COMP20170

Introduction to Robotics

Assoc. Prof. Eleni Mangina

Room B2.05

Teaching Assistant & Mentor:

Evan O'Keeffe (PhD student – IRC Scholar)

School of Computer Science

Ext. 2858

eleni.mangina@ucd.ie

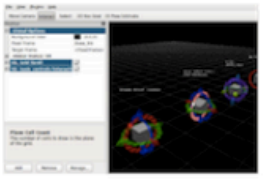
ROS.org

- ROS architecture & philosophy
- ROS master, nodes, and topics
- Console commands
- Catkin workspace and build system
- Launch-files
- Gazebo simulator

ROS.org

What is ROS?

ROS = Robot Operating System

ros.org

Plumbing	Tools	Capabilities	Ecosystem
▪ Process management ▪ Inter-process communication ▪ Device drivers	▪ Simulation ▪ Visualization ▪ Graphical user interface ▪ Data logging	▪ Control ▪ Planning ▪ Perception ▪ Mapping ▪ Manipulation	▪ Package organization ▪ Software distribution ▪ Documentation ▪ Tutorials

ROS.org

History of ROS

- Originally developed in 2007 at the Stanford Artificial Intelligence Laboratory
- Since 2013 managed by OSRF
- Today used by many robots, universities and companies
- De facto standard for robot programming

ros.org

ROS Philosophy

- **Peer to peer**
  Individual programs communicate over defined API (ROS *messages*, *services*, etc.).
- **Distributed**
  Programs can be run on multiple computers and communicate over the network.
- **Multi-lingual**
  ROS modules can be written in any language for which a client library exists (C++, Python, MATLAB, Java, etc.).
- **Light-weight**
  Stand-alone libraries are wrapped around with a thin ROS layer.
- **Free and open-source**
  Most ROS software is open-source and free to use.

ROS.org

ROS Workspace Environment

- Defines context for the current workspace
- Default workspace loaded with

```
source .bashrc
```

This is already setup in the provided installation.

Overlay your *catkin workspace* with

```
> cd ~/catkin_ws
> source devel/setup.bash
```

See setup with

```
> cat ~/.bashrc
```

Check your workspace with

```
> echo $ROS_PACKAGE_PATH
```

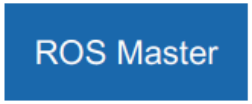
More info
http://wiki.ros.org/indigo/Installation/Ubuntu
http://wiki.ros.org/catkin/workspaces

```
lbros@lbros-VirtualBox:~/catkin_ws$ source devel/setup.bash
lbros@lbros-VirtualBox:~/catkin_ws$ echo $ROS_PACKAGE_PATH
/home/lbros/catkin_ws/src:/opt/ros/kinetic/share
lbros@lbros-VirtualBox:~/catkin_ws$
```

ROS.org

ROS Master

- Manages the communication between nodes
- Every node registers at startup with the master



Start a master with

```
> roscore
```

```
lbros@lbros-VirtualBox:~/catkin_ws$ roscore
... logging to /home/lbros/.ros/log/0e9e3440-f681-11e7-be49-08002776b3d2/roslaun
ch-lbros-VirtualBox-12722.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://lbros-VirtualBox:43513/
ros_comm version 1.12.12


SUMMARY
========

PARAMETERS
 * /rosdistro: kinetic
 * /rosversion: 1.12.12

NODES

auto-starting new master
process[master]: started with pid [12733]
ROS_MASTER_URI=http://lbros-VirtualBox:11311/

setting /run_id to 0e9e3440-f681-11e7-be49-08002776b3d2
process[rosout-1]: started with pid [12746]
started core service [/rosout]
```

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

ROS.org

ROS Nodes

- Single-purpose, executable program
- Individually compiled, executed, and managed
- Organized in *packages*

Run a node with

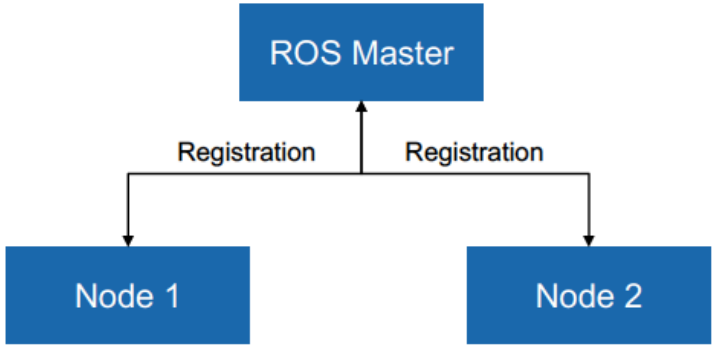
```
> rosrun package_name node_name
```

See active nodes with

```
> rosnode list
```

Retrieve information about a node with

```
> rosnode info node_name
```



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

ROS.org

ROS Topics

- Nodes communicate over *topics*
  - Nodes can *publish* or *subscribe* to a topic
  - Typically, 1 publisher and *n* subscribers
- Topic is a name for a stream of *messages*

