20URDF

https://github.com/ros/urdf_tutorial/blob/master/urdf/06-flexible.urdf

Design consideration for R2D2 robot

http://wiki.ros.org/urdf/Tutorials/Building%20a%20Movable%20Robot%20Model%20with%20URDF

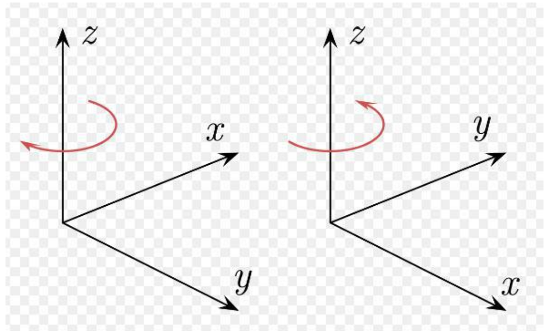


Head:

✓ Head and Body joint is a continuous joint Ranges from +infinity to – infinity

✓ Axis of rotation in xyz triplet is z axis

```
<joint name="head_swivel" type="continuous">
  <parent link="base_link"/>
  <child link="head"/>
  <axis xyz="0 0 1"/>
  <origin xyz="0 0 0.3"/>
</joint>
```

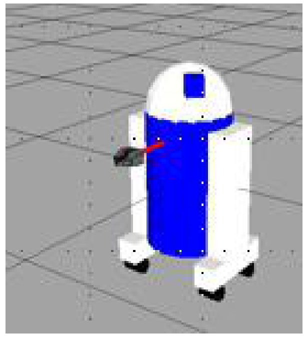


Gripper:

✓ Both the right and the left gripper joints are modelled as revolute joints.

✓ Revolute joints have limits that needs to be defined.

```
<joint name="left_gripper_joint" type="revolute">
  <axis xyz="0 0 1"/>
  <limit effort="1000.0" lower="0.0" upper="0.548" velocity="0.5"/>
  <origin rpy="0 0 0" xyz="0.2 0.01 0"/>
  <parent link="gripper_pole"/>
  <child link="left_gripper"/>
</joint>
```

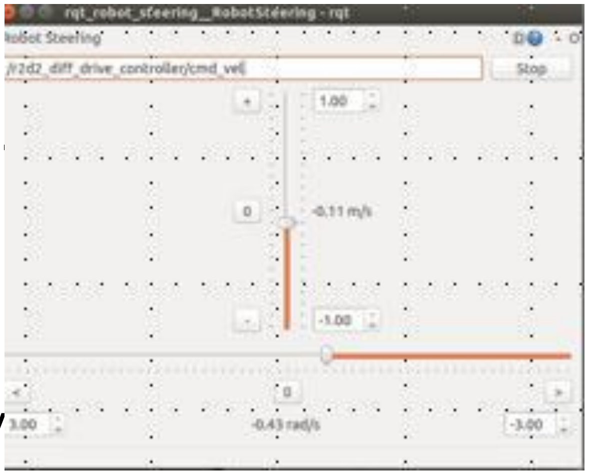


Design consideration for R2D2 robot

http://wiki.ros.org/urdf/Tutorials/Building%20a%20Movable%20Robot%20Model%20with%20URDF

Pose estimation: (provide information about the absolute or relative position and orientation of a robot)

First the GUI parses the URDF and finds all the non-fixed joints and their limits. Then, it uses the values of the sliders to publish **sensor_msgs/JointState messages.**



rqt_robot_steering__RobotSteering - rqt
Robot Steering
/r2d2_diff_drive_controller/cmd_vel
Stop
1.00
0
-0.11 m/s
-1.00
3.00
0
-0.43 rad/s
-3.00

Those are then used by **robot_state_publisher** to calculate all of transforms between the different parts. The resulting transform tree is then used to display all of the shapes in Rviz.

JointState messages: (This is a message that holds data to describe the state of a set of torque controlled joints)

# Message Type within sensor_msg ROS package (standard - http://wiki.ros.org/sensor_msgs)

The robot_state_publisher node subscribes to the JointState message and publishes the state of the robot to the tf transform library.

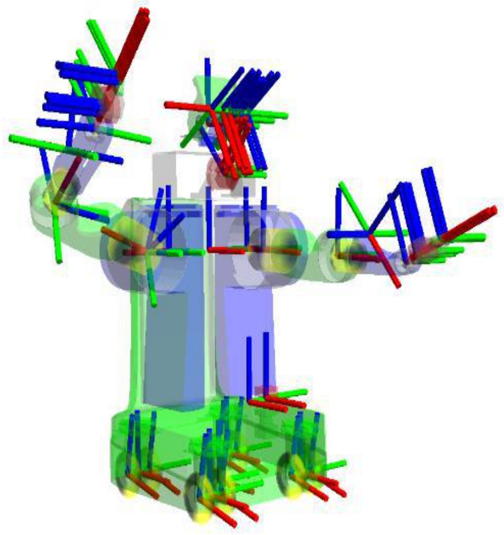
The state of each joint (revolute or prismatic) is defined by:
# * the position of the joint (rad or m),
# * the velocity of the joint (rad/s or m/s) and
# * the effort that is applied in the joint (Nm or N).

Design consideration for R2D2 robot

http://wiki.ros.org/urdf/Tutorials/Building%20a%20Movable%20Robot%20Model%20with%20URDF

TF:
tf is a package that lets the user keep track of **multiple coordinate frames** over time. tf maintains the relationship between coordinate frames in a tree structure buffered in time, and lets the user transform points, vectors, etc between any two coordinate frames at any desired point in time.

More at http://wiki.ros.org/tf



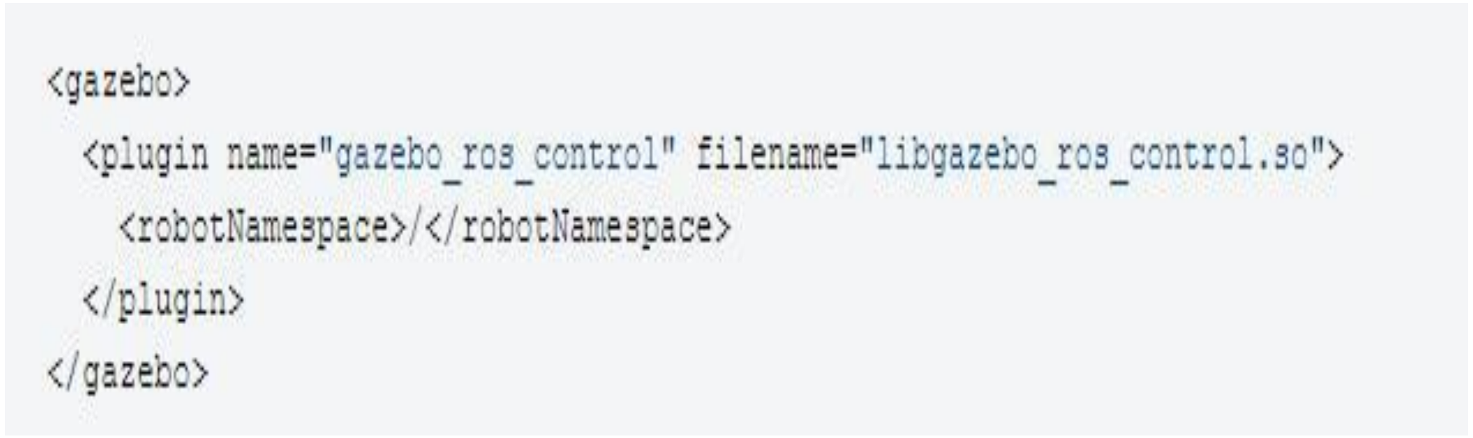
robot_state_publisher : This package allows you to **publish the state** of a robot to tf. Once the state gets published, it is available to all components in the system that also use tf. The package takes the joint angles of the robot as input and publishes the 3D poses of the robot links, using a kinematic tree model of the robot.

More at http://wiki.ros.org/robot_state_publisher

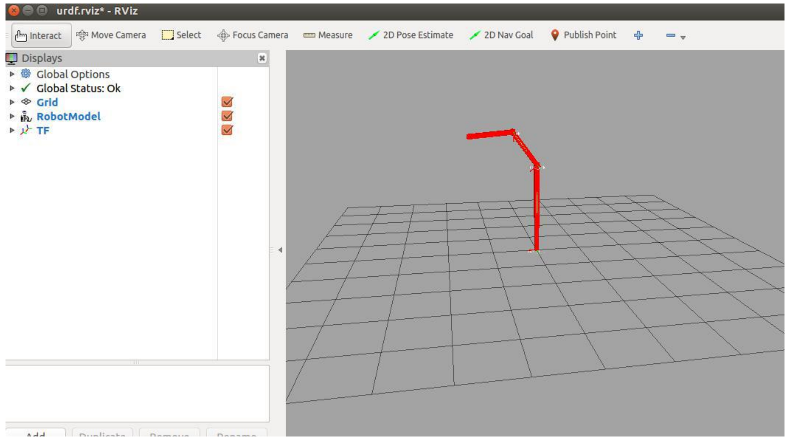
What is Gazebo? Xacro?

