Software Architectures for Robotics

Lab Session I

Installing ROS and basic commands

UNIVERSITÀ DEGLI STUDI DI GENOVA | Dibris

EMARO LAB

Table of contents

Installing ROS

Always start from the tutorial page!

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

Let's look for an installation tutorial...

http://wiki.ros.org/kinetic/Installation/Ubuntu

Follow the steps :)

Installing ROS

Which **version** of ROS should I choose?

It depends!

- If unsure, go for `ros-kinetic-desktop-full` (rqt, rviz, gazebo, navigation and perception)
- Use `ros-kinetic-desktop` if you do not plan to use a lot of default libraries or Gazebo
- Use `ros-kinetic-ros-base` for embedded projects

Installing ROS

Source your installation!

Sourcing tells your system where your installation is!

```
$ source /opt/ros/<distro>/setup.bash
```

- Only the current shell will be sourced this way!
- Add the source to `.bashrc` in your home folder

Installing ROS

Remember to initialize `rosdep`:

```
$ sudo rosdep
$ init rosdep update
```

`rosdep` is necessary for ROS to work.

It can be used to install dependencies for your sources

Creating your first workspace

What is a workspace?

It is a folder which contains your **sources** and compiled **binaries.**

It **depends** on the build tool used, so don't try to build with the wrong tool!

We will use the `catkin_make` tool.

Creating your first workspace

Creating a workspace is easy:

```
$ mkdir -p ~/<ws_name>/src
$ cd ~/<ws_name>/
$ catkin_make
```

If `catkin_make` is run on an empty folder, it will initialize a new workspace.

Again, remember to source your workspace:

```
source ~/<your_usr>/<ws_name>/devel/setup.bash
```

The `catkin_make` command

If `catkin_make` is called on a workspace that contains packages in `/src`, it will **build** them.

```
$ catkin_make -j4          //defines max processors n
$ catkin_make clean        //cleans the workspace
$ catkin_make --pkg <p> //specify a single package
```

 `$ catkin_make -j4` is particularly useful on slow machines or under heavy loads to avoid freezes!

Basic Functions

Some recurrent ROS commands:

```
$ roscore                          //launches the ros core
$ rosrun <pkg> <node>              //runs a node
$ roslaunch <pkg> <launch>         //launches a launchfile

$ rostopic list                    //lists all topics
$ rostopic echo <topic>            //echoes msgs on topic

$ rosmsg show <msg>                //prints msg fields

$ rosservice ...
$ rossrv ...
```

Basic Functions

Try to run some commands!

First, we advice to get `Terminator` terminal emulator.

```
$ sudo add-apt-repository ppa:gnome-terminator
$ sudo apt-get update
$ sudo apt-get install terminator
```

`Terminator` is useful to monitor nodes running together.

`ctrl+shift+o` / `ctrl+shift+e` to open new panes.

Basic Functions

Try to run some commands!

In the first pane, run the ROS core

```
$ roscore
```

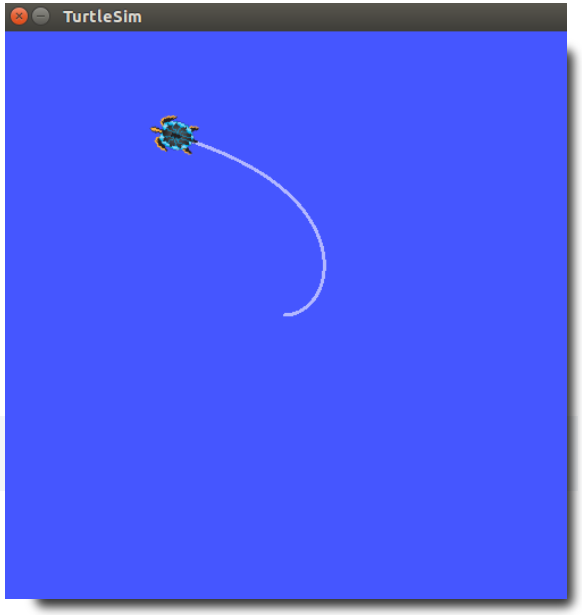
In the second pane, run:

```
$ rosrun turtlesim turtlesim_node
```

In third pane, run:

```
$ rosrun turtlesim turtle_teleop_key
```

Basic Functions



Try to run some commands!