List active topics with

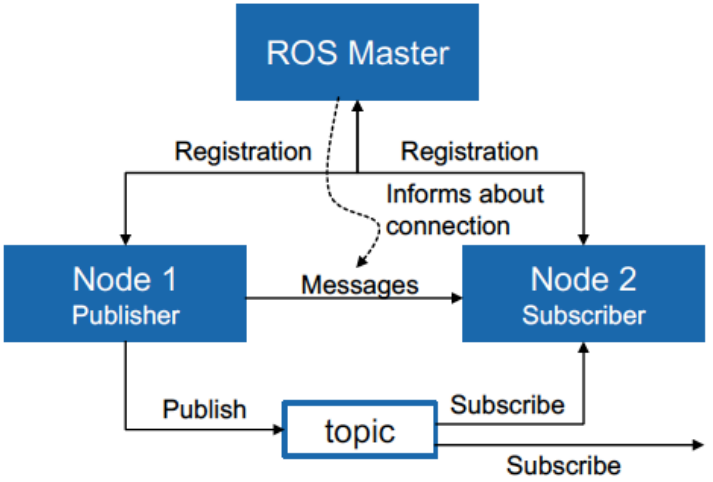
```
> rostopic list
```

Subscribe and print the contents of a topic with

```
> rostopic echo /topic
```

Show information about a topic with

```
> rostopic info /topic
```



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

ROS.org

ROS Messages

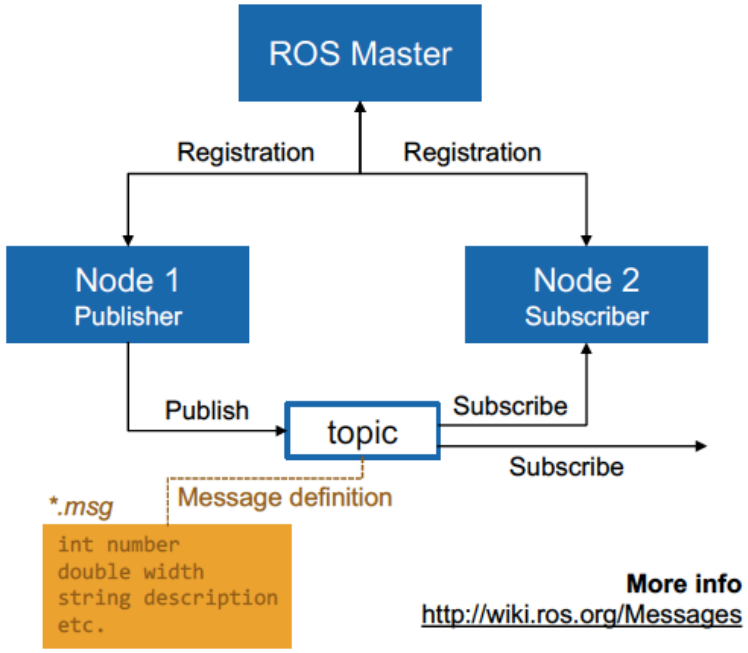
- Data structure defining the *type* of a topic
- Compromised of a nested structure of integers, floats, booleans, strings etc. and arrays of objects
- Defined in **.msg* files

See the type of a topic

```
> rostopic type /topic
```

Publish a message to a topic

```
> rostopic pub /topic type args
```



ROS.org

ROS Messages

Pose Stamped Example

geometry_msgs/Point.msg

```
float64 x
float64 y
float64 z
```

sensor_msgs/Image.msg

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

geometry_msgs/PoseStamped.msg

```
std_msgs/Header header
 uint32 seq
 time stamp
 string frame_id
geometry_msgs/Pose pose
  geometry_msgs/Point position
     float64 x
     float64 y
     float64 z
   geometry_msgs/Quaternion
orientation
     float64 x
     float64 y
     float64 z
     float64 w
```

