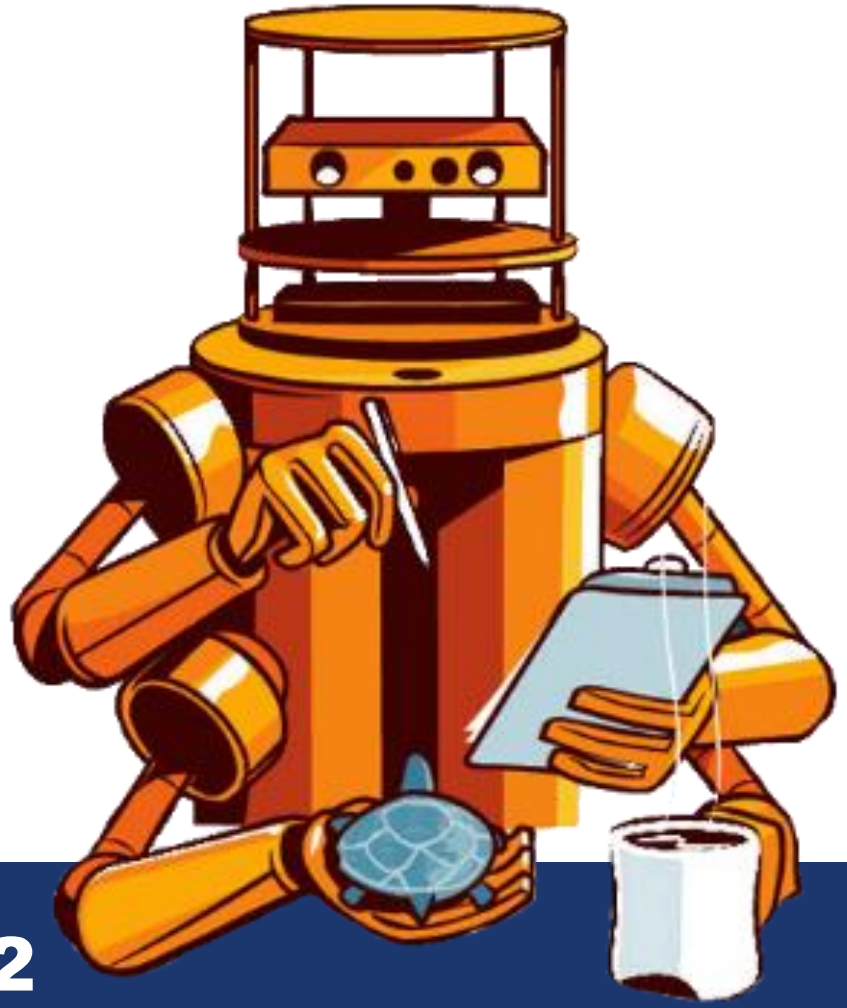


Lecture 2

2021

INFO 802

Master Advanced Mechatronics

Luc Marechal

UNIVERSITÉ SAVOIE MONT BLANC

POLYTECH ANNECY-CHAMBÉRY

ROS

Publisher, Subscriber Node
ROS launch

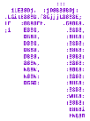
Objectives

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

- ☑ Create a custom launch file.
- ☑ Code a Publisher node and use the `rospy.Publisher function`
- ☑ Code a Subscriber node and use the `rospy.Subscriber function`
- ☑ Know what is a `callback` function and how it works.
- ☑ Achieve at least grade 80% of the Assignement

Script editor in Ubuntu

There are many options to edit script in Ubuntu :



- Nano is a Command Line editor → Not user friendly for Python coding

```
> sudo nano <filename>
```

Gedit is the official default text editor of Ubuntu → A bit basic

```
> sudo gedit <filename>
```

- **Sublime Text3** is a halfway IDE text editor with auto-completion of basic functions → Nice !

```
> sudo subl <filename>
```

- **Visual Studio Code** → Super Nice !

```
> sudo code <filename>
```

IDE for ROS

There is no best IDEs, only the IDE that works best for you !

