



INFO 703

Master Advanced Mechatronics

Luc Marechal

2025

 ROS

ROS Command Tools

Tutorial 1

Objectives

At the end of this lecture, you are excepted to :

- Use ROS command line tools to get information on nodes, topics and message type
- Know what a ROS message is made up of.
- Find which library a ROS message comes from.
- Create a custom launch file.
- Achieve at least grade 80% of the Assignment

ROS Command Tools

Turtlesim

Turtle_teleop_key node



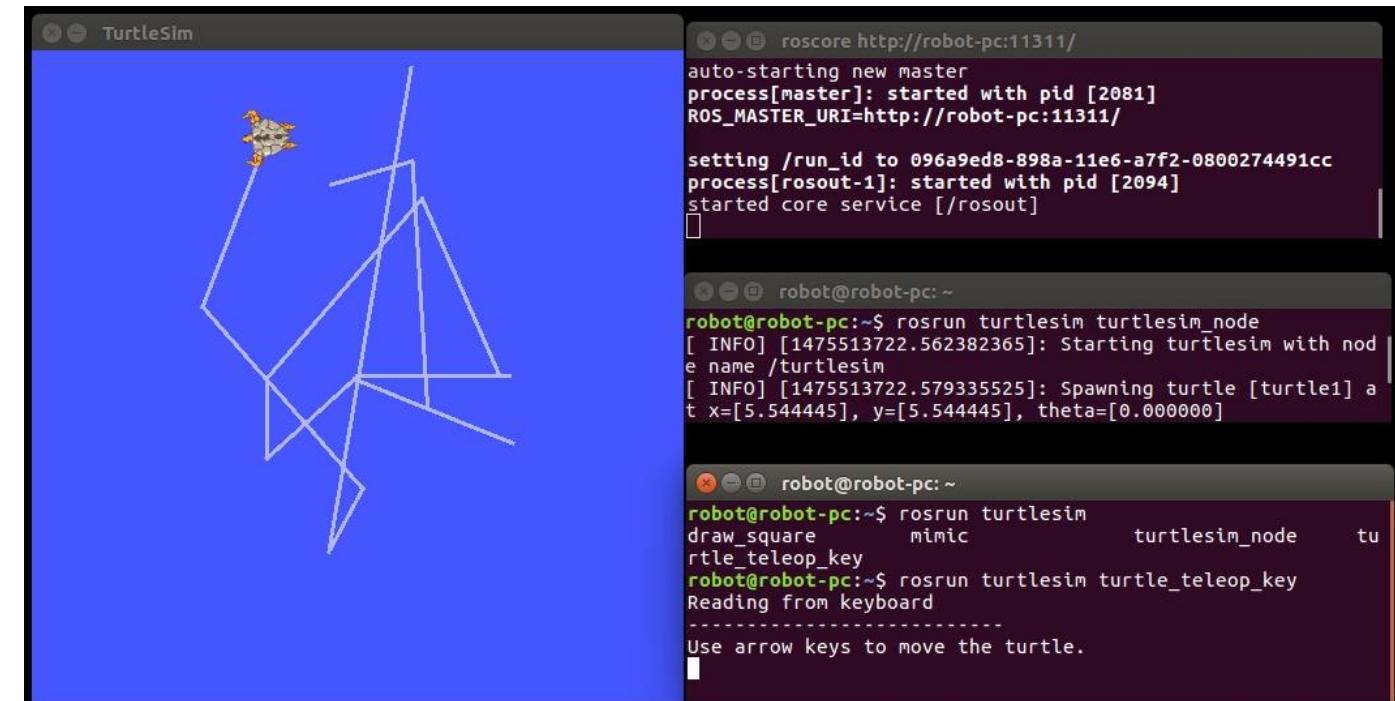
Test moving the turtle
(with the *turtle_teleop_key* node)

Recall: Open a terminal for each command

```
> roscore
```

```
> rosrun turtlesim turtlesim_node
```

```
> rosrun turtlesim turtle_teleop_key
```



The terminal which *turtle_teleop_key* is running on MUST be selected.
Change the turtle's position by pressing arrow keys on the keyboard.