In the first pane, run the ROS core

```
$ roscore
```

In the second pane, run:

```
$ rosrun turtlesim turtlesim_node
```

In third pane, run:

```
$ rosrun turtlesim turtle_teleop_key
```

Basic Functions

You can move the turtle with the arrow keys.

The two processes are communicating through a publish/subscribe protocol over a topic

In a fourth pane, run:

```
$ rostopic echo /turtle1/cmd_vel
```

Now you can see the velocity messages exchanged!

Basic Functions

Let's investigate further...

```
$ rostopic info /turtle1/cmd_vel
```

```
$ rosmsg info geometry_msgs/Twist
```

```
emaro@emaro-box:~$ rosmsg info geometry_msgs/Twist
geometry_msgs/Vector3 linear
  float64 x
  float64 y
  float64 z
geometry_msgs/Vector3 angular
  float64 x
  float64 y
  float64 z

emaro@emaro-box:~$
```

Basic Functions

Now try publishing a velocity command:

```
$ rostopic pub /turtle1/cmd_vel
geometry_msgs/Twist "linear:
  x: 1.0
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 0.0"
```

Basic Functions

All basic commands works similarly!

You can refer to the command's help for details:

```
$ <command> -h
```

Try out `rostopic`, `rosmsg`, `rosservice`, `rossrv`, `rosparam`, `...`

Basic Functions

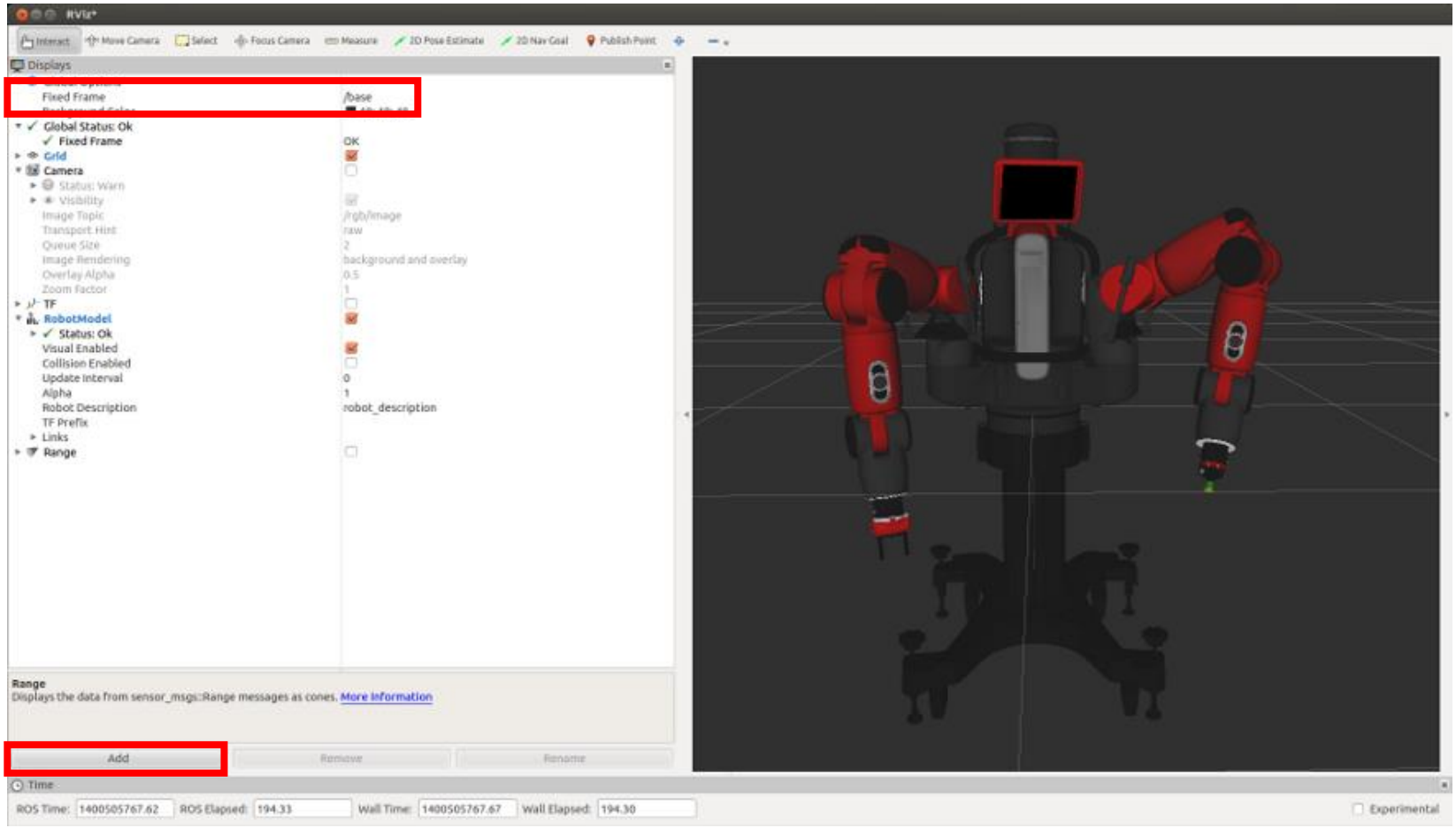
`rostopic` ≠ `rosmsg`

`rostopic` is used to interact with active topics, it **won't** run without a `roscore`

`rosmsg` is used to visualize the messages installed on the system

Obviously, `rosservice` ≠ `rossrv` too...

RViz


RViz*
Interact
Move Camera
Select
Focus Camera
Measure
2D Pose Estimate
2D Nav Goal
Publish Point
Displays
Fixed Frame
/base
Global Status: Ok
Fixed Frame
OK
Grid
Camera
Status: Warn
Visibility
Image Topic
/rgb/image
Transport Hint
raw
Queue Size
2
Image Rendering
background and overlay
Overlay Alpha
0.5
Zoom Factor
1
TF
RobotModel
Status: Ok
Visual Enabled
Collision Enabled
Update Interval
0
Alpha
1
Robot Description
robot_description
TF Prefix
Links
Range
Range
Displays the data from sensor_msgs::Range messages as cones. More Information
Add
Remove
Rename
Time
ROS Time: 1400505767.62
ROS Elapsed: 194.33
Wall Time: 1400505767.67
Wall Elapsed: 194.30
Experimental

RViz

Pay attention that the `Fixed frame` is correct (usually it should be `Base`)

Click on `Add` to add more visualization tools, like:

- `Camera`
- `PointCloud2`
- `TF`

Later, you will be able to test them on the Baxter

rqt

Launch rqt (`roscore` must be running):

```
$ rqt
```

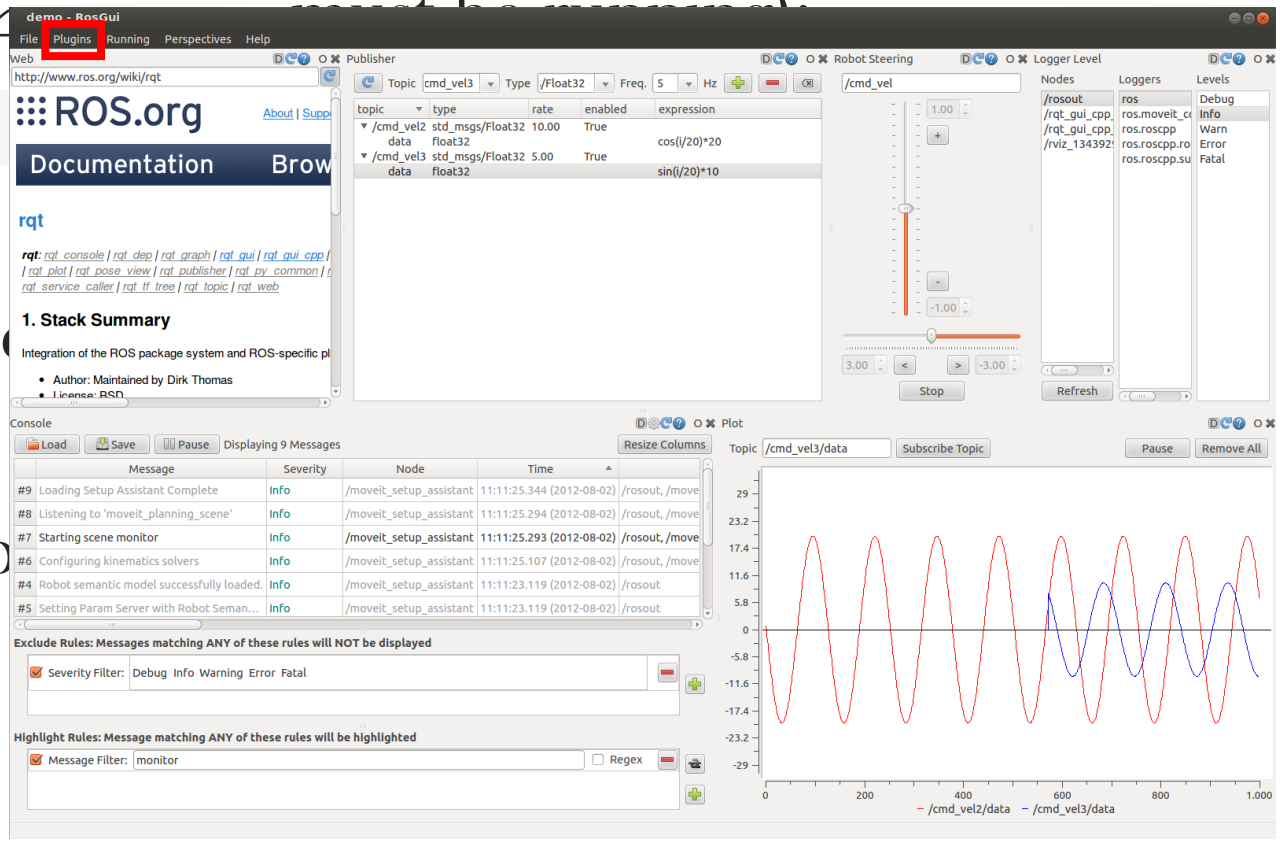
- You can add more views from the `Plugin` tab
- Views can be freely arranged on the screen