Eclipse, **Net Beans**, **Qt Creator**: popular on Ubuntu ( )
Anaconda: nice interface ( )
but the ROS environment has to be set up and can be tedious

ROS Development Studio: only online ( )

Clion: user friendly and easy to setup ( )

RoboWare Studio: IDE especially designed for working with ROS. The installation is quite easy, and automatically detects and loads an ROS environment without additional configurations. It has different out-of-the-box features ( )

Create first node *Hello World (Python)*

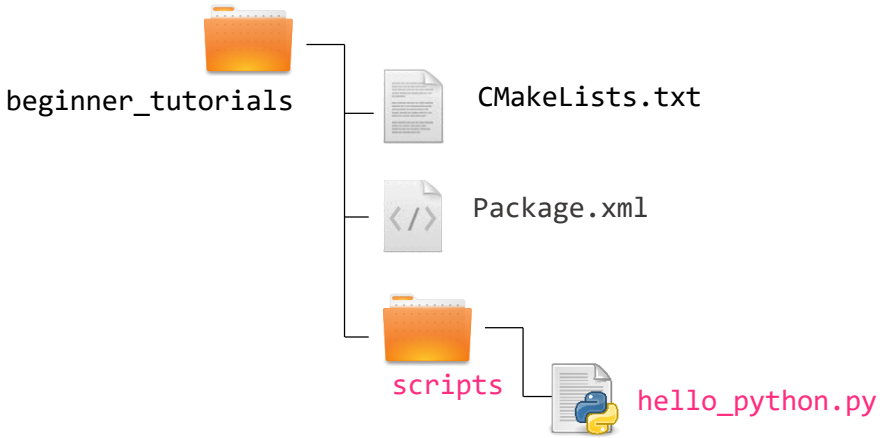
with **rospy** (Python Client Library)

```
#!/usr/bin/env python3
# -*- coding utf-8 -*-

__author__ = "Luc Marechal"
__copyright__ = "The Hello World Project copyright"
__credits__ = "myself"
__license__ = "GPL"
__version__ = "0.0.1"
__maintainer__ = "Luc Marechal"
__email__ = "luc@univ-smb.fr"
__status__ = "Development"

import rospy
rospy.init_node('hello_python')

rate = rospy.Rate(10)

while not rospy.is_shutdown():
  print("Hello World")
  rate.sleep()
```

This is the *shebang*. It lets the OS know that this is a Python file, and that it should be passed to the Python interpreter

Create the node

```
> ~~mkdir ~/catkin_ws/src/beginner_tutorials/scripts~~
> cd ~/catkin_ws/src/beginner_tutorials/scripts
> sudo subl hello_python.py
```



Building first node *Hello World (Python)*

with **rospy** (Python Client Library)

Make the file executable

```
> sudo chmod +x hello_python.py
```

→ Give execution permissions to the file

Build package

```
> cd ~/catkin_ws
> catkin_make beginner_tutorials
```

Make sure you have sourced your workspace's setup.bash file

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

Run your node

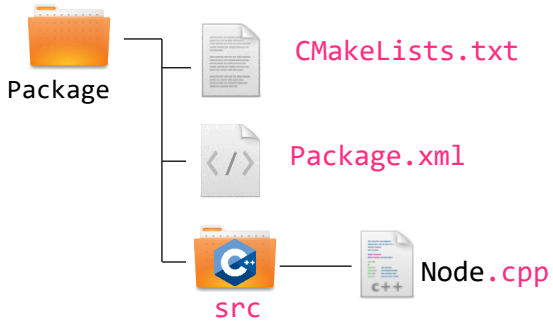
```
> rosrun beginner_tutorials hello_python.py
```

→ Extension needed

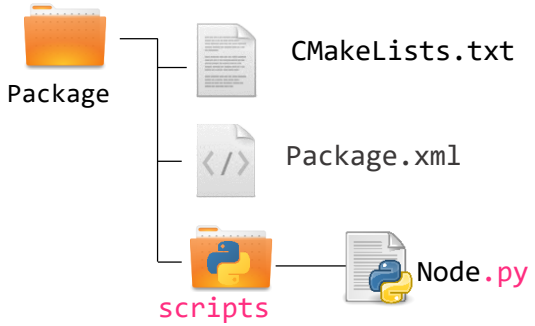
http://www2.ece.ohio-state.edu/~zhang/RoboticsClass/docs/ECE5463_ROSTutorialLecture1.pdf