ROS Command Tools

topic

List all active topics on ROS:

```
> rostopic list
```

```
luc@USMB:~$ rostopic list
/rosout
/rosout_agg
/turtle1/cmd_vel
/turtle1/color_sensor
/turtle1/pose
luc@USMB:~$
```

Display which message is used on a topic:

```
> rostopic type [topic_name]
```

```
luc@USMB:~$ rostopic type /turtle1/pose
turtlesim/Pose
```

Get more information on a topic:

```
> rostopic info [topic_name]
```

```
luc@USMB:~$ rostopic type /turtle1/pose
Type: turtlesim/Pose
```

Publishers:
* /turtlesim (<http://localhost:40351/>)

Subscribers: None

ROS Command Tools

node

List all active node running on ROS:

```
> rosnode list
```

```
luc@USMB:~$ rosnode list
/rosout
/teleop_turtle
/turtlesim
```

Display information including publication/subscription:

```
> rosnode info [node_name]
```

```
luc@USMB:~$ rosnode info turtlesim
-----
Node [/turtlesim]
Publications:
* /rosout [rosgraph_msgs/Log]
* /turtle1/color_sensor [turtlesim/Color]
* /turtle1/pose [turtlesim/Pose]

Subscriptions:
* /turtle1/cmd_vel [geometry_msgs/Twist]

Services:
* /clear
* /kill
* /reset
* /spawn
* /turtle1/set_pen
* /turtle1/teleport_absolute
```

ROS Command Tools

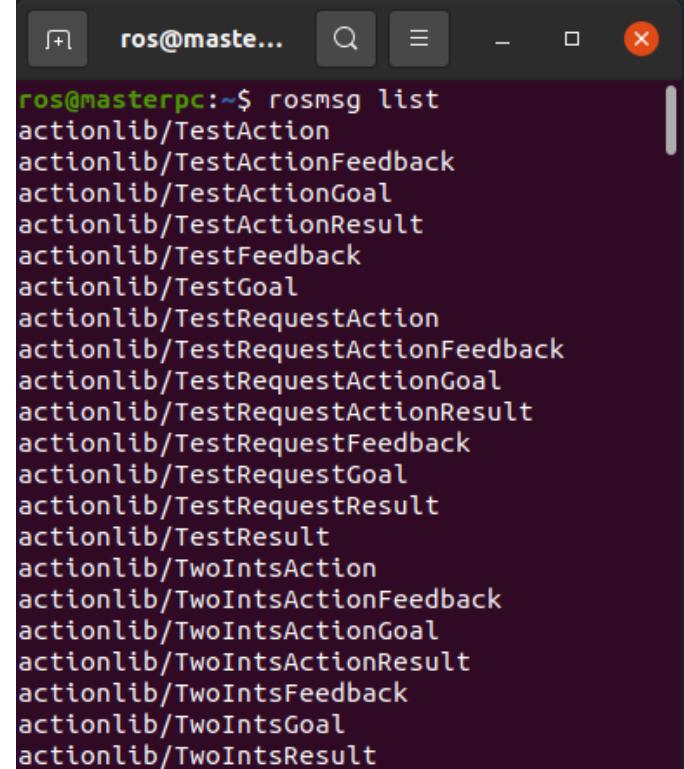
msg

Show all messages available in ROS:

```
> rosmsg list
```

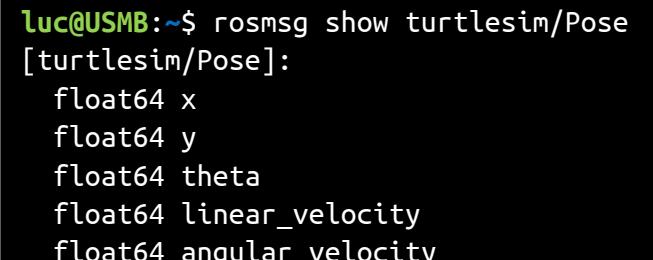
Show the content of a message type:

```
> rosmsg show [message_type]
```



```
ros@masterpc:~$ rosmsg list
actionlib/TestAction
actionlib/TestActionFeedback
actionlib/TestActionGoal
actionlib/TestActionResult
actionlib/TestFeedback
actionlib/TestGoal
actionlib/TestRequestAction
actionlib/TestRequestActionFeedback
actionlib/TestRequestActionGoal
actionlib/TestRequestActionResult
actionlib/TestRequestFeedback
actionlib/TestRequestGoal
actionlib/TestRequestResult
actionlib/TestResult
actionlib/TwoIntsAction
actionlib/TwoIntsActionFeedback
actionlib/TwoIntsActionGoal
actionlib/TwoIntsActionResult
actionlib/TwoIntsFeedback
actionlib/TwoIntsGoal
actionlib/TwoIntsResult
```

```
> rosmsg show turtlesim/Pose
```



```
luc@USMB:~$ rosmsg show turtlesim/Pose
[turtlesim/Pose]:
  float64 x
  float64 y
  float64 theta
  float64 linear_velocity
  float64 angular_velocity
```