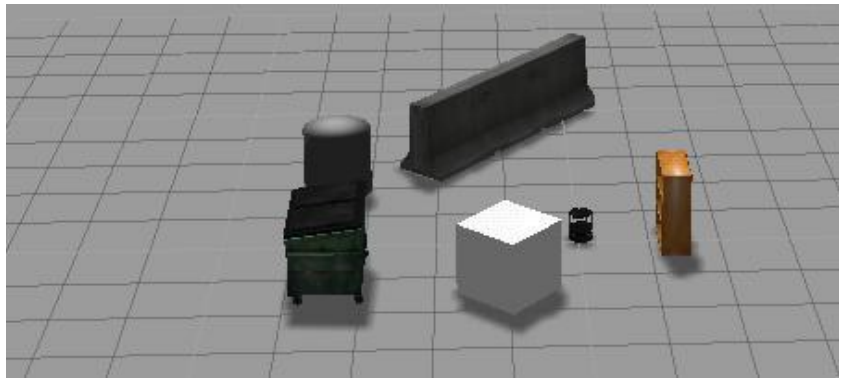
http://gazebosim.org/

- ✓ Gazebo is a robust physics engine, high-quality graphics, and convenient programmatic and graphical interfaces. We can link Gazebo to ROS using plugin.
- ✓ **roslaunch gazebo_ros empty_world.launch**



```
<gazebo>
  <plugin name="gazebo_ros_control" filename="libgazebo_ros_control.so">
    <robotNamespace>/</robotNamespace>
  </plugin>
</gazebo>
```





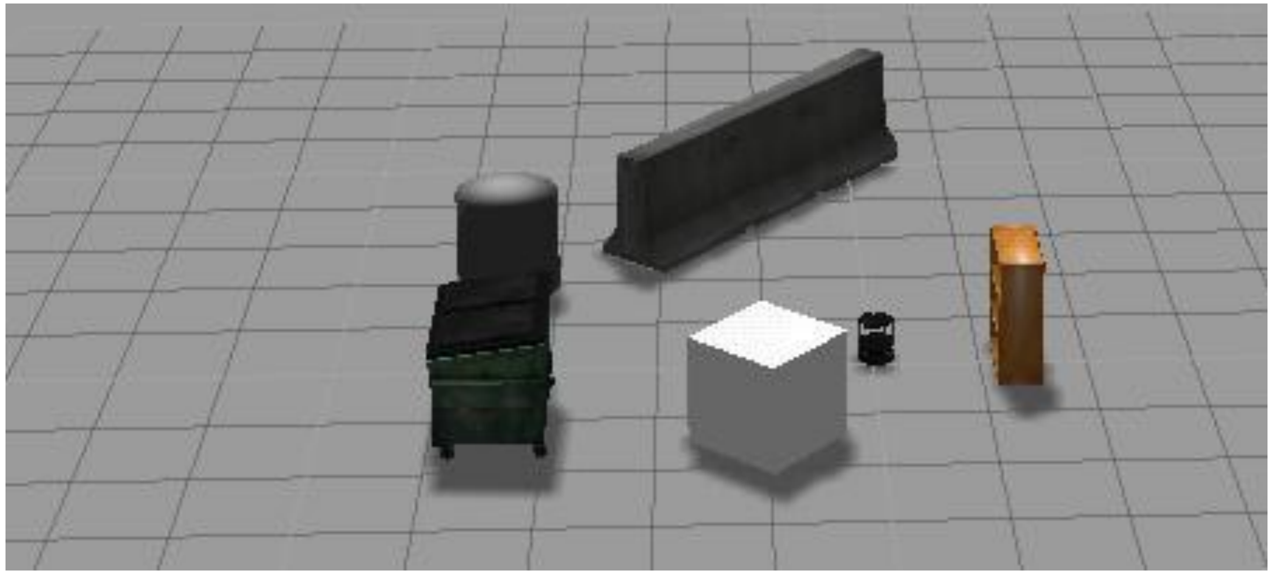
Gazebo Basic commands

- ✓ **sudo apt-get install ros-indigo-gazebo-ros-pkgs ros-indigo-gazebo-ros-control**

- ✓ **gazebo**

- ✓ **which gzserver**
- ✓ **which gzclient**

- ✓ **roslaunch gazebo_ros empty_world.launch**



What is Xacro

- ✓ **Xacro (XML Macros) Xacro is an XML macro language**

- ✓ With xacro, you can construct shorter and more readable XML files by using macros that expand to larger XML expressions

- ✓ Why Xacro?
  - ✓ <xacro:include filename="path to filename/filename" />
  - ✓ Use of property blocks
  - ✓ Use of macro for sequence of statements

Xacro - Example

XML without Xacro

```
<link name="base_link">
  <visual>
    <geometry>
      <cylinder length="0.6" radius="0.2"/>
    </geometry>
    <material name="blue"/>
  </visual>
  <collision>
    <geometry>
      <cylinder length="0.6" radius="0.2"/>
    </geometry>
  </collision>
</link>
```

XML with Xacro

```
<xacro:property name="width" value="0.2" />
<xacro:property name="bodylen" value="0.6" />
<link name="base_link">
    <visual>
        <geometry>
            <cylinder radius="${width}" length="${bodylen}"/>
        </geometry>
        <material name="blue"/>
    </visual>
    <collision>
        <geometry>
            <cylinder radius="${width}" length="${bodylen}"/>
        </geometry>
    </collision>
</link>
```

R2D2 execution order

Package: **URDF_Tutorials (make sure you are at ~/catkin_ws/src/urdf_tutorial)**

1# roslaunch urdf_tutorial display.launch model:=urdf/05-visual.urdf (R2-D2 structure)

2# roslaunch urdf_tutorial display.launch model:=urdf/06-flexible.urdf (R2-D2 with movable structure & controller)

3# roslaunch urdf_tutorial display.launch model:=urdf/08-macroed.urdf.xacro (R2-D2 with macro)

Package: **urdf_sim_tutorial   (make sure you are at ~/catkin_ws/src/urdf_sim_tutorial)**

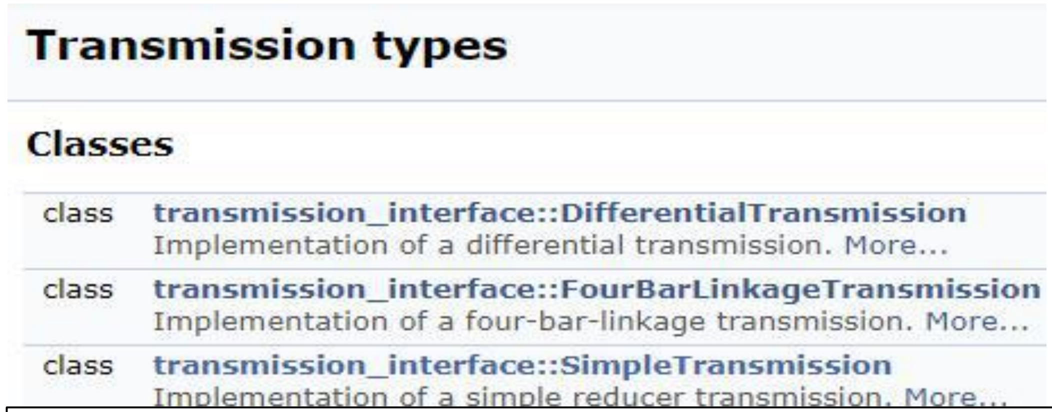
4# roslaunch urdf_sim_tutorial gazebo.launch (R2-D2 in Gazebo)

5# roslaunch urdf_sim_tutorial 13-diffdrive.launch (R2-D2 in Gazebo with controller for wheels)

Transmission types

http://docs.ros.org/jade/api/transmission_interface/html/c++/classtransmission__interface_1_1SimpleTransmission.html#details