Create Nodes Summary

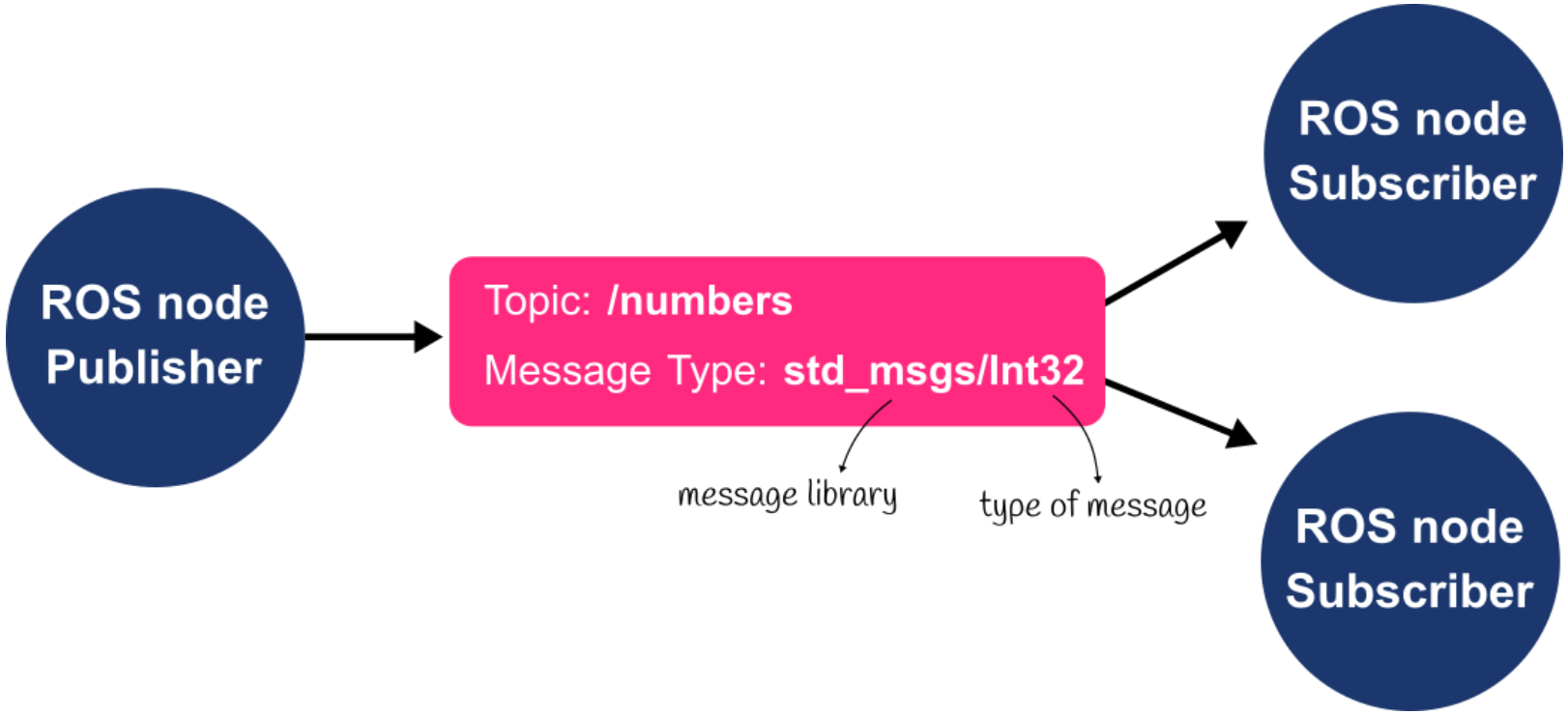
1) Create your **.cpp* file in /src folder of the package
2) Customize CMakeLists.txt and package.xml files
3) Build the package which contains the node
4) Source your workspace
5) Run your node



1) Create your **.py* file in /scripts folder of the package
2) Make the file executable
3) Source your workspace
4) Run your node with the *.py* extention



Creating a Publisher and a Subscriber Node (Python)



The publisher node publishes a **message** of type *Int32* on the **topic** named *numbers*

The subscriber node subscribes to the topic named numbers on which the message is of type Int32

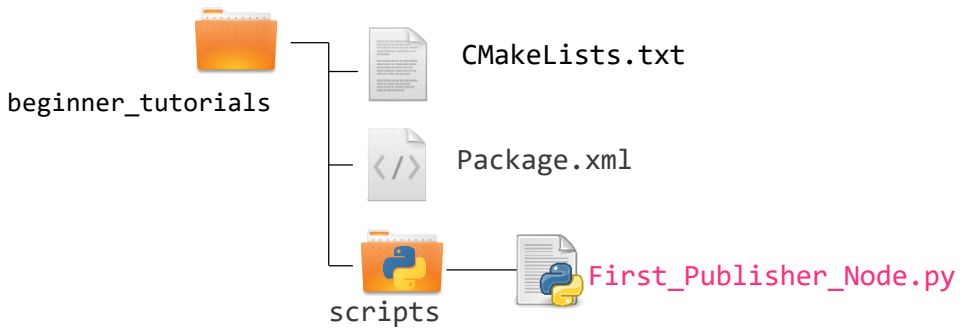
Creating a Publisher and a Subscriber Node (Python)