ROS Command Tools

msg

See message definition information:

```
> rosmsg show [message_type]
```

```
> rosmsg show Pose
```

```
luc@USMB:~$ rosmsg show Pose
[turtlesim/Pose]:
float32 x
float32 y
float32 theta
float32 linear_velocity
float32 angular_velocity

[geometry_msgs/Pose]:
geometry_msgs/Point position
    float64 x
    float64 y
    float64 z
geometry_msgs/Quaternion orientation
    float64 x
    float64 y
    float64 z
    float64 w
```



The message of type *Pose* is defined in the package *turtlesim* but also in the package *geometry_msgs* but they are not the same !

ROS Command Tools

System File

Get information on packages

```
> rospack find [package_name]
```

Change directory (cd) directly to a package or a stack

```
> roscd [location_name[/subdir]]
```

/s directly in a package by name rather than by absolute path

```
> rosfs [location_name[/subdir]]
```

ROS CHEAT SHEET MELODIC

WORKSPACES	CMakeLists.txt	RUNNING SYSTEM
Create Workspace	Skeleton	Run ROS using plain: roscore
<pre>mkdir catkin_ws && cd catkin_ws wstool init src catkin_make source devel/setup.bash</pre>	<pre>cmake_minimum_required(VERSION 2.8.3) project(package_name) find_package(catkin REQUIRED) catkin_package()</pre>	Alternatively, rosrun will run its own roscore automatically if it can't find one: rosrun my_package package.launchfile.launch Suppress this behaviour with the --wait flag.
Add Repo to Workspace	Package Dependencies	Nodes, Topics, Messages
<pre>rosscd; cd ../src wstool set repo_name \ -git http://github.com/org/repo_name.git \ --version=melodic-devel wstool up</pre>	To use headers or libraries in a package, or to use a package's exported CMake macros, express a build-time dependency: <pre>find_package(catkin REQUIRED COMPONENTS roscpp)</pre>	<pre>rossnode list rostopic list rostopic echo cmd_vel rostopic hz cmd_vel rostopic info cmd_vel rossmg show geometry_msgs/Twist</pre>
Resolve Dependencies in Workspace	Packages	Remote Connection
<pre>sudo rosdep init # only once rosdep update rosdep install --from-paths src --ignore-src \ --rosdistro=\${ROS_DISTRO} -y</pre>	Create a Package	Master's ROS environment: <ul style="list-style-type: none">ROS_IP or ROS_HOSTNAME set to this machine's network address.ROS_MASTER_URI set to the URI containing that IP or hostname.
	Package Folders	Your environment: <ul style="list-style-type: none">ROS_IP or ROS_HOSTNAME set to your machine's network address.ROS_MASTER_URI set to the URI from the master.
	<pre>catkin_create_pkgs package_name [dependencies ...]</pre>	To debug, check pinging from each side to the other, run rosif on each side.
	Messages, Services	ROS Console
	<pre>include/package_name C++ header files src Source files. Python libraries in subdirectories scripts Python nodes and scripts msg, srv, action Message, Service, and Action definitions</pre>	Adjusting using rqt_logger_level and monitor via rqt_console. To enable debug output across sessions, edit the \$HOME/.ros/config/rosconsole.config and add a line for your package: <pre>log4j.logger.ros.package_name=DEBUG</pre>
	Build Libraries, Executables	And then add the following to your session: <pre>export ROSCONSOLE_CONFIG_FILE=\$HOME/.ros/config/rosconsole.config</pre>
	<pre>catkin_generate_changelog # review & commit changelogs catkin_prepare_release bloom-release --track melodic --ros-distro melodic repo_name</pre>	Use the rosrun --screen flag to force all node output to the screen, as if each declared <node> had the output="screen" attribute.
	Reminders	
	<ul style="list-style-type: none">Testable logicPublish diagnosticsDesktop dependencies in a separate package	
	Installation	
	<pre>install(TARGETS \${PROJECT_NAME} DESTINATION \${CATKIN_PACKAGE_LIB_DESTINATION}) install(TARGETS \${PROJECT_NAME}_node DESTINATION \${CATKIN_PACKAGE_BIN_DESTINATION}) install(PROGRAMS scripts/myscript DESTINATION \${CATKIN_PACKAGE_BIN_DESTINATION}) install(DIRECTORY launch DESTINATION \${CATKIN_PACKAGE_SHARE_DESTINATION})</pre>	