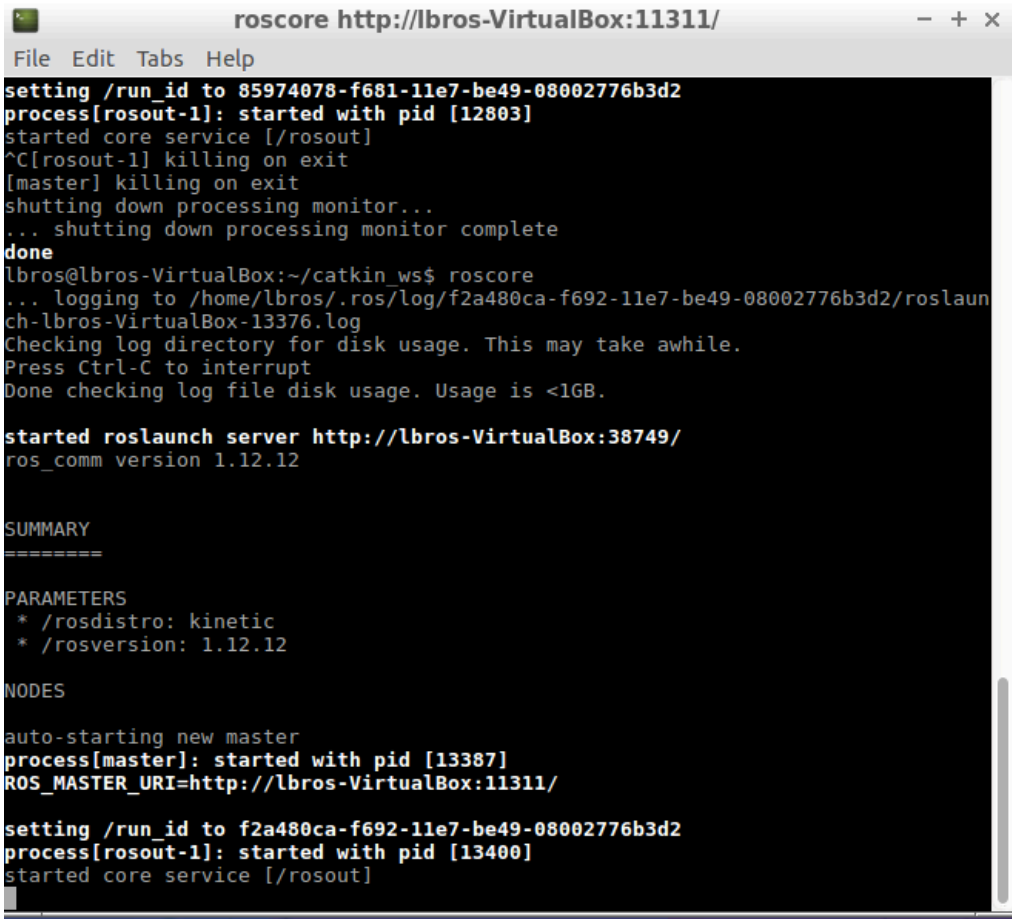
ROS.org

Example

Console Tab Nr. 1 – Starting a *roscore*

Start a roscore with

```
> roscore
```



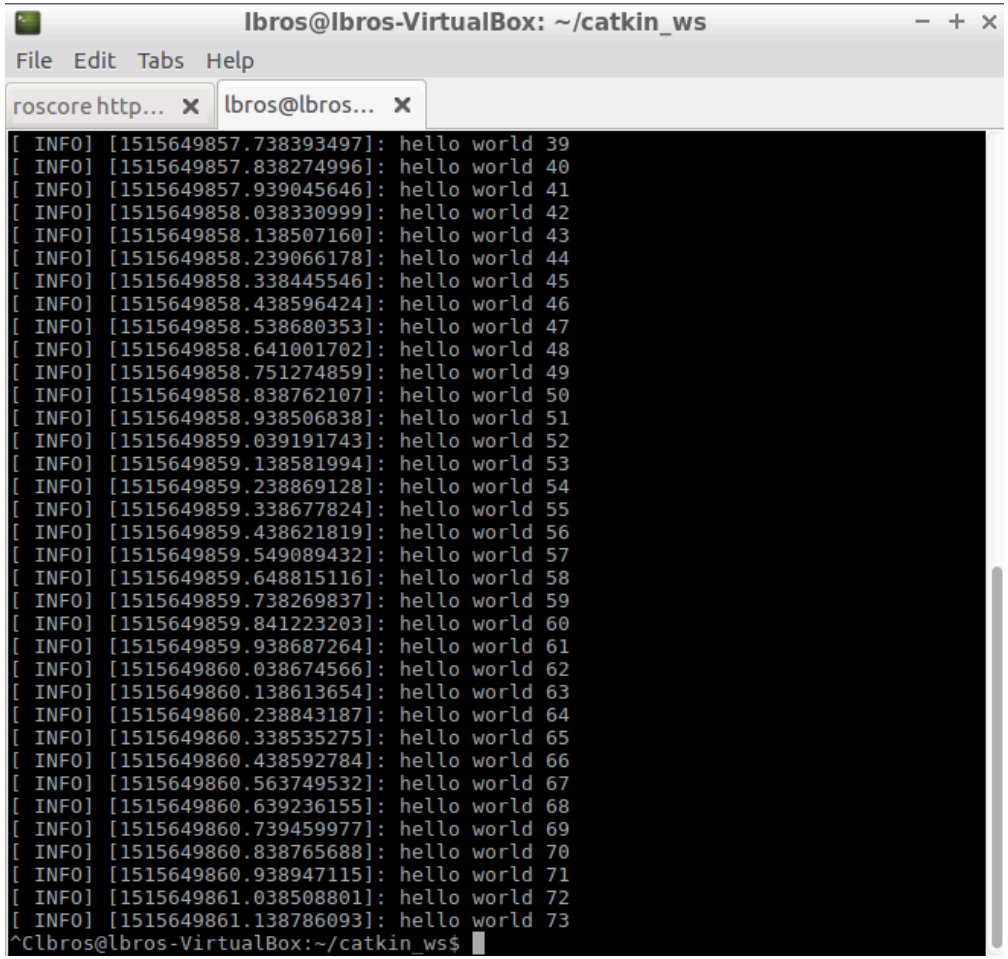
ROS.org

Example

Console Tab Nr. 2 – Starting a *talker* node

Run a talker demo node with

```
> rosrun roscpp_tutorials talker
```



ROS.org

Example

Console Tab Nr. 3 – Analyze *talker* node

See the list of active nodes

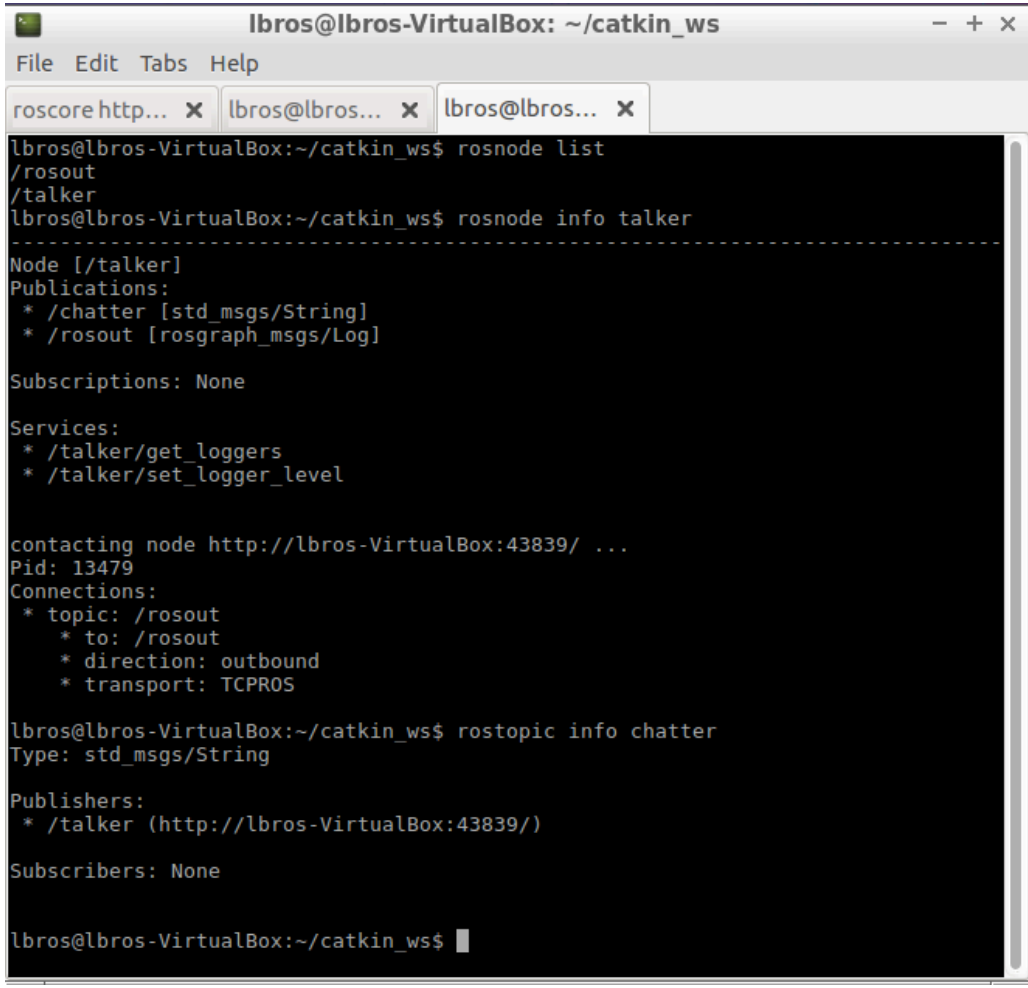
```
> rosnode list
```

Show information about the *talker* node

```
> rosnode info /talker
```

See information about the *chatter* topic

```
> rostopic info /chatter
```



ROS.org

Example

Console Tab Nr. 3 – Analyze *chatter* topic

Check the type of the *chatter* topic

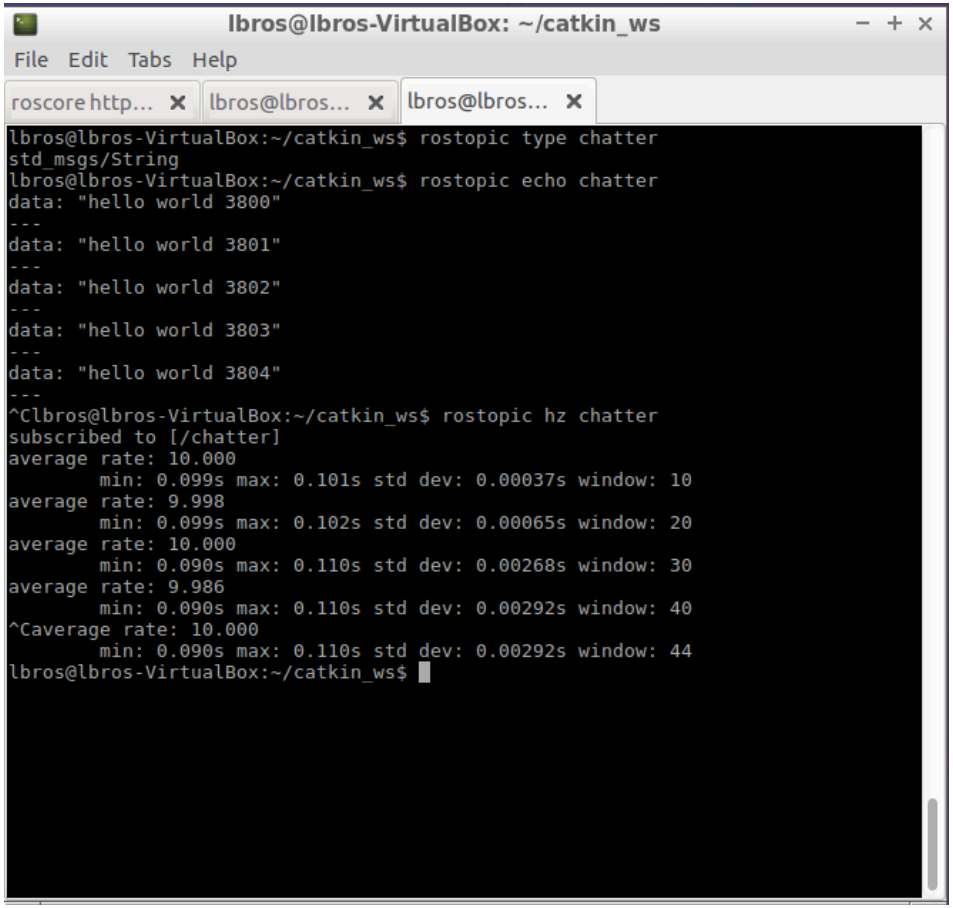
```
> rostopic type /chatter
```

Show the message contents of the topic

```
> rostopic echo /chatter
```

Analyze the frequency

```
> rostopic hz /chatter
```



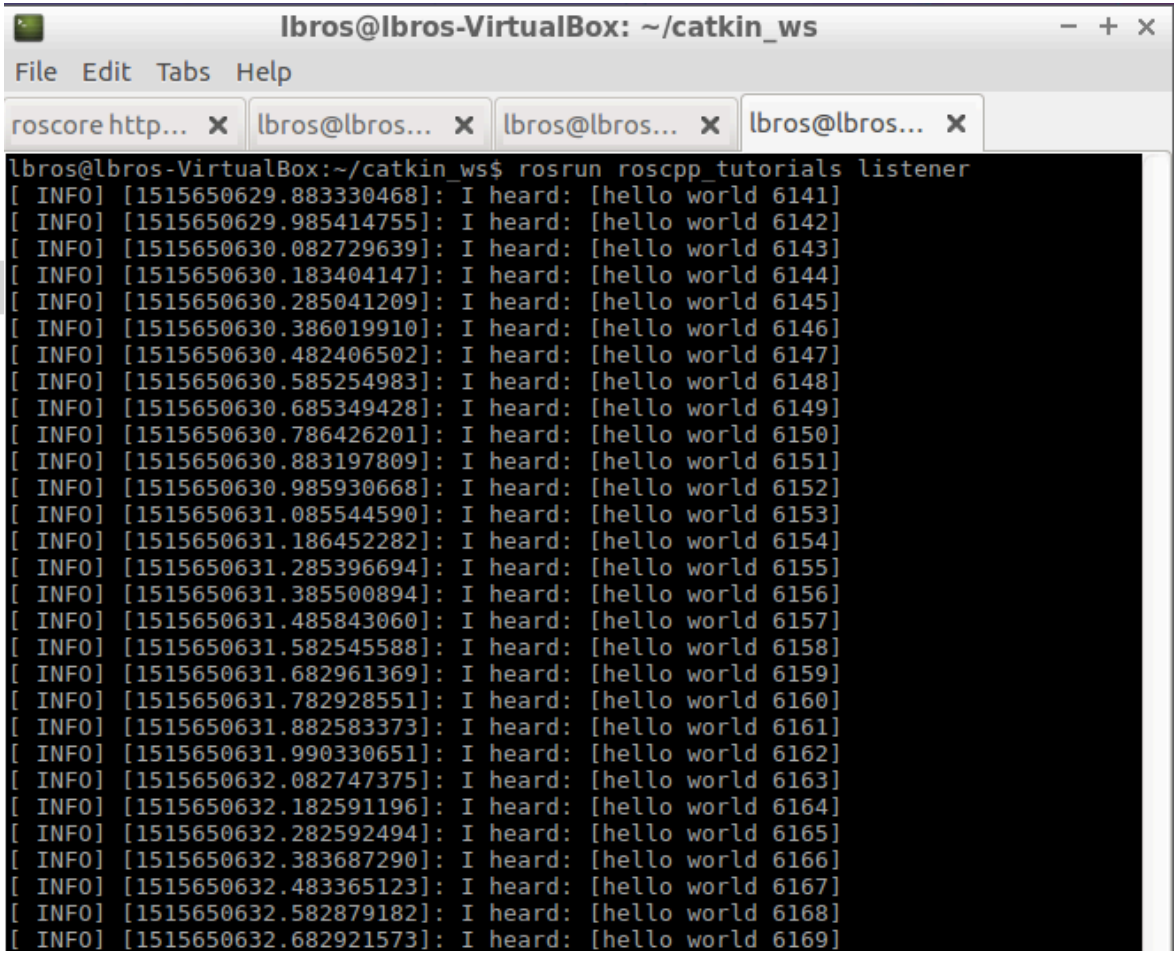
ROS.org

