ROS COMMANDS

ROBOTICS

POLITECNICO
MILANO 1863

FILE SYSTEM TOOLS

Ros Desktop-full come with lots of tutorials and tools

Before creating our own package and start writing some code we will learn how to navigate the ROS file system and use the turtlesim package to test some of the most useful tools

FILE SYSTEM TOOLS

Change directory in the ROS file system

roscd [package_name[/subdir]]

roscd roscpp && pwd	/opt/ros/kinetic/share/roscpp
roscd roscpp/srv	/opt/ros/kinetic/share/roscpp/srv
roscd robby_roboto	~/catkin_ws/src/robby_roboto

FILE SYSTEM TOOLS

Getting information about installed packages

rospack <subcommand> [options] [package]

subcommands (among the others)

depends [package]	package dependencies
find [package]	find package directory
list	list available packages

rospack find roscpp	/opt/ros/kinetic/share/roscpp
rospack list	<several packages>

STARTING THE MIDDLEWARE

To start the ROS middleware just type in a terminal

```
roscore
```

Now it is possible to display information about the elements currently running

```
rosnode list
rostopic list
rostopic echo /rosout
rosservice list
rqt_graph
```

DEALING WITH NODES

Getting information about running nodes

rosnode <command> [other_commands]

subcommands (among the others)

ping	test connectivity to node
info	print information about node
kill	kill a running node
cleanup	purge registration information of unreachable nodes

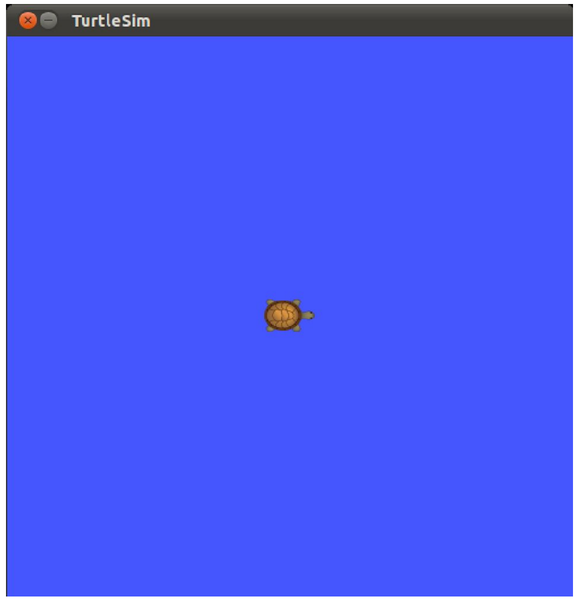
```
rosnode list
rosnode info /rosout
```

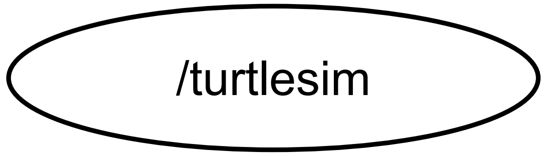
STARTING ROS NODES

To start a ROS node type in a terminal

rosrun [package_name] [node_name]

```
rosrun turtlesim turtlesim_node
rosnode ping /turtlesim
rosnode info /turtlesim
```





STARTING ROS NODES

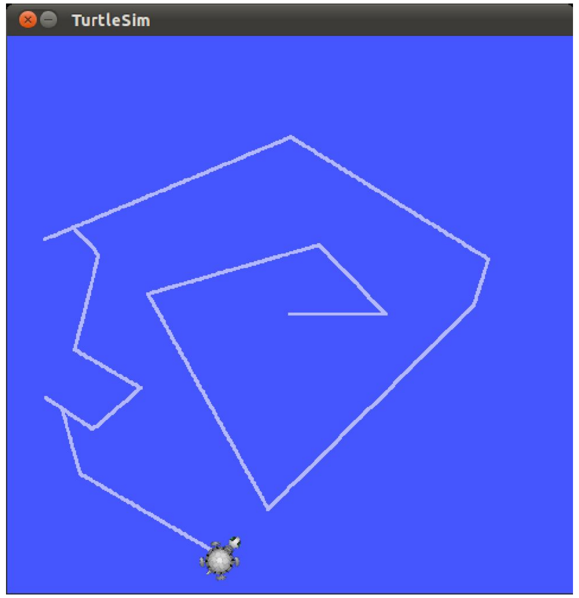
In a new terminal

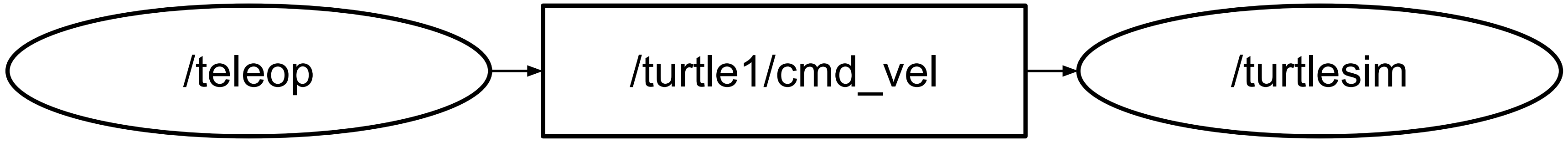
```
rosrun turtlesim turtle_teleop_key
```

Notes:

turtle_teleop_key is publishing the key strokes on a topic

turtlesim subscribes to the same topic to receive the key strokes





$$\begin{tikzcd}
\text{/teleop} \arrow[r] & \boxed{\text{/turtle1/cmd\_vel}} \arrow[r] & \text{/turtlesim}
\end{tikzcd}$$

DEALING WITH TOPICS

To show the running node type in a terminal

```
rqt_graph
```

To plot published data on a topic

```
rqt_plot /turtle1/pose/x /turtle1/pose/y
rqt_plot /turtle1/pose/x:y
```

To monitor a topic on a terminal type

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

DEALING WITH TOPICS CONT.