www.clearpathrobotics.com/ros-cheat-sheet

© 2019 Clearpath Robotics, Inc. All Rights Reserved.

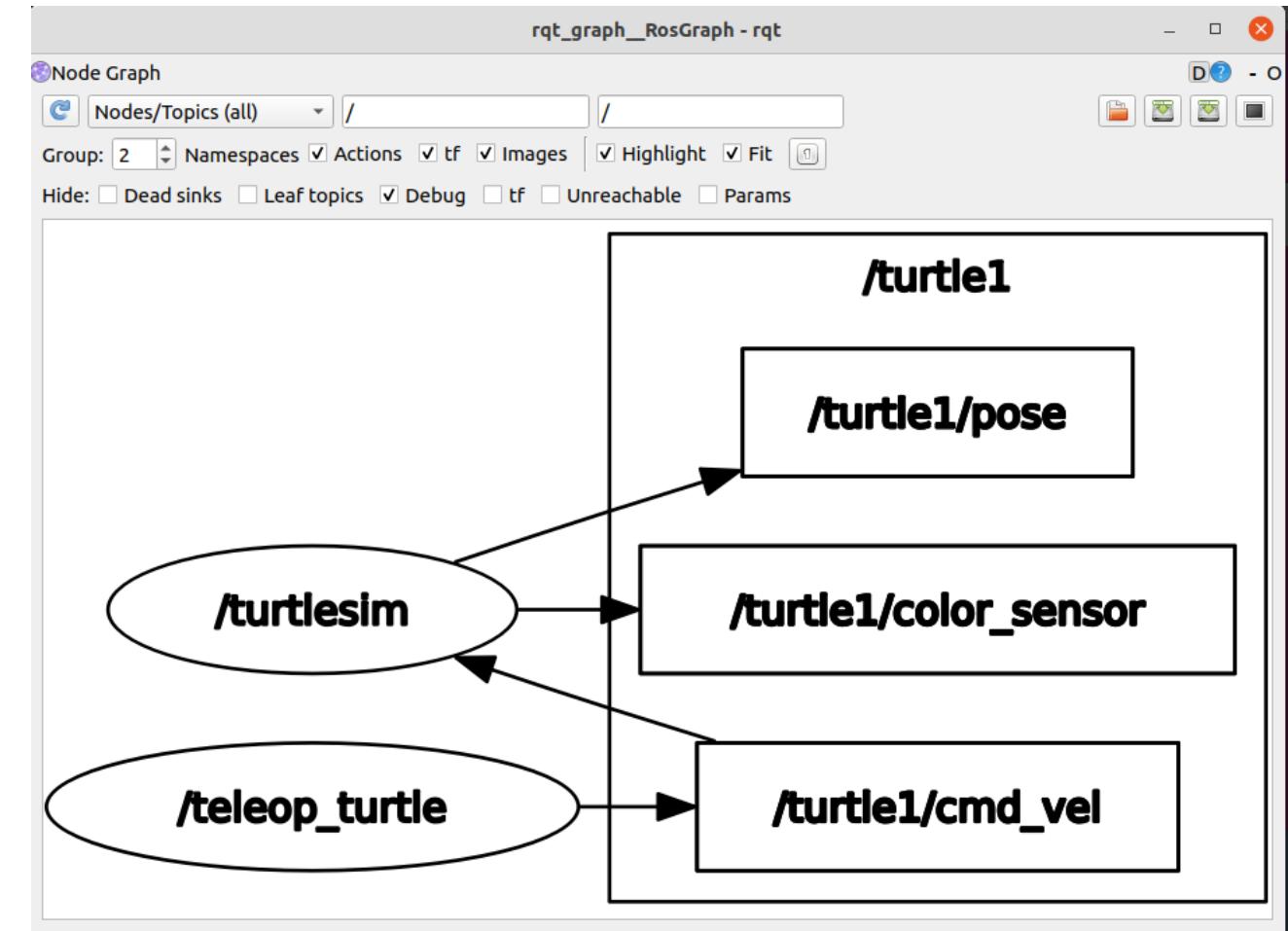
More info

<http://wiki.ros.org/ROS/Tutorials/NavigatingTheFilesystem>