rqt

Launch rqt (r

```
$ rqt
```

- You can ad
- Views can b



rqt

The most useful function has a shortcut:

```
$ rqt_graph
```

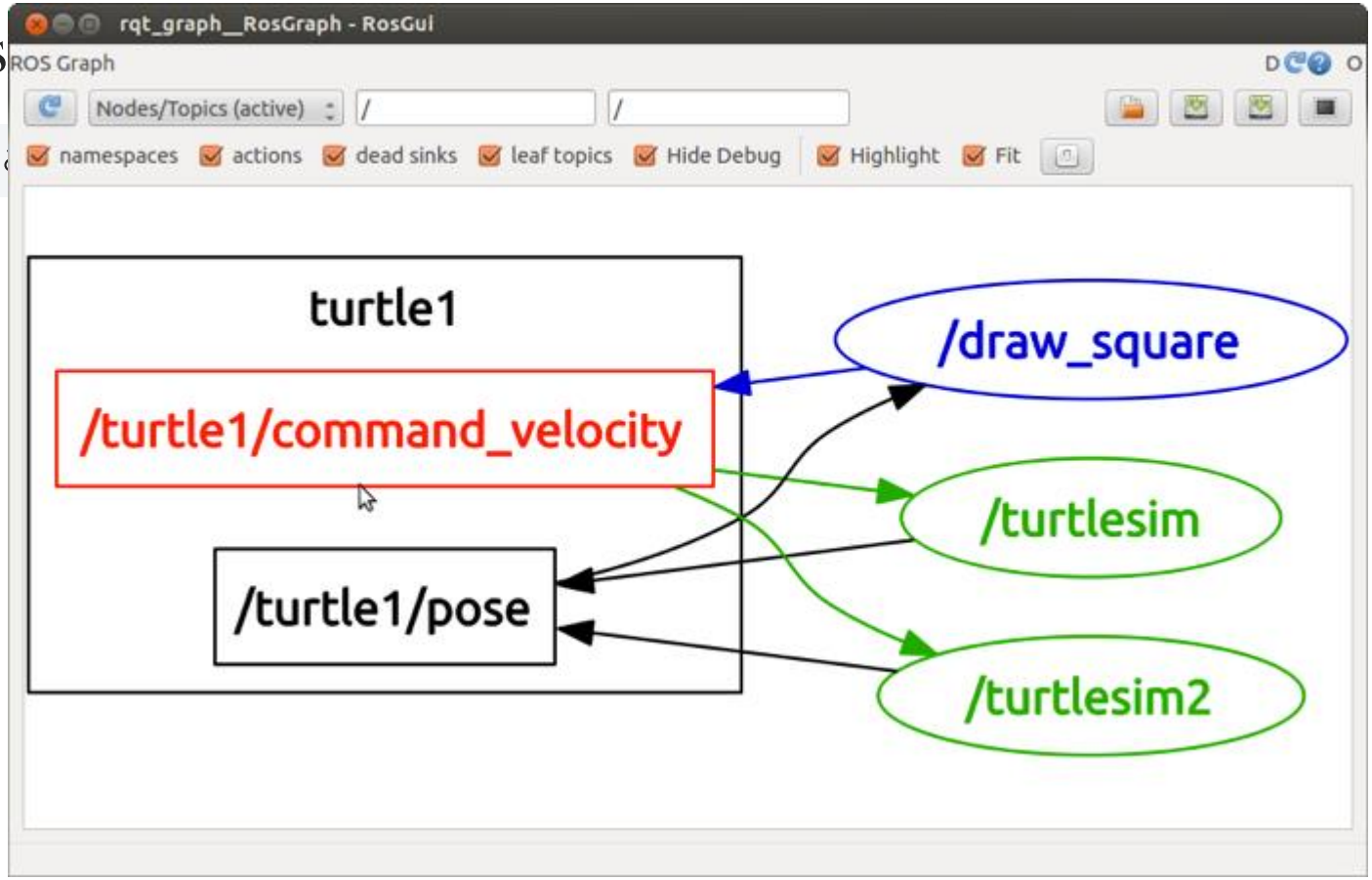
This will visualize **all publish/subcribe connections!**

rqt

The mos

```
$ rqt_gr
```

This will



Tips & Tricks

Use pipe `grep` to filter out your screen output:

```
$ rostopic list | grep <keyword>
$ rosmsg list | grep <keyword>
```

This is useful to reduce the number of candidates while searching a topic/service/parameter

Next Time...

- Creating a new package
- Dependencies in `CMakeLists & Package.xml`
- Adding `.msg` and `.srv` files
- Add excutables to your package
- Writing a publisher
- Writing a subscriber