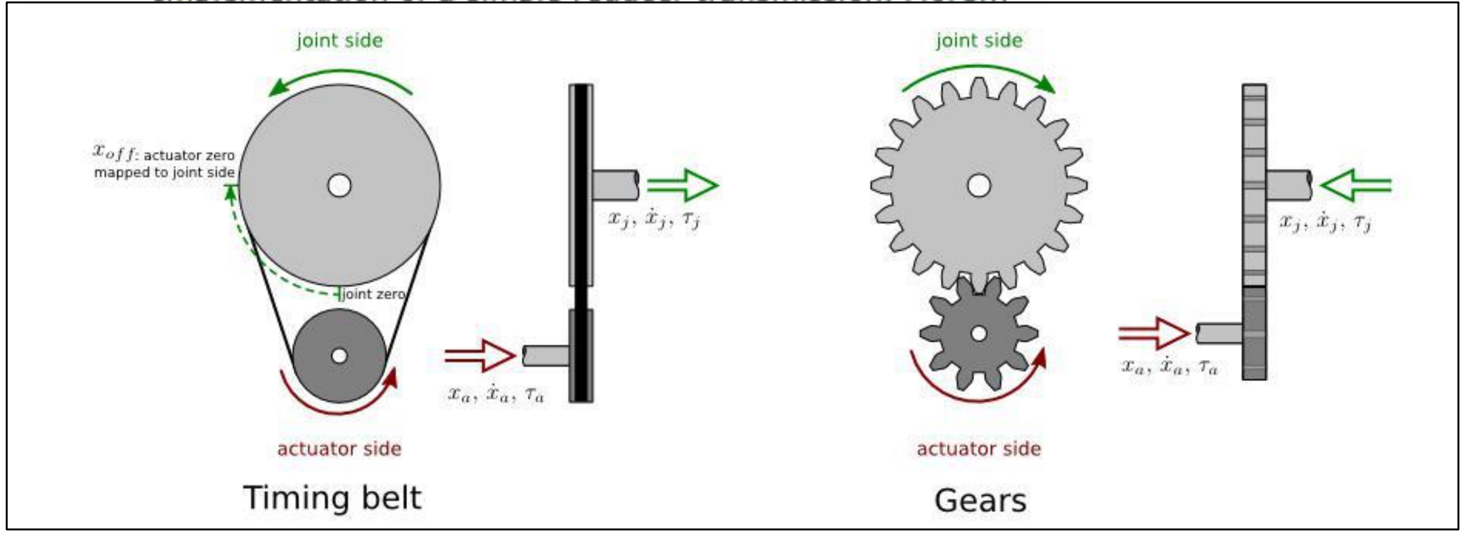
- ✓ For every non-fixed joint, we need to specify a transmission, which tells Gazebo what to do with the joint (**transmission_interface)**
- ✓



Transmission types

Classes

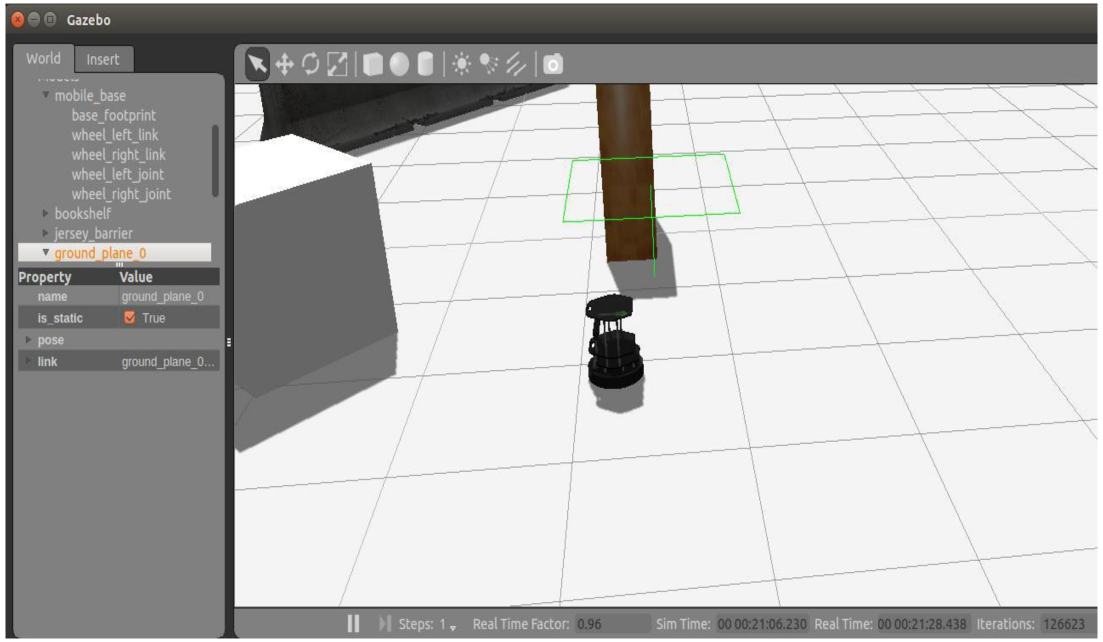
class	**transmission_interface::DifferentialTransmission** Implementation of a differential transmission. More...
class	**transmission_interface::FourBarLinkageTransmission** Implementation of a four-bar-linkage transmission. More...
class	**transmission_interface::SimpleTransmission** Implementation of a simple reducer transmission. More...



Section: Running the Turtle Bot2 with keyboard telemetry

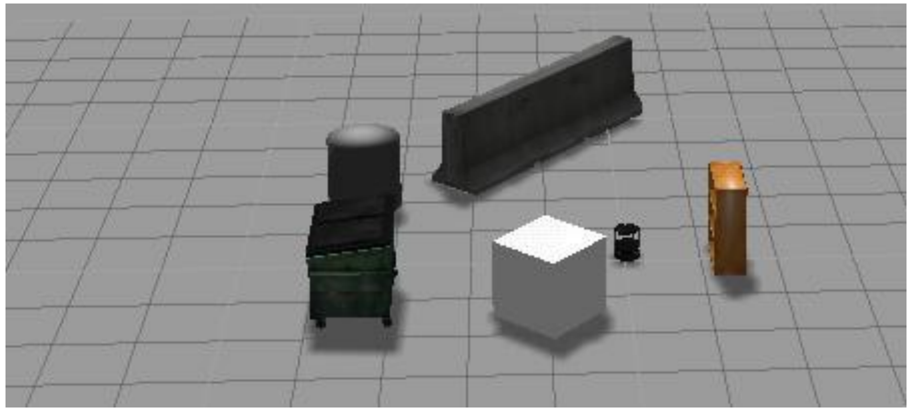
Anirban Ghatak, MieRobot

Lab # Running the Turtle Bot



Property	Value
name	ground_plane_0
is_static	✓ True
pose	
link	ground_plane_0...

```
Steps: 1   Real Time Factor: 0.96   Sim Time: 00 00:21:06.230   Real Time: 00 00:21:28.438   Iterations: 126623
```



ROS service

Topics should be used for continuous data streams (sensor data, robot state).

Services should be used for remote procedure calls that terminate quickly, e.g. for querying the state of a node or doing a quick calculation such as Inverse Kinematics.

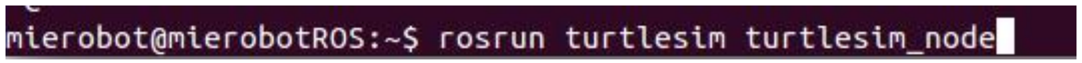
They should never be used for longer running processes, in particular processes that might be required to pre-empt if exceptional situations occur and they should never change or depend on state to avoid unwanted side effects for other nodes.

Commands:

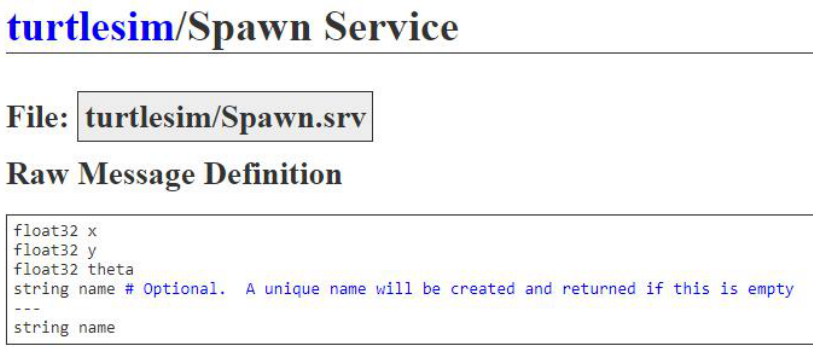
- ✓ **rosservice list Print information about active services.**
- ✓ **rosservice node Print the name of the node providing a service.**
- ✓ **rosservice call Call the service with the given args.**
- ✓ **rosservice args List the arguments of a service.**
- ✓ **rosservice type Print the service type.**
- ✓ **rosservice uri Print the service ROSRPC uri.**
- ✓ **rosservice find Find services by service type.**

A worked out example

Step 1#



```
mierobot@mierobotROS:~$ rosrun turtlesim turtlesim_node
```



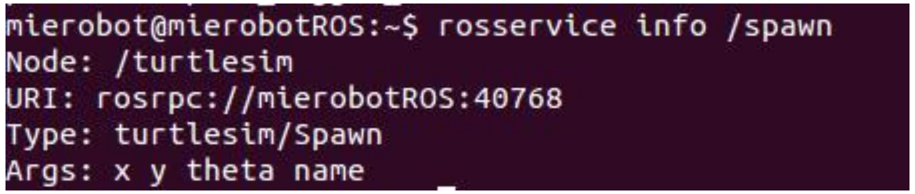
turtlesim/Spawn Service

File: turtlesim/Spawn.srv

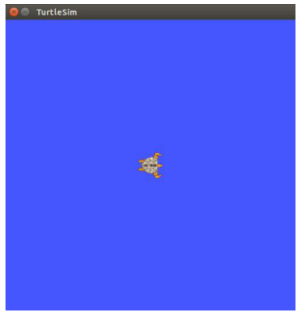
Raw Message Definition

```
float32 x
float32 y
float32 theta
string name # Optional.  A unique name will be created and returned if this is empty
---
string name
```