ROS computation graph *rqt*

Visualize running topics and nodes

```
> rosrun rqt_graph rqt_graph
```

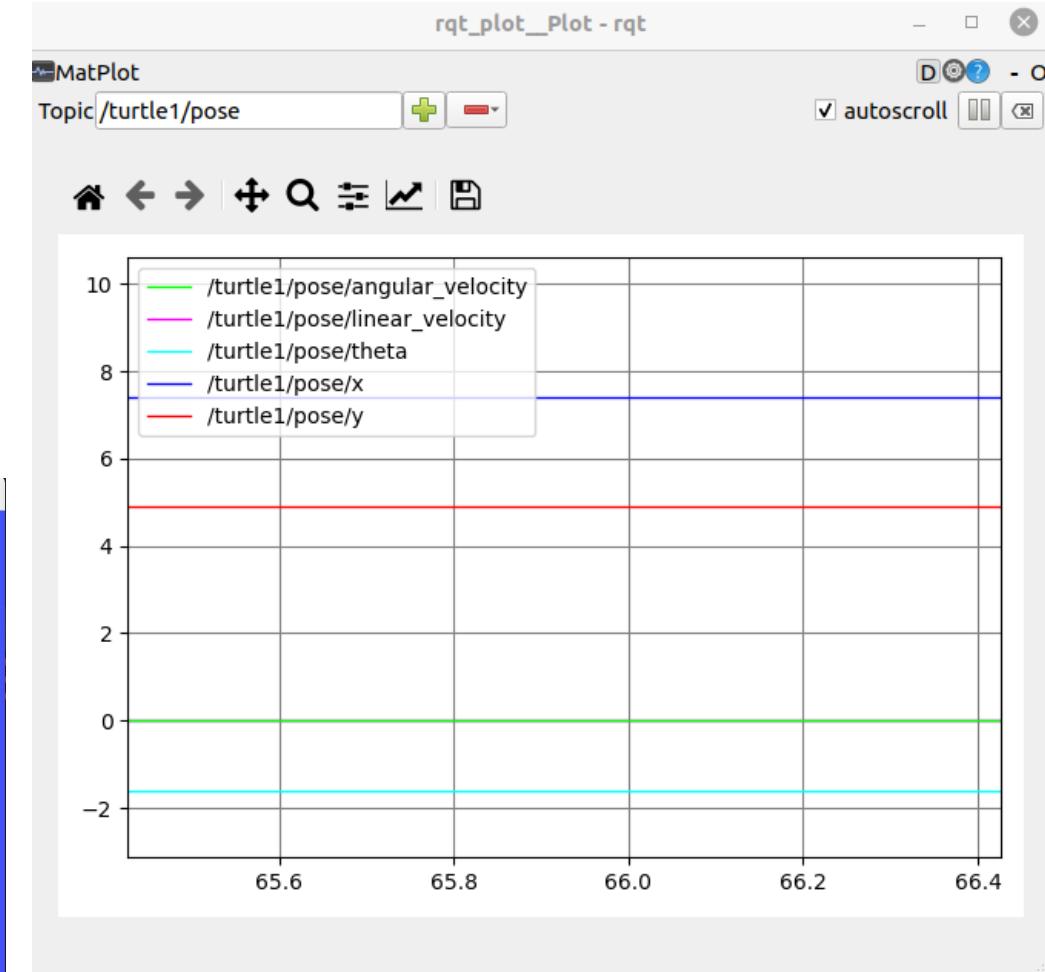
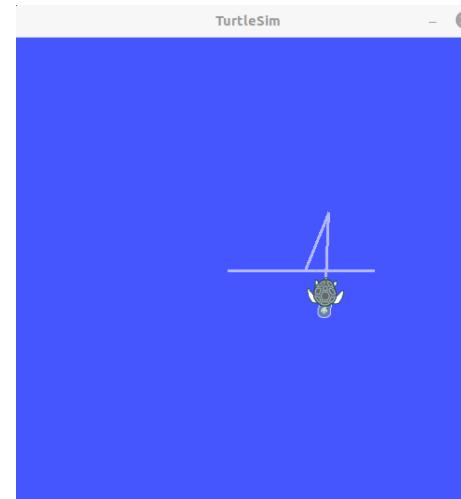