Example

Console Tab Nr. 4 – Starting a *listener* node

Run a listener demo node with

```
> rosrun roscpp_tutorials listener
```



ROS.org

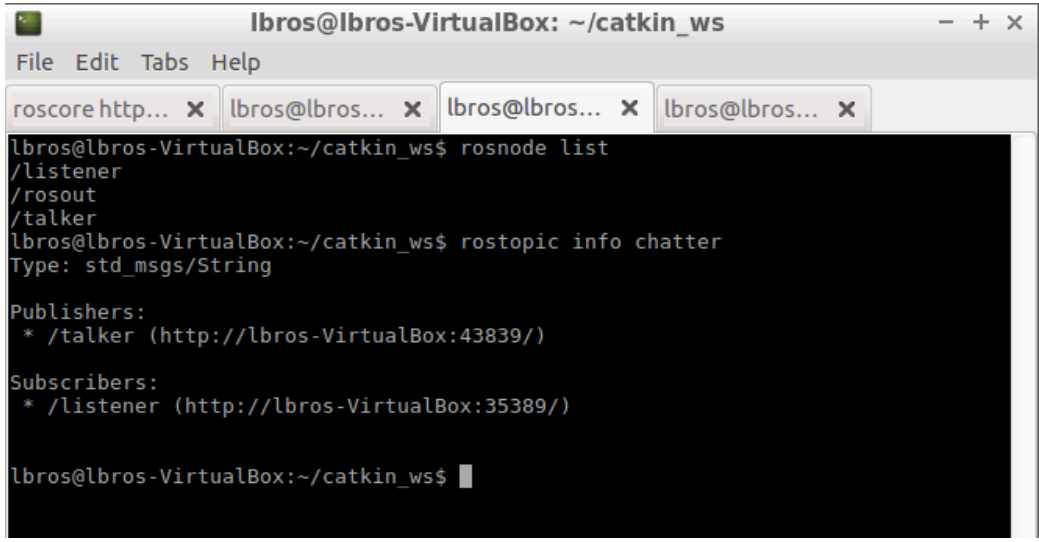
Example
Console Tab Nr. 3 – Analyze

See the new *listener* node with

```
> rosnode list
```

Show the connection of the nodes over the chatter topic with

```
> rostopic info /chatter
```



ROS.org

Example

Console Tab Nr. 3 – Publish Message from Console

Close the *talker* node in console nr. 2 with `Ctrl` + `C`

Publish your own message with

```
[ INFO] [1515650917.681739261]: hello world 9019
[ INFO] [1515650917.783052753]: hello world 9020
^C[ INFO] [1515650917.882553801]: hello world 9021
lbros@lbros-VirtualBox:~/catkin_ws$ rostopic pub /chatter std_msgs/String "data:
 COMP20170 Introduction to Robotics Module"
publishing and latching message. Press ctrl-C to terminate
```

Check the output of the *listener* in console nr. 4

```
[ INFO] [1515650917.582120782]: I heard: [hello world 9018]