Step 2#

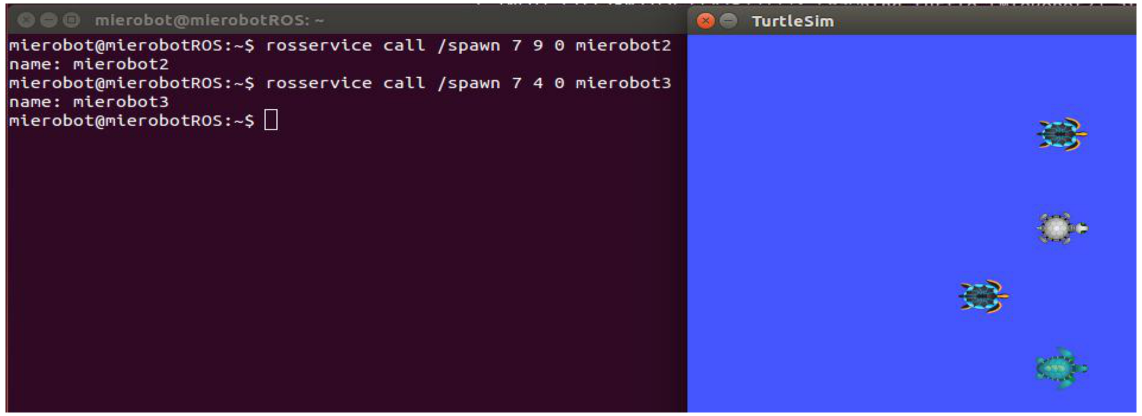


```
mierobot@mierobotROS:~$ rosservice info /spawn
Node: /turtlesim
URI: rosrpc://mierobotROS:40768
Type: turtlesim/Spawn
Args: x y theta name
```



(0,0)

Step 3#



```
mierobot@mierobotROS:~$ rosservice call /spawn 7 9 0 mierobot2
name: mierobot2
mierobot@mierobotROS:~$ rosservice call /spawn 7 4 0 mierobot3
name: mierobot3
mierobot@mierobotROS:~$
```

Commands for Turtle Bot2

Launch command:

- ✓ **roslaunch turtlebot_gazebo turtlebot_world.launch**
- ✓ **rosservice call gazebo/get_model_state '{model_name: mobile_base}’**
- ✓ **rosservice list**
- ✓ **rostopic list | grep mobile_base**
- ✓ **rostopic type /mobile_base/commands/velocity**
- ✓ **rostopic pub -r 10 mobile_base/commands/velocity \geometry_msgs/Twist '{linear: {x: 0.3}}’**

Teleop with Keyboard:

- ✓ **roslaunch turtlebot_teleop keyboard_teleop.launch**

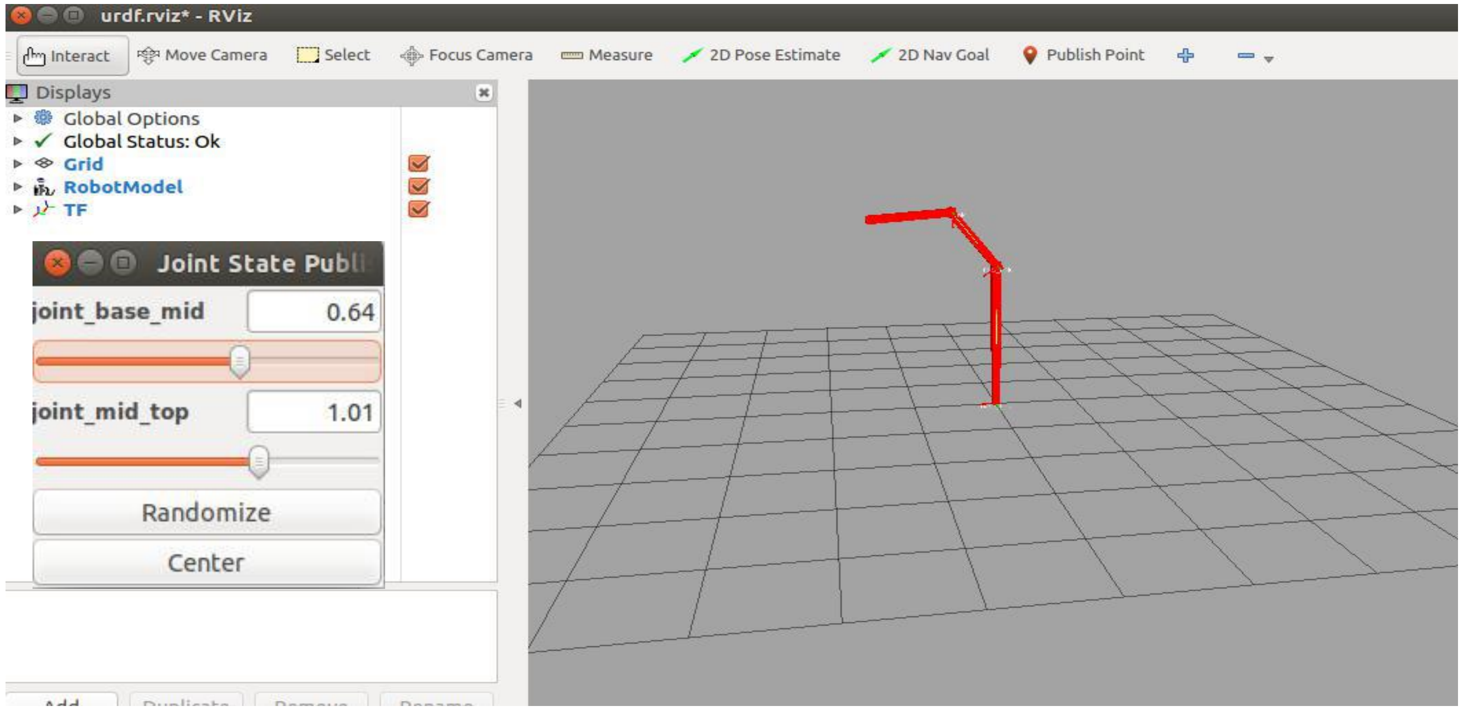
Hint

Shift + Left click changes camera angle

Section:3

Gazebo simulation for a Robotic Arm using Xacro

Our section goal



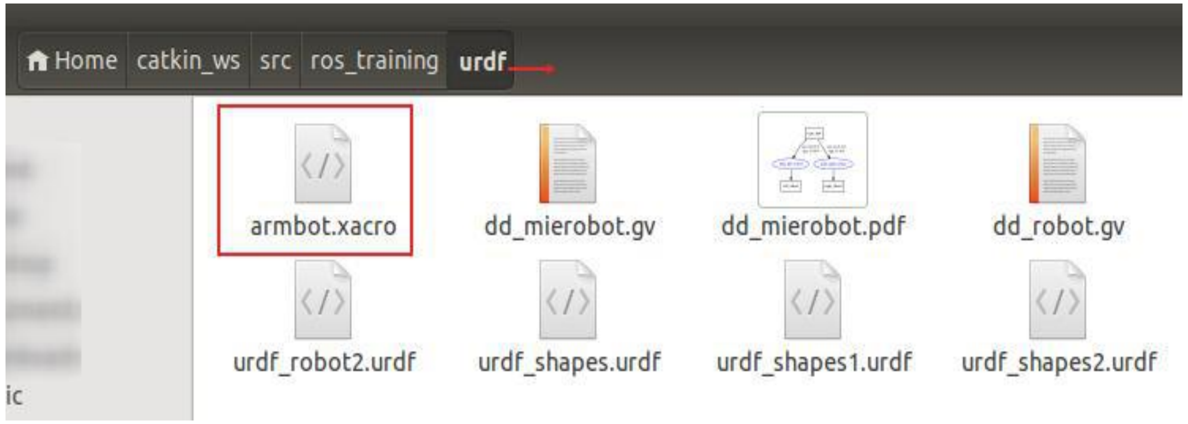
joint_base_mid 0.64

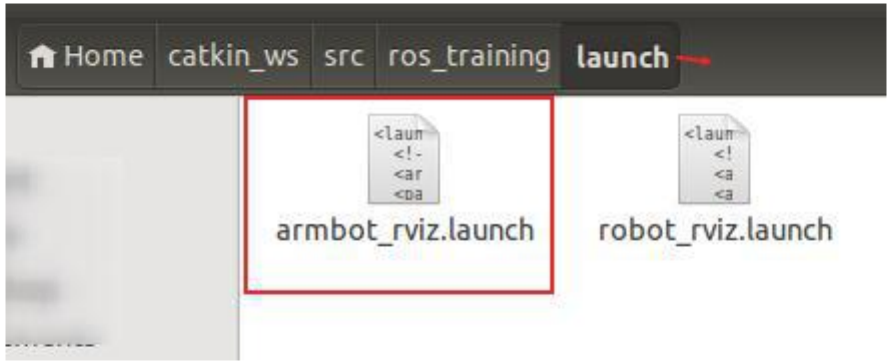
joint_mid_top 1.01

We will re-use concepts learnt in previous section on Gazebo & Xacro so please make sure you have watched the previous section