ROS computation graph *rqt*

Visualize running topics and nodes

```
> rosrun rqt_plot rqt_plot
```

It shows the values published on a topic



ROS computation graph *rqt*

- *rqt_graph* creates a dynamic graph of what's going on in the system
- *rqt_console* attaches to ROS's logging framework to display output from nodes. *rqt_logger_level* allows us to change the verbosity level (DEBUG, WARN, INFO, and ERROR) of nodes as they run.
- Prerequisite: Install *rqt* package

```
> sudo apt-get install ros-noetic-rqt ros-noetic-rqt-common-plugins
```

Launch *rqt_console*

```
> rosrun rqt_console rqt_console
```

Launch *rosrun rqt_logger_level rqt_logger_level* (in another terminal)

```
> rosrun rqt_logger_level rqt_logger_level
```

ROS Launch files

ROS Launch

- Imagine a scenario in which we have to launch 10 or 20 nodes for a robot.
 - It will be difficult if we run each node in a terminal one by one !!!

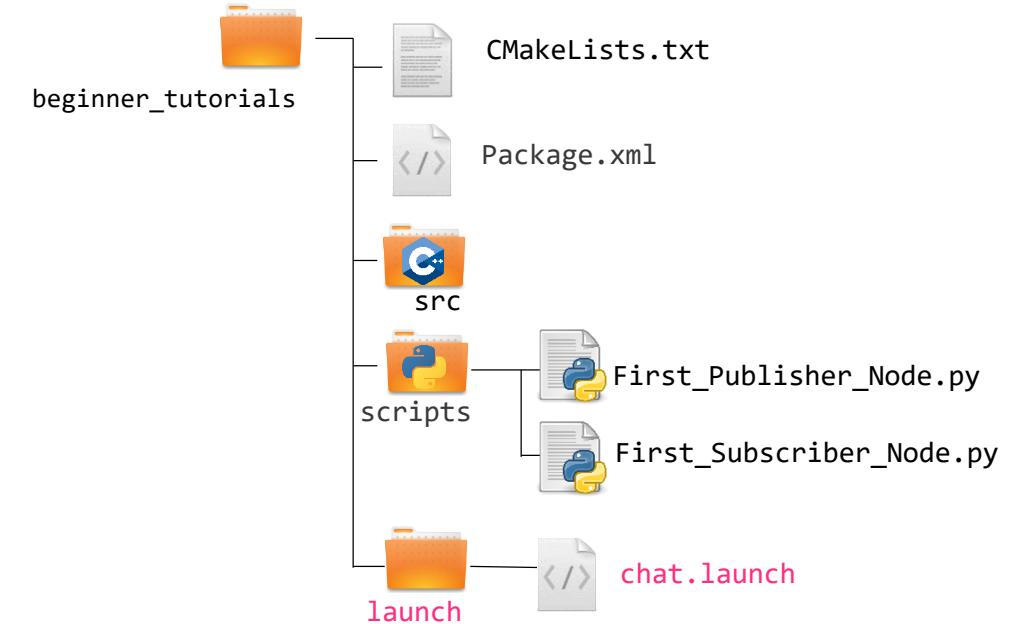
```
terminal one by one !!!
```

The screenshot shows a terminal window titled "luc@USMB: ~". The log output is as follows:

```
[ INFO] [1552050671.521743102]: Starting turtlesim with node name /turtlesim
[ INFO] [1552050671.528928464]: Spawning turtle [turtle1] at x=[5,544445], y=[544445], theta=[0,000000]
[ INFO] [1552050671.544445], theta=[0,000000]
[ INFO] [1552050671.544445], theta=[0,000000]
```

ROS Launch

- *launch* is a tool for launching multiple nodes (as well as setting parameters)
- written in XML but file suffix: `*.launch`
- the launch file needs to be located in a folder named “`launch`” inside de package folder
- If not yet running, launch automatically starts a `roscore`