Getting information about ROS topics

rostopic <command> [topic_name]

subcommands (among the others)

echo	print messages to screen
find	find topics by type
hz	display publishing rate of topic
info	print information about active topic
list	list active topics
pub	publish data to topic
type	print topic type

DEALING WITH TOPICS CONT.

Getting information about ROS topics

```
rostopic type [topic_name]
```

```
rostopic type /turtle1/cmd_vel
```

Publishing ROS topics

```
rostopic pub [topic] [msg type] [args]
```

```
$ rostopic pub -1 /turtle1/cmd_vel geometry_msgs/Twist -- '[2.0, 0.0, 0.0]' '[0.0, 0.0, 1.8]'
```

DEALING WITH TOPICS CONT.

```
$ rostopic pub -1 /turtle1/cmd_vel geometry_msgs/Twist -- '[2.0, 0.0, 0.0]' '[0.0, 0.0, 1.8]'
```

The -1 option force rostopic to publish the message only once, if you want to publish the message at a specific frequency you will use:

```
$ rostopic pub /turtle1/cmd_vel geometry_msgs/Twist -r 1 -- '[2.0, 0.0, 0.0]' '[0.0, 0.0, 1.8]'
```

Where the -r 1 option specify that the message will be published at 1hz frequency

MESSAGES (ALSO SERVICES)

Getting information about msg/srv files

rosmsg <command> [msg/srv_file]

subcommands (among the others)

show	Display the fields in the msg/srv.
list	Display names of all msg/srv.
package	List all the msg/srv in a package.
packages	List all packages containing the msg/srv.

```
rosmsg show Pose
rosmsg package nav_msgs
```

DEALING WITH SERVICES

Calling services from command line and getting information:

rosservice <command> [other_commands]

subcommand (among the others)

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

```
rosservice call /reset
rosservice type /reset
```

BAGS

bag: file format to store messages data

Used to test different algorithm with the exact same input and to debug a system when it’s not monitorable at runtime

To record a bag use:

rosbag record

to record all the topics use:

$ rosbag record -a

to record only a subset of the topic use:

$ rosbag record topic1 topic2 etc

BAGS

To get info regarding a beg use the command:

```
$ rosbag info bag_name
```

To play a bag run:

```
$ rosbag play bag_name
```

remember that to run rosbag you need an active ros session (roscore should be on)

Always monitor your bag size, sometimes logging all the topics (if you are working with cameras) is not the best idea because you will produce more data/sec than your max disk writing speed.

CREATE THE ROS WORKSPACE

ROBOTICS

POLITECNICO MILANO 1863

CREATING THE WORKSPACE

ROS uses a custom compiling environment called **Catkin**

cmake/make with specific flags

Requires a workspace with a specific structure

Easy to setup and “easy” to use

```
mkdir -p ~/catkin_ws/src
cd ~/catkin_ws/
catkin_make
echo "source ~/catkin_ws/devel/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

WORKSPACE STRUCTURE

Source space (/src):

contains the source code of catkin packages.

Subfolder of this are the ROS packages you want to add to your system

All your stuff goes here!

Build space (/build):

space where cmake is invoked to build the catkin packages

Not where catkin_make is invoked!

cmake and catkin keep their cache information and other intermediate files here

Devel space (/devel):

Space where built targets are placed prior to being installed

PACKAGE CREATION

Command to create a new package

catkin_create_pkg [package_name] [depend1] [depend2] [depend3]

Before running the script cd to your src directory, then:

catkin_create_pkg beginner_tutorials std_msgs rospy roscpp

Important Notes

roscpp and rospy are client libraries to use C++ and Python

!!Before being able to do that you should have creates a ros_workspace!!

PACKAGE CREATION

cd to the new package, the script should have created:

-**CMakeLists.txt**

-**package.xml**

-**include** folder

-**src** folder

cd to your catkin workspace root to compile the new package, simply using **catkin_make**

EDITORS/ IDEs

ROBOTICS

POLITECNICO
MILANO 1863

ROSED

rosed is part of the rosbash suite

Allow the user to edit files using directly the package name, rather than typing the entire path

rosed [package_name] [filename]

rosed roscpp Logger.msg

The default editor is vim

You can edit the .bashrc file setting a more user friendly editor

IDEs

No official IDE by ROS

C++ editor with ROS specific plugins

On ROS wiki you can guides on how to properly configure the plugins

http://wiki.ros.org/IDEs

Simply add some features like easier compiling and some debug tools

Roboware

Based on Visual Studio

Designed for ROS

No need to install third parties plugin

Offers some functionalities:

- Run program directly inside Roboware
- Debugger
- Automatic file generation
- CMakeLists and Package.xml automatic update (partial)
- Integrated ros tool