Steps to run the arm robot

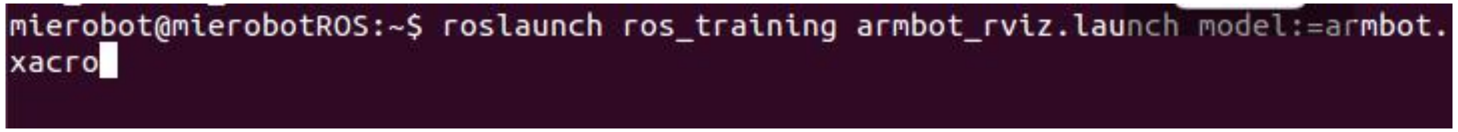
Copy the right launch file & Xacro in right folder path ~/catkin_ws/src/ros_training





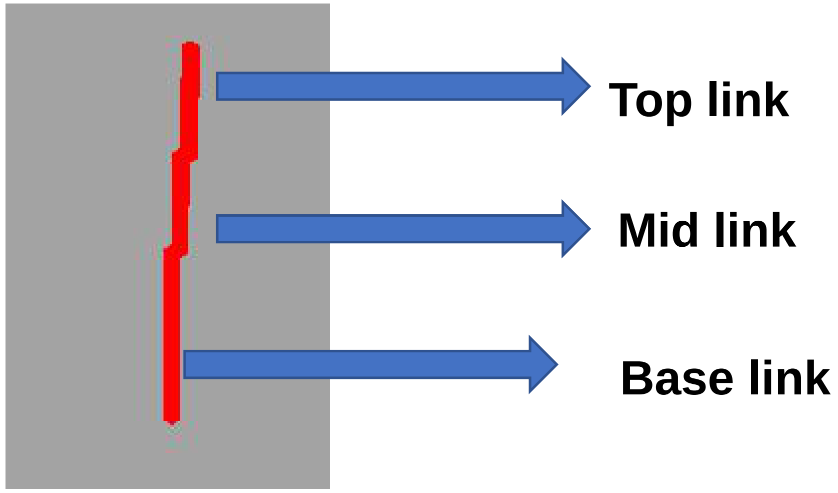
Command:
roslaunch ros_training armbot_rviz.launch model:-armbot.Xacro

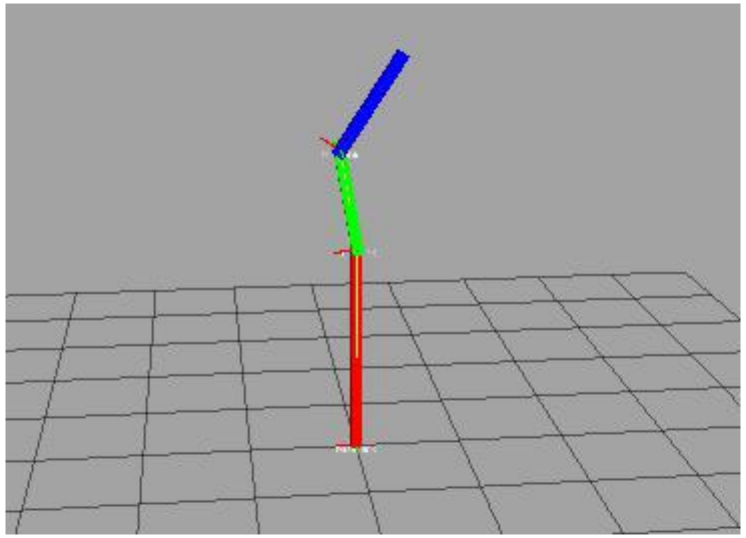
roslaunch ros_training armbot_rviz.launch model:-armbot2.xacro



```
mierobot@mierobotROS:~$ roslaunch ros_training armbot_rviz.launch model:=armbot.
xacro
```

Understanding the approach

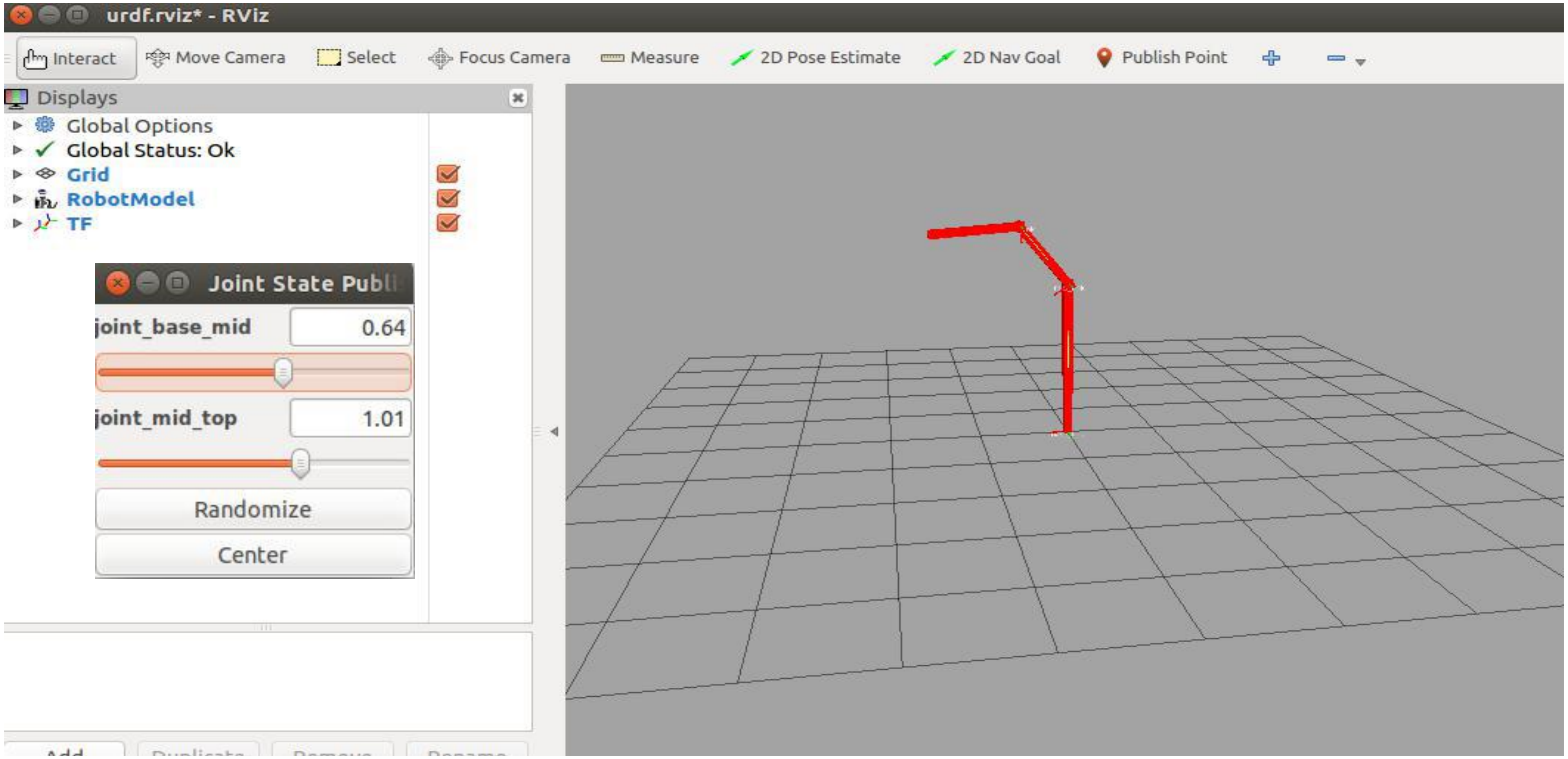




Axle offset: an offset for axis of rotation

A damping coefficient is a material property that indicates whether a material will bounce back or return energy to a system. A ball will have low damping coefficient.