Example

The file `chat.launch` is created in order to launch the node :
`First_Publisher_Node.py` and `First_Subscriber_Node.py`

More info

<http://wiki.ros.org/rosLaunch>

ROS Launch

Start a launch file from a package with

```
> roslaunch [package_name] [file_name.launch]
```

Or browse to the folder and start a launch file with

```
> roslaunch [file_name.launch]
```

Example console output for:

```
> roslaunch beginner_tutorials chat.launch
```

```
/home/luc/catkin_ws/src/beginner_tutorials/launch/chat.launch http://localhost:11311
File Edit View Search Terminal Help
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://USMB:33599/

SUMMARY
=====

PARAMETERS
  * /rosdistro: melodic
  * /rosversion: 1.14.3

NODES
/
  First_Publisher_Node (beginner_tutorials/First_Publisher_Node.py)
  First_Subscriber_Node (beginner_tutorials/First_Subscriber_Node.py)

auto-starting new master
process[master]: started with pid [3021]
ROS_MASTER_URI=http://localhost:11311

setting /run_id to df062496-6923-11ea-ac4b-0800270a6f6f
process[rosout-1]: started with pid [3032]
started core service [/rosout]
process[First_Publisher_Node-2]: started with pid [3038]
process[First_Subscriber_Node-3]: started with pid [3040]
[INFO] [1584541338.862159]: /First_Subscriber_NodeI heard 1
```

More info

<http://wiki.ros.org/roslaunch>

ROS Launch

Other example

Turtle.Launch

```
<launch>
  <node name="turtlesim_node" pkg="turtlesim" type="turtlesim_node"/>
  <node name="turtle_teleop_key" pkg=" turtlesim " type="turtle_teleop_key"/>
</launch>
```

- **launch**: root element of the Launch files. This is an XML document, and every XML document has one
- **node**: each `<node>` tag specifies a node to be launched
- **name**: name of the node (free to choose)
- **pkg**: package containing the node
- **type**: the executable name (if the executable is a python file, don't forget the `.py` extention)
- **output**: specifies where to output log messages (screen -> consol, log -> log file)
`output="screen"` makes the ROS log messages appear on the launch terminal window

ROS Launch

Other example

Turtle.Launch

```
<launch>
  <node name="turtlesim_node" pkg="turtlesim" type="turtlesim_node"/>
  <node name="turtlesim_target_node" pkg="beginner_tutorials" type="turtlesim_target_node.py" output="screen"/>
</launch>
```

turtlesim_node is NOT a python script

turtlesim_target_node.py IS a python script
The .py extention is needed

- **launch**: root element of the Launch files. This is an XML document, and every XML document has one
- **node**: each `<node>` tag specifies a node to be launched
- **name**: name of the node (free to choose)
- **pkg**: package containing the node
- **type**: the executable name (if the executable is a python file, don't forget the `.py` extention)
- **output**: specifies where to output log messages (screen -> consol, log -> log file)
`output="screen"` makes the ROS log messages appear on the launch terminal window

ROS Launch

File Structure

chat.Launch

```
<launch>
  <node name="First_Publisher_Node" pkg="beginner_tutorials" type="First_Publisher_Node.py"/>
  <node name="First_Subscriber_Node" pkg="beginner_tutorials" type="First_Subscriber_Node.py" output="screen"/>
</launch>
```

if you don't put the option `output="screen"` you will not be able to see messages on the Terminal



- **launch**: root element of the Launch files. This is an XML document, and every XML document has one
- **node**: each `<node>` tag specifies a node to be launched
- **name**: name of the node (free to choose)
- **pkg**: package containing the node
- **type**: the executable name (if the executable is a python file, don't forget the `.py` extention)
- **output**: specifies where to output log messages (screen -> consol, log -> log file)
`output="screen"` makes the ROS log messages appear on the launch terminal window

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>
```

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>
```

ROS Launch

Important facts

- `rosrun` can only launch one node at a time, from a single package WHILE `roslaunch` can launch two or more nodes at the same time, from multiple packages
- `roslaunch` will automatically start `roscore` (if not running already) WHILE `rosrun` does not.
- `roslaunch` uses launch files, which can automatically start other programs by including other launch files, set some parameters, etc WHILE `rosrun` does not use launch files - it launches the nodes directly.