Writing the publisher Node

- This node will publish an integer value on a topic called *numbers*

Edit a .py file in scripts folder

```
> cd ~/catkin_ws/beginner_tutorials/
> mkdir scripts
> cd scripts
> sudo subl First_Publisher_Node.py
```



First_Publisher_Node.py

```
#!/usr/bin/env python3
import rospy
from std_msgs.msg import Int32


def First_Publisher_Node():
    pub = rospy.Publisher('numbers', Int32, queue_size=10)

    rospy.init_node('First_Publisher_Node', anonymous=True)

    rate = rospy.Rate(10) # 10hz

    number_count=0
    while not rospy.is_shutdown():
        rospy.loginfo(number_count)
        pub.publish(number_count)
        rate.sleep()
        number_count += 1

if __name__ == '__main__':
    First_Publisher_Node()
```

Creating a Publisher and a Subscriber Node (Python)

Examining the publisher Node

Every Python ROS Node will have this declaration at the top.

You need to import rospy if you are writing a ROS Node.
std_msgs.msg import is so that we can reuse the std_msgs/Int32 message type

The node is publishing to the numbers topic using the message type Int32
The queue_size argument limits the amount of queued messages if any subscriber is not receiving them fast enough.

anonymous = True ensures that your node has a unique name by adding random numbers to the end of NAME.

Helper class to run loop at desired frequency (here 10 Hz)

First_Publisher_Node.py

```
#!/usr/bin/env python3
import rospy
from std_msgs.msg import Int32


def First_Publisher_Node():
    pub = rospy.Publisher('numbers', Int32, queue_size=10)

    rospy.init_node('First_Publisher_Node', anonymous=True)

    rate = rospy.Rate(10) # 10hz

    number_count=0
    while not rospy.is_shutdown():
        rospy.loginfo(number_count)
        pub.publish(number_count)
        rate.sleep()
        number_count += 1

if __name__ == '__main__':
    First_Publisher_Node()
```

Creating a Publisher and a Subscriber Node (Python)

Examining the publisher Node

```
pub = rospy.Publisher(name of the topic, message_type, queue size)
```

pub is an Object

queue size: this is the size of the outgoing message queue used for **asynchronous** publishing

```
pub.publish(message)
```

publish() is a method of the pub Object
It publishes the message on the ROS network at the topic location

More info
http://wiki.ros.org/rospy/Overview/Publishers%20and%20Subscribers

Creating a Publisher and a Subscriber Node (Python)

Examining the publisher Node

`rospy.loginfo`

```
rospy.loginfo("my message")
```

This is a help for you. It prints anything you want in the Terminal.

Here we use it to print in the Terminal the message that is published on the topic

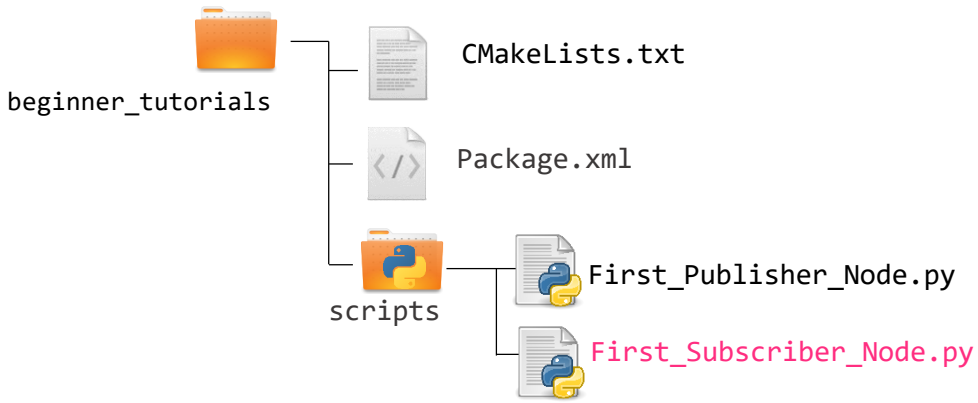
Creating a Publisher and a Subscriber Node (Python)

Writing the subscriber Node

- This node will subscribe to an integer value on a topic called *numbers*

Edit a .py file in scripts folder

```
> cd ~/catkin_ws/beginner_tutorials/scripts
> sudo subl First_Subscriber_Node.py
```



First_Subscriber_Node.py

```python
#!/usr/bin/env python3
import rospy
from std_msgs.msg import Int32

def callback(msg):
    rospy.loginfo("I heard %s", msg.data)

def First_Subscriber_Node():
  # In ROS, nodes are uniquely named. If two nodes with the same name are
  # launched, the previous one is kicked off. The anonymous=True flag means that
  # rospy will choose a unique name for our 'listener' node so that multiple
  # listeners can run simultaneously.

    rospy.init_node('First_Subscriber_Node', anonymous=True)

    rospy.Subscriber('numbers', Int32, callback)

    rospy.spin()

if __name__ == '__main__':
    First_Subscriber_Node()
```