[ INFO] [1515650917.682179493]: I heard: [hello world 9019]
[ INFO] [1515650917.783547353]: I heard: [hello world 9020]
[ INFO] [1515651125.734036397]: I heard: [COMP20170 Introduction to Robotics Mod
ule]
```

ROS.org

catkin Build System

- *catkin* is the ROS build system to generate executables, libraries, and interfaces
- We suggest to use the *Catkin Command Line Tools*

➔ Use **`catkin build`** instead of **`catkin_make`**

Navigate to your catkin workspace with

```
> cd ~/catkin_ws
```

Build a package with

```
> catkin build package_name
```

! Whenever you build a **new** package, update your environment

```
> source devel/setup.bash
```

The catkin command line tools are pre-installed in the provided installation.

More info
http://wiki.ros.org/catkin/Tutorials
https://catkin-tools.readthedocs.io/

ROS.org

catkin Build System

The catkin workspace contains the following spaces

Work here

src

The *source space* contains the source code. This is where you can clone, create, and edit source code for the packages you want to build.

Don't touch

build

The *build space* is where CMake is invoked to build the packages in the source space. Cache information and other intermediate files are kept here.

Don't touch

devel

The *development (devel) space* is where built targets are placed (prior to being installed).

If necessary, clean the entire build and devel space with

```
> catkin clean
```

More info
http://wiki.ros.org/catkin/workspaces

ROS.org

catkin Build System

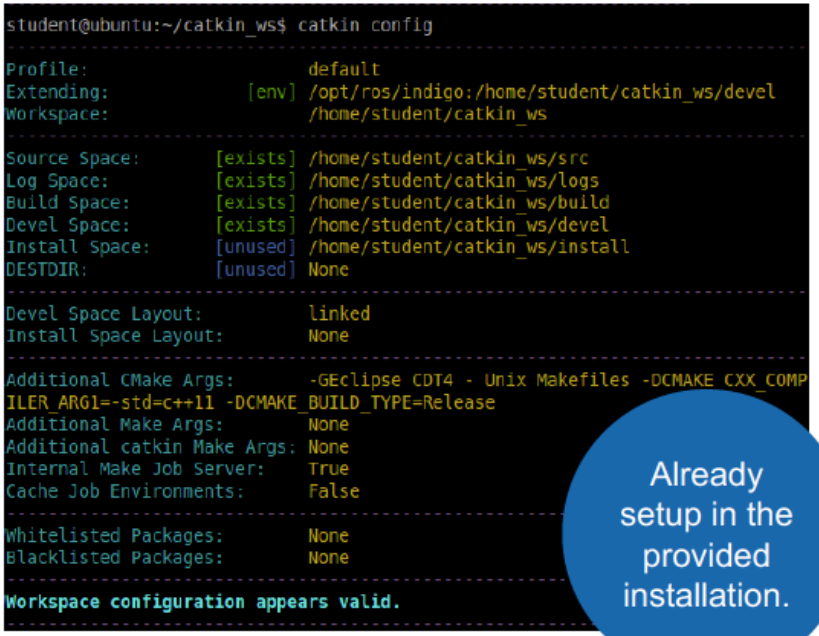
The catkin workspace setup can be checked with

```
> catkin config
```

For example, to set the *CMake build type* to Release (or Debug etc.), use

```
> catkin build --cmake-args
      -DCMAKE_BUILD_TYPE=Release
```

More info
http://catkin-tools.readthedocs.io/en/latest/verbs/catkin_config.html
http://catkin-tools.readthedocs.io/en/latest/cheat_sheet.html



Already setup in the provided installation.