Lab# Gazebo simulation for a Robotic Arm



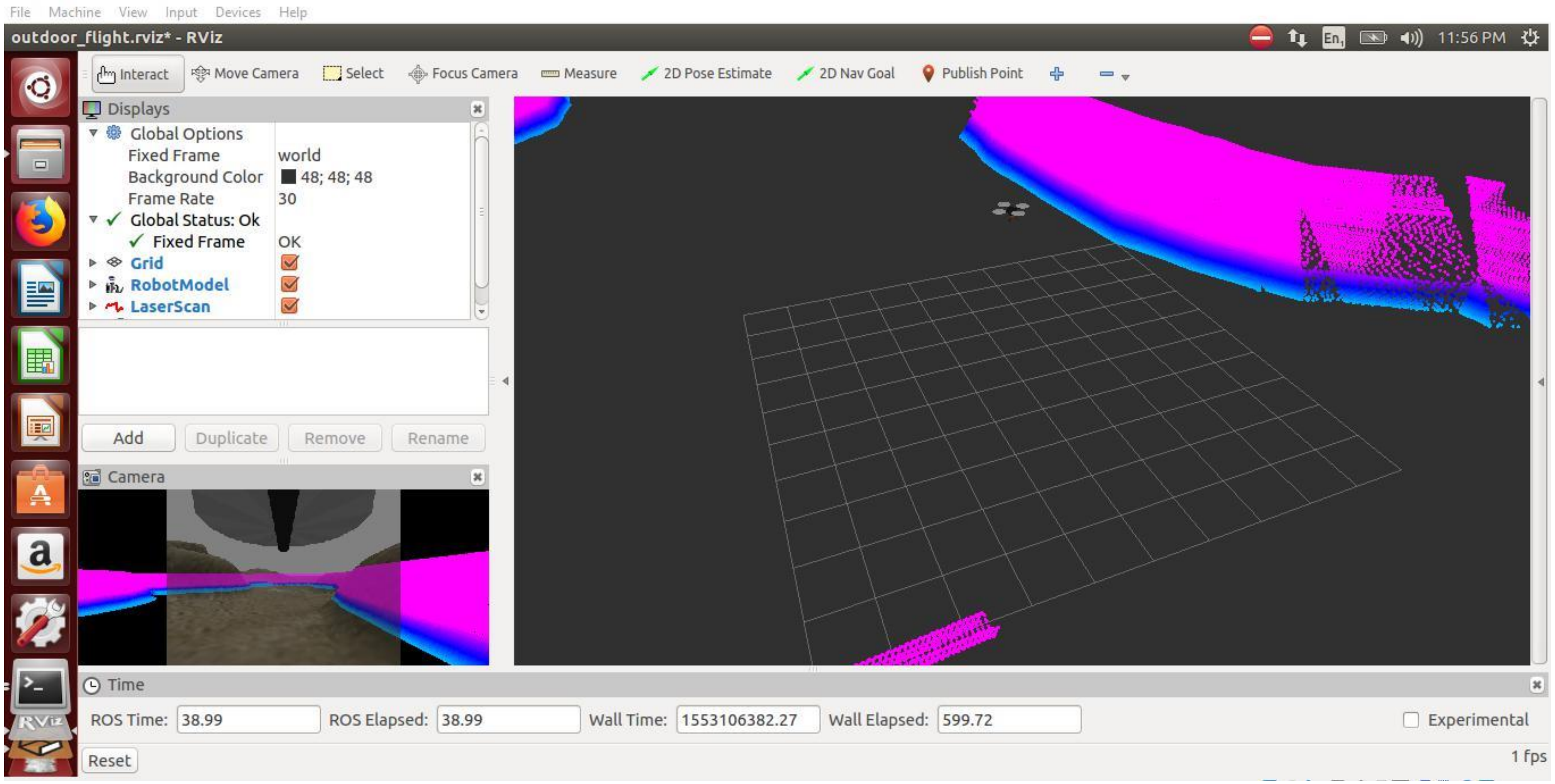
joint_base_mid 0.64

joint_mid_top 1.01

Section:4

Use a XBOX controller to control a drone in ROS

Our section goal



Global Options
Fixed Frame world
Background Color ■ 48; 48; 48
Frame Rate 30
✓ Global Status: Ok
✓ Fixed Frame OK
Grid ✓
RobotModel ✓
LaserScan ✓

ROS Time: 38.99 ROS Elapsed: 38.99 Wall Time: 1553106382.27 Wall Elapsed: 599.72 ☐ Experimental

1 fps

Section: Use the XBOX controller to control Turtle Sim

Anirban Ghatak, MieRobot

Knowing and installing jstest-gtk

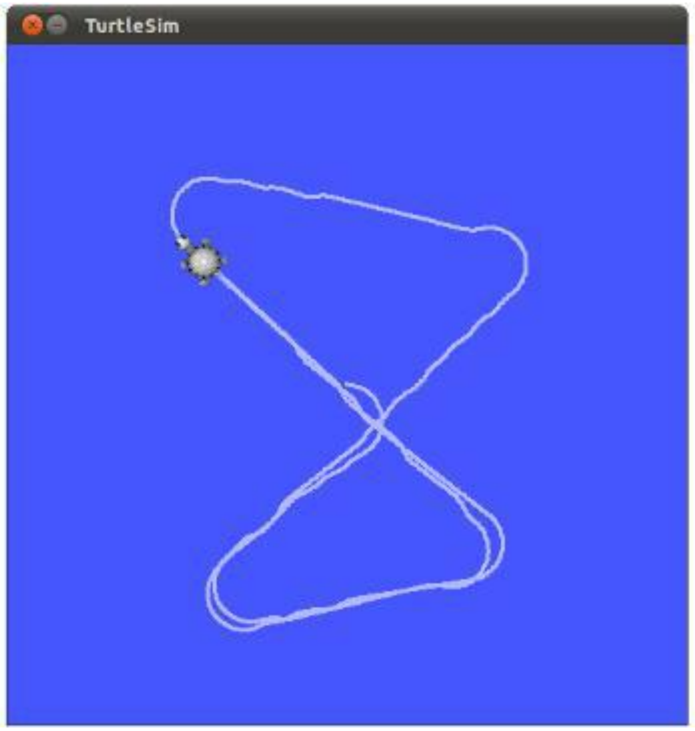
https://launchpad.net/ubuntu/xenial/+package/jstest-gtk

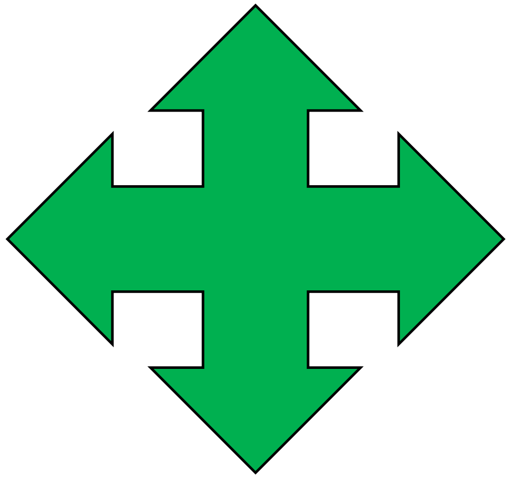
- ✓ **jstest-gtk** is a simple graphical joystick tester
- ✓ It provides a list of attached joysticks
- ✓ It can display which buttons & axes are pressed
- ✓ Package can remap axes and buttons, and calibrate the device
- ✓ I used a low cost Joy stick for this course ($4)

Package install: sudo apt-get install jstest-gtk

Test: jstest-gtk

Lab# Run Turtle Sim with XBOX JS



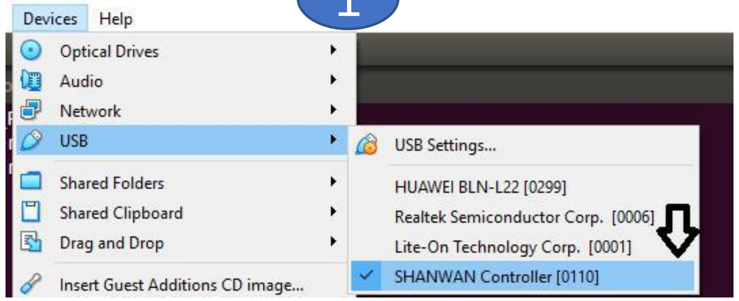


Commands for the JS Turtle Sim (1/2)

http://wiki.ros.org/joy/Tutorials/ConfiguringALinuxJoystick

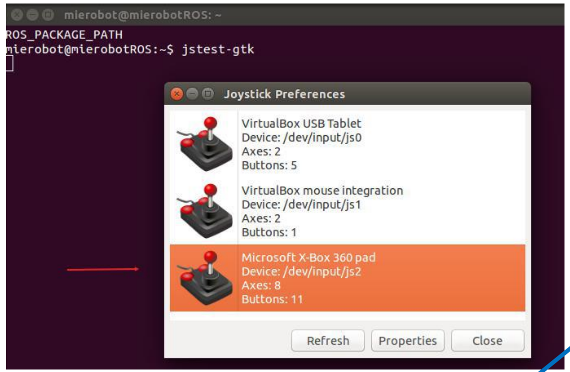
Make sure USB is detected

1


Devices
Help
Optical Drives
Audio
Network
USB
Shared Folders
Shared Clipboard
Drag and Drop
Insert Guest Additions CD image...
USB Settings...
HUAWEI BLN-L22 [0299]
Realtek Semiconductor Corp. [0006]
Lite-On Technology Corp. [0001]
SHANWAN Controller [0110]

Test: jstest-gtk

2



```
mierobot@mierobotROS: ~
ROS_PACKAGE_PATH
mierobot@mierobotROS:~$ jstest-gtk
```