Creating a Publisher and a Subscriber Node (Python)

Examining the subscriber Node

First_Subscriber_Node.py

```python
#!/usr/bin/env python3
import rospy
from std_msgs.msg import Int32

def callback(msg):
    rospy.loginfo("I heard %s", msg.data)

def First_Subscriber_Node():
   # In ROS, nodes are uniquely named. If two nodes with the same name are launched, the
   # previous one is kicked off. The anonymous=True flag means that rospy will choose a
   # unique name for our 'listener' node so that multiple listeners can run simultaneously.

    rospy.init_node('First_Subscriber_Node', anonymous=True)

    rospy.Subscriber('numbers', Int32, callback)

    rospy.spin()

if __name__ == '__main__':
    First_Subscriber_Node()
```

`rospy.loginfo`: logs messages to the filesystem

The `anonymous=True` flag tells rospy to generate a unique name for the node so that you can have multiple listener.py nodes run easily

When new messages are received, callback* is invoked with the message as the first argument.

`rospy.spin()`: simply keeps the node from exiting until the node has been shutdown

*`Callback` = function that is passed as an argument to other function

Creating a Publisher and a Subscriber Node (Python)

Examining the subscriber Node

```
rospy.Subscriber(name of the topic, message_type, callback_function)
```

The callback function can be seen as a message handler
It contains the message read on the topic as its first argument.
This why in its definition the argument is the message

```
def callback_function(message):
```

Example
If the message is a *std_msgs/Int32*

```
rospy.Subscriber('my_topic', Int32, callback)
```

```
def callback(msg):
  value_read = msg.data
  ...
```

structure of Int32 message type

```
luc@USMB:~$ rosmsg show Int32
[std_msgs/Int32]:
int32 data
```

Creating a Publisher and a Subscriber Node (Python)

Examining the subscriber Node

`rospy.loginfo`

```
rospy.loginfo("I heard %s", msg.data)
```

Here we use it to printout in the Terminal the message that we read on the topic

```
luc@USMB:~$ rosmsg show Int32
[std_msgs/Int32]:
int32 data
```

In our exemple, to access the Int32 message value, we need to use: msg.data
(because this is how the message Int32 in constructed)

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

If we wanted to access the theta value of a Pose message, we need would need: msg.theta

Creating a Publisher and a Subscriber Node (Python)

Building the nodes

Make the node executable (for Python only)

```
> sudo chmod +x First_Subscriber_Node.py
> sudo chmod +x First_Publisher_Node.py
```

Build package
(we use Cmake as the build system even for Python nodes)

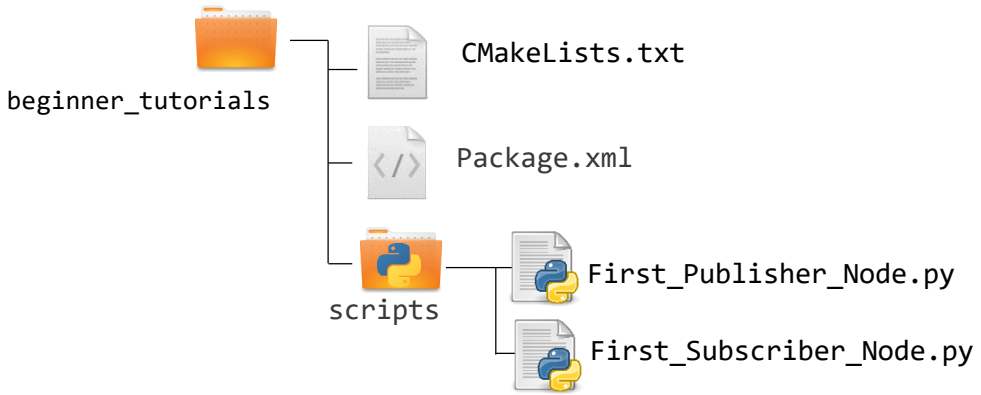
```
> cd ~/catkin_ws
> catkin_make
```

Make sure you have sourced your workspace's setup.bash file

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

Run your nodes

```
> rosrun beginner_tutorials First_Publisher_Node.py
> rosrun beginner_tutorials First_Subscriber_Node.py
```



Creating a Publisher and a Subscriber Node (Python)

Recall : basic structure of a Node

Basic structure of a subscriber node