ROS.org

Example

Open a terminal and browse to your `git` folder

```
> cd ~/git
```

Clone the Git repository with

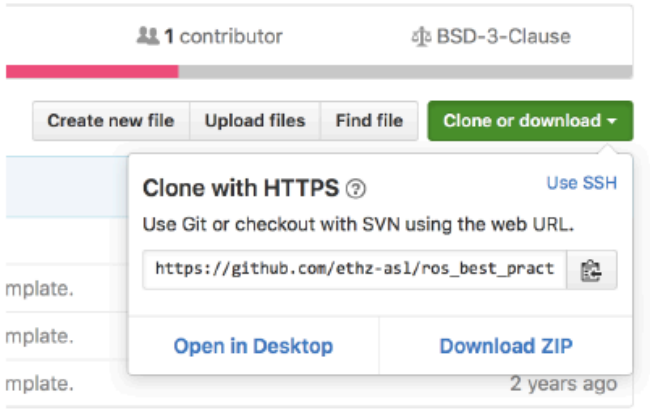
```
> git clone https://github.com/ethz-asl/ros_best_practices.git
```

Symlink the new package to your catkin workspace

```
> ln -s ~/git/ros_best_practices/ ~/catkin_ws/src/
```

Note: You could also directly clone to your catkin workspace, but using a common `git` folder is convenient if you have multiple catkin workspaces.

https://github.com/ethz-asl/ros_best_practices



ROS.org

Example

Go to your catkin workspace

```
> cd ~/catkin_ws
```

Build the package with

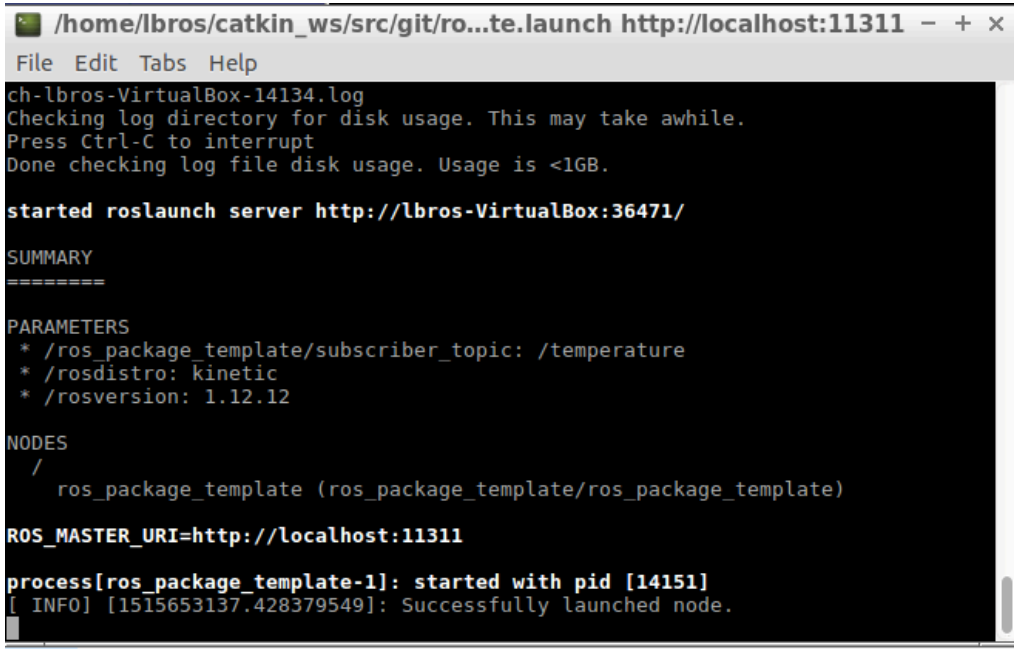
```
lbros@lbros-VirtualBox:~/catkin_ws$ catkin_make ros_package_template
```

Re-source your workspace setup

```
> source devel/setup.bash
```

Launch the node with

```
> roslaunch ros_package_template
  ros_package_template.launch
```



ROS.org

ROS Launch

- *launch* is a tool for launching multiple nodes (as well as setting parameters)
- Are written in XML as **.launch* files
- If not yet running, launch automatically starts a roscore

Browse to the folder and start a launch file with

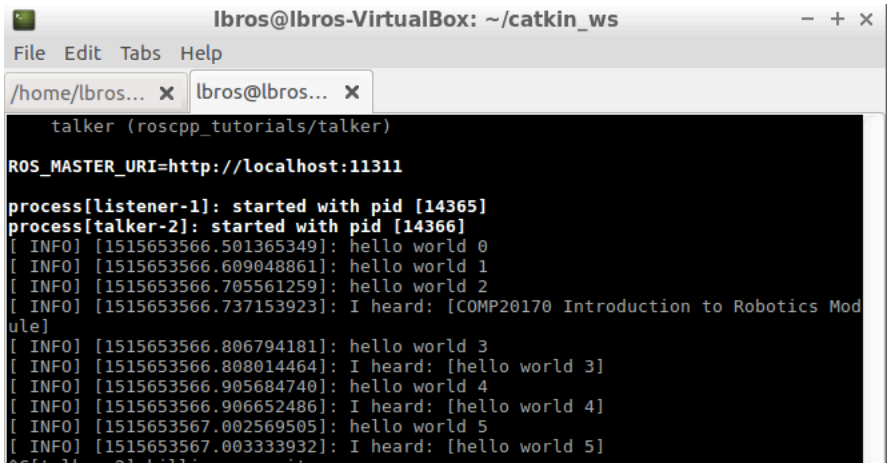
```
> roslaunch file_name.launch
```

Start a launch file from a package with

```
> roslaunch package_name file_name.launch
```