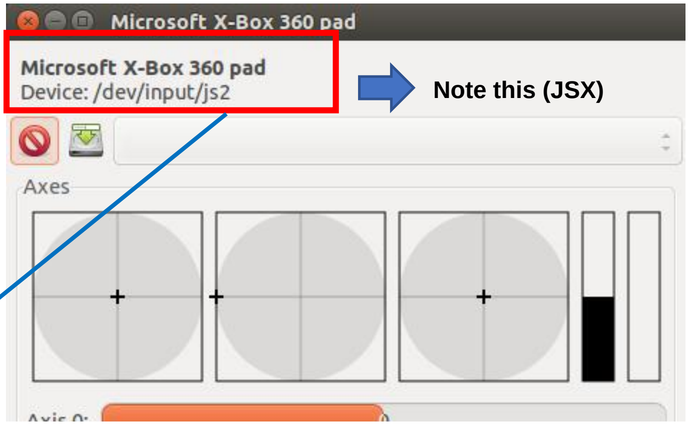
Joystick Preferences

VirtualBox USB Tablet
Device: /dev/input/js0
Axes: 2
Buttons: 5

VirtualBox mouse integration
Device: /dev/input/js1
Axes: 2
Buttons: 1

Microsoft X-Box 360 pad
Device: /dev/input/js2
Axes: 8
Buttons: 11

3

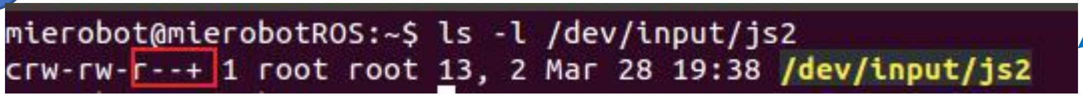


Microsoft X-Box 360 pad

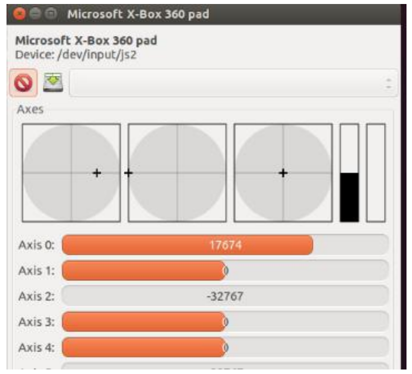
Microsoft X-Box 360 pad
Device: /dev/input/js2

Note this (JSX)

4



```
mierobot@mierobotROS:~$ ls -l /dev/input/js2
crw-rw-r--+ 1 root root 13, 2 Mar 28 19:38 /dev/input/js2
```



Microsoft X-Box 360 pad

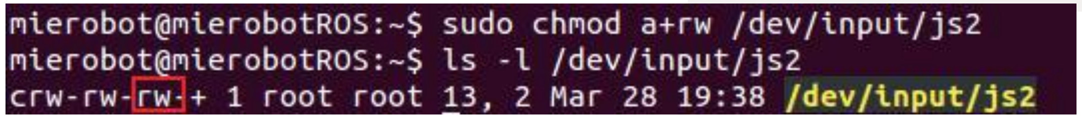
Microsoft X-Box 360 pad
Device: /dev/input/js2

Axis 0: 17674
Axis 1: 0
Axis 2: -32767
Axis 3: 0
Axis 4: 0

The Red box above should be rw (read write). Above it has read permission only & we need to update the rights.

5

sudo chmod a+rw /dev/input/js2



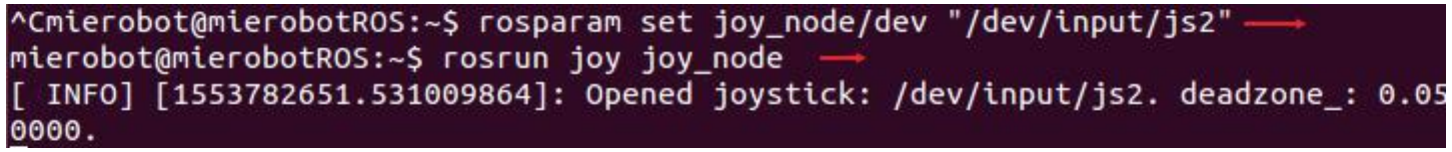
```
mierobot@mierobotROS:~$ sudo chmod a+rw /dev/input/js2
mierobot@mierobotROS:~$ ls -l /dev/input/js2
crw-rw-rw-+ 1 root root 13, 2 Mar 28 19:38 /dev/input/js2
```

6

Test: jstest-gtk

Commands for the JS Turtle Sim (2/2)

Run roscore and type below 1



```
^Cmierobot@mierobotROS:~$ rosparam set joy_node/dev "/dev/input/js2"
mierobot@mierobotROS:~$ rosrun joy joy_node
[ INFO] [1553782651.531009864]: Opened joystick: /dev/input/js2. deadzone_: 0.05
0000.
```

Install via **sudo apt-get install ros-indigo-joy**

2

rosrun joy joy_node

Control Turtle Sim:

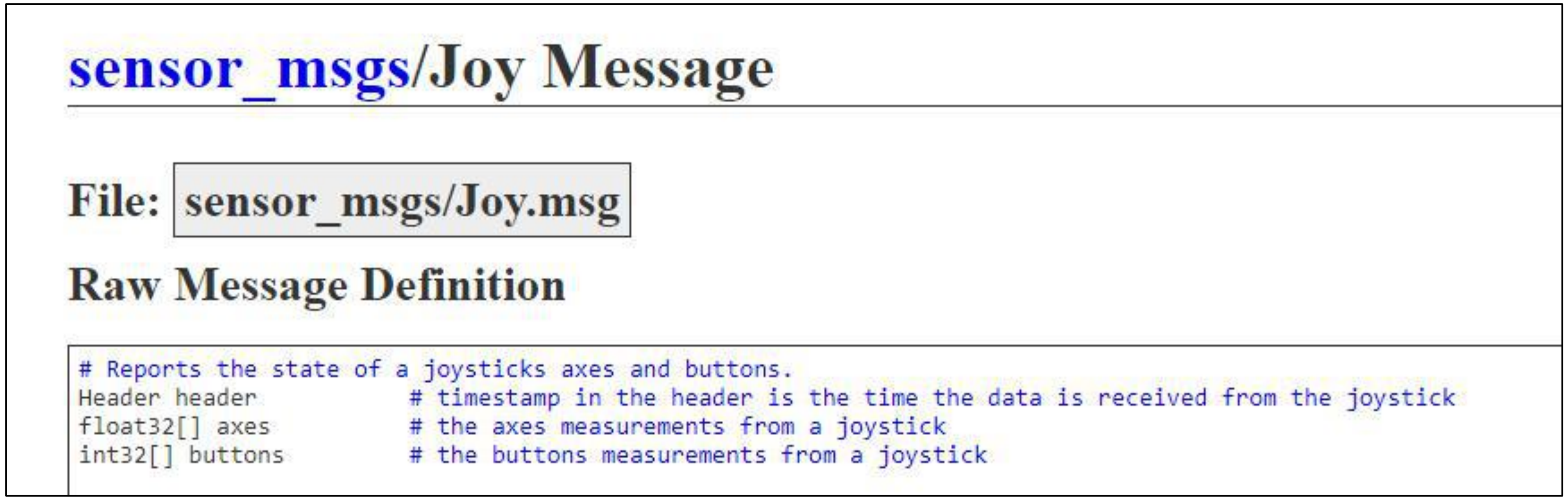
3

roslaunch ros_training turtlesim_js.launch

Joy ROS package

http://wiki.ros.org/joy

- Generic Linux package to control a Joystick
- Contains ROS driver for the Joystick
- Complete documentation is at http://wiki.ros.org/joy/Tutorials/WritingTeleopNode
- The joy package contains joy_node, a node that interfaces a generic Linux joystick to ROS. This node publishes a "Joy" message (which contains the current state of each one of the joystick's axes) to **turtle1/cmd_vel**
- **Node name**: **joy_node** **Topic name**:**joy** **msg name:sensor_msgs/Joy**



sensor_msgs/Joy Message

File: sensor_msgs/Joy.msg

Raw Message Definition

```
# Reports the state of a joysticks axes and buttons.
Header header           # timestamp in the header is the time the data is received from the joystick
float32[] axes          # the axes measurements from a joystick
int32[] buttons         # the buttons measurements from a joystick
```

Joy message type

Section:4

Use a XBOX controller to control a drone in ROS

What is Hector?

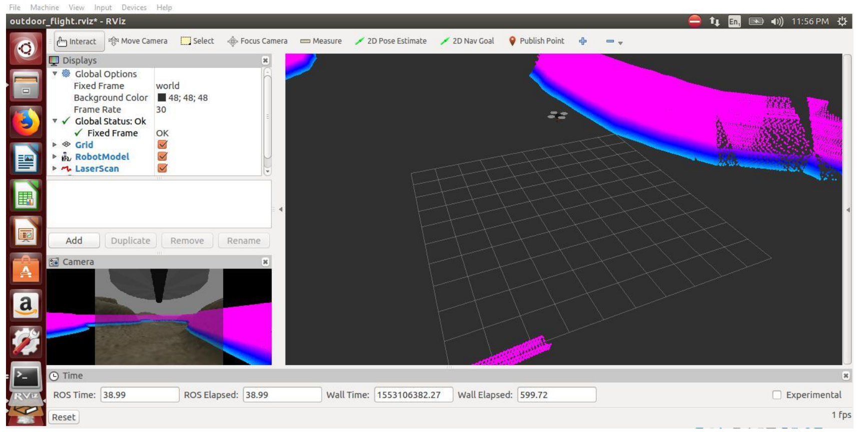
- Simulated ROS quadcopter made for software testing and simulation
- Hardware cost due to crash can be very high for quadrotors so it is an ideal way to test a quadcopter
- The meta package is under **hector_quadrotor**
- It comes with both indoor and outdoor scenes for flying
- An xbox controller can be used to control the quadrotor using **xbox_controller.launch**
- Supports both Rviz and Gazebo

Install by:

- **sudo apt-get install ros-hydro-hector-quadrotor-demo**
- Remember to change the version with the ROS version you are using example use for Indigo
  sudo apt-get install ros-indigo-hector-quadrotor-demo

Install from source:
http://wiki.ros.org/hector_quadrotor/Tutorials/Quadrotor%20outdoor%20flight%20demo

Lab # Run Hector with Xbox

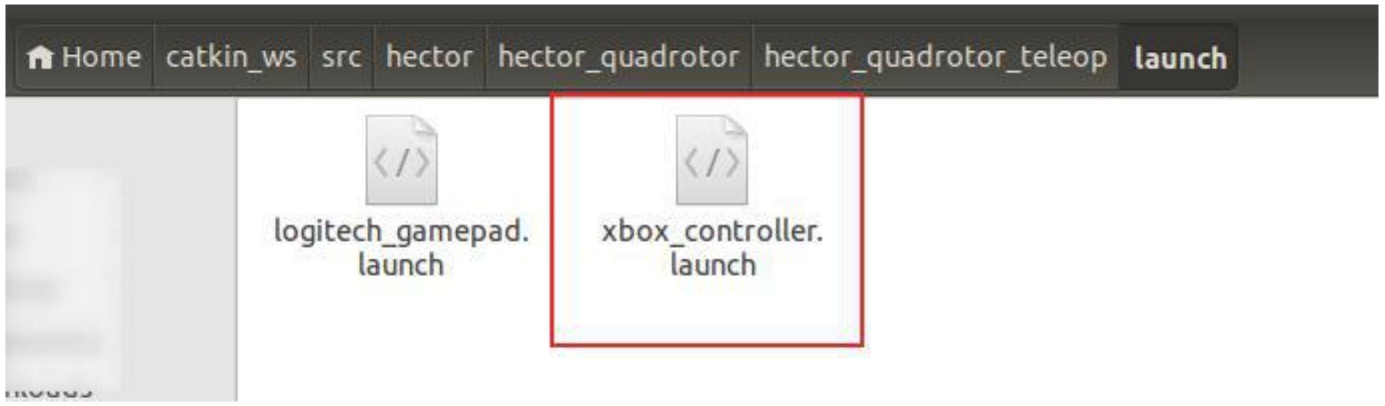


```
Fixed Frame          world
Background Color     48; 48; 48
Frame Rate           30
Global Status: Ok
Fixed Frame          OK
```

```
ROS Time: 38.99   ROS Elapsed: 38.99   Wall Time: 1553106382.27   Wall Elapsed: 599.72
```

roslaunch hector_quadrotor_demo outdoor_flight_gazebo.launch

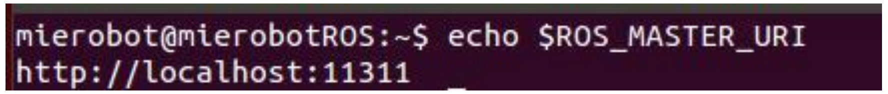
roslaunch hector_quadrotor_teleop xbox_controller.launch



Prerequisites:

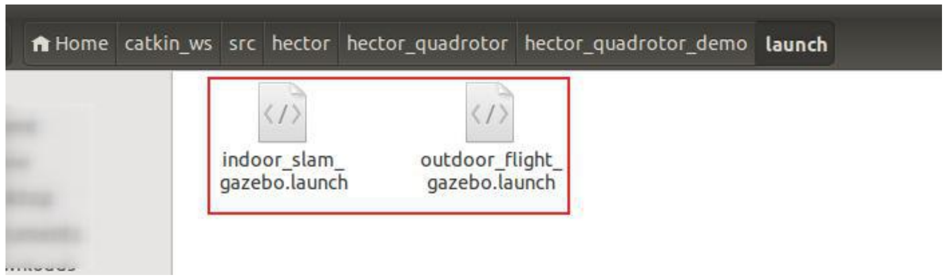
jstest-gtk (This should show JS movement)

echo $ROS_MASTER_URI

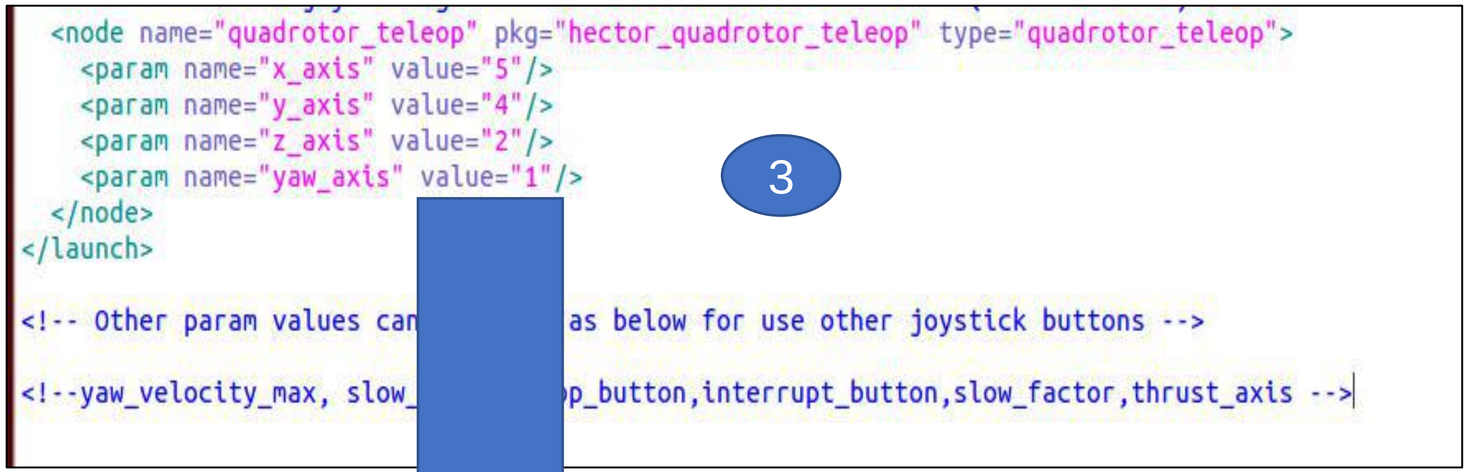


```
mierobot@mierobotROS:~$ echo $ROS_MASTER_URI
http://localhost:11311
```

$ export ROS_MASTER_URI=http://localhost:11311 (Use this command if you see no result with ROS_MASTER_URI)



Important parameters



```
  <node name="quadrotor_teleop" pkg="hector_quadrotor_teleop" type="quadrotor_teleop">
    <param name="x_axis" value="5"/>
    <param name="y_axis" value="4"/>
    <param name="z_axis" value="2"/>
    <param name="yaw_axis" value="1"/>
  </node>
</launch>

<!-- Other param values can          as below for use other joystick buttons -->

<!--yaw_velocity_max, slow_          p_button,interrupt_button,slow_factor,thrust_axis -->
```

Other parameters for quadrotor_type:

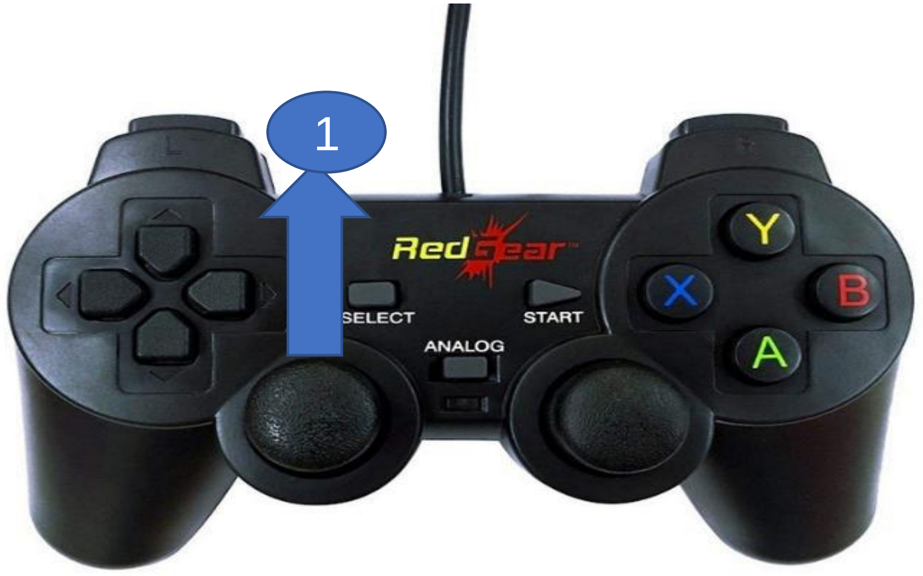
thrust_axis,
yaw_velocity_max,
slow_button, go_button,
stop_button,
interrupt_button,
slow_factor

3

2

Value from JS =0
Add +1

Value at Param =
0+1=1



Installation section

Setting up your Catkin workspace and GitHub

```
$ export ROS_MASTER_URI=http://localhost:11311
```
(Use this command if you see no result with ROS_MASTER_URI)