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

Example console output for
`roslaunch roscpp_tutorials talker_listener.launch`



ROS.org

ROS Launch
File Structure

talker_listener.launch

```
<launch>
    <node name="listener" pkg="roscpp_tutorials" type="listener" output="screen"/>
    <node name="talker" pkg="roscpp_tutorials" type="talker" output="screen"/>
</launch>
```

! Notice the syntax difference for self-closing tags: `<tag></tag>` and `<tag/>`

- **`launch`**: Root element of the launch file
- **`node`**: Each *<node>* tag specifies a node to be launched
- **`name`**: Name of the node (free to choose)
- **`pkg`**: Package containing the node
- **`type`**: Type of the node, there must be a corresponding executable with the same name
- **`output`**: Specifies where to output log messages (`screen`: console, `log`: log file)

More info
http://wiki.ros.org/roslaunch/XML
http://wiki.ros.org/roslaunch/Tutorials/Roslaunch%20tips%20for%20larger%20projects

ROS.org

ROS Launch
Arguments

- Create re-usable launch files with <arg> tag, which works like a parameter (default optional)

```
<arg name="arg_name" default="default_value"/>
```

- Use arguments in launch file with

```
$(arg arg_name)
```

- When launching, arguments can be set with

```
> roslaunch launch_file.launch arg_name:=value
```

range_world.launch (simplified)

```
<?xml version="1.0"?>
<launch>
  <arg name="use_sim_time" default="true"/>
  <arg name="world" default="gazebo_ros_range"/>
  <arg name="debug" default="false"/>
  <arg name="physics" default="ode"/>

  <group if="$(arg use_sim_time)">
    <param name="/use_sim_time" value="true" />
  </group>

  <include file="$(find gazebo_ros)
                                /launch/empty_world.launch">
    <arg name="world_name" value="$(find gazebo_plugins)/
                    test/test_worlds/$(arg world).world"/>
    <arg name="debug" value="$(arg debug)"/>
    <arg name="physics" value="$(arg physics)"/>
  </include>
</launch>
```

More info
http://wiki.ros.org/roslaunch/XML/arg

ROS.org

ROS Launch
Including Other Launch Files

- Include other launch files with <include> tag to organize large projects

```
<include file="package_name"/>
```

- Find the system path to other packages with

```
$(find package_name)
```

- Pass arguments to the included file

```
<arg name="arg_name" value="value"/>
```

range_world.launch (simplified)

```
<?xml version="1.0"?>
<launch>
  <arg name="use_sim_time" default="true"/>
  <arg name="world" default="gazebo_ros_range"/>
  <arg name="debug" default="false"/>
  <arg name="physics" default="ode"/>

  <group if="$(arg use_sim_time)">
    <param name="/use_sim_time" value="true" />
  </group>

  <include file="$(find gazebo_ros)
                                /launch/empty_world.launch">
    <arg name="world_name" value="$(find gazebo_plugins)/
                    test/test_worlds/$(arg world).world"/>
    <arg name="debug" value="$(arg debug)"/>
    <arg name="physics" value="$(arg physics)"/>
  </include>
</launch>
```

More info
http://wiki.ros.org/roslaunch/XML/include

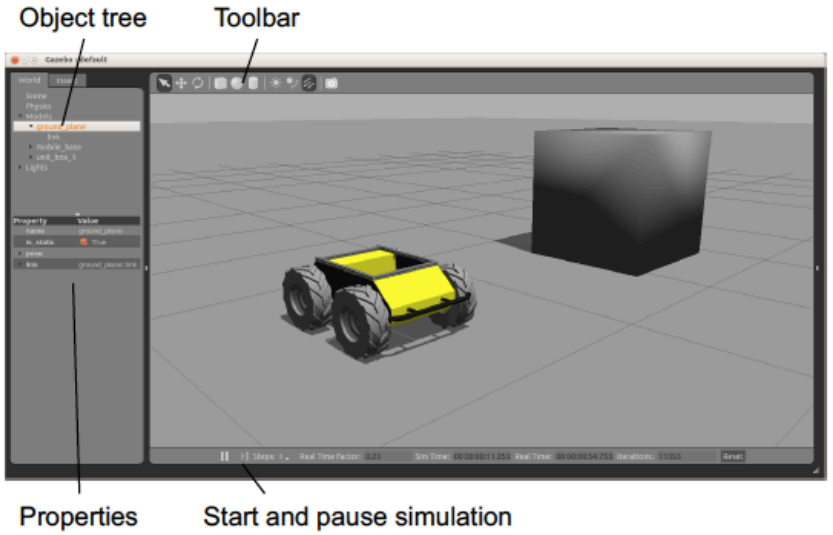
ROS.org

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

Further References

- **ROS Wiki**
  - http://wiki.ros.org/
- **Installation**
  - http://wiki.ros.org/ROS/Installation
- **Tutorials**
  - http://wiki.ros.org/ROS/Tutorials
- **Available packages**
  - http://www.ros.org/browse/
- **ROS Cheat Sheet**
  - https://github.com/ros/cheatsheet/releases/download/0.0.1/ROScheatsheet_catkin.pdf
- **ROS Best Practices**
  - https://github.com/ethz-asl/ros_best_practices/wiki
- **ROS Package Template**
  - https://github.com/ethz-asl/ros_best_practices/tree/master/ros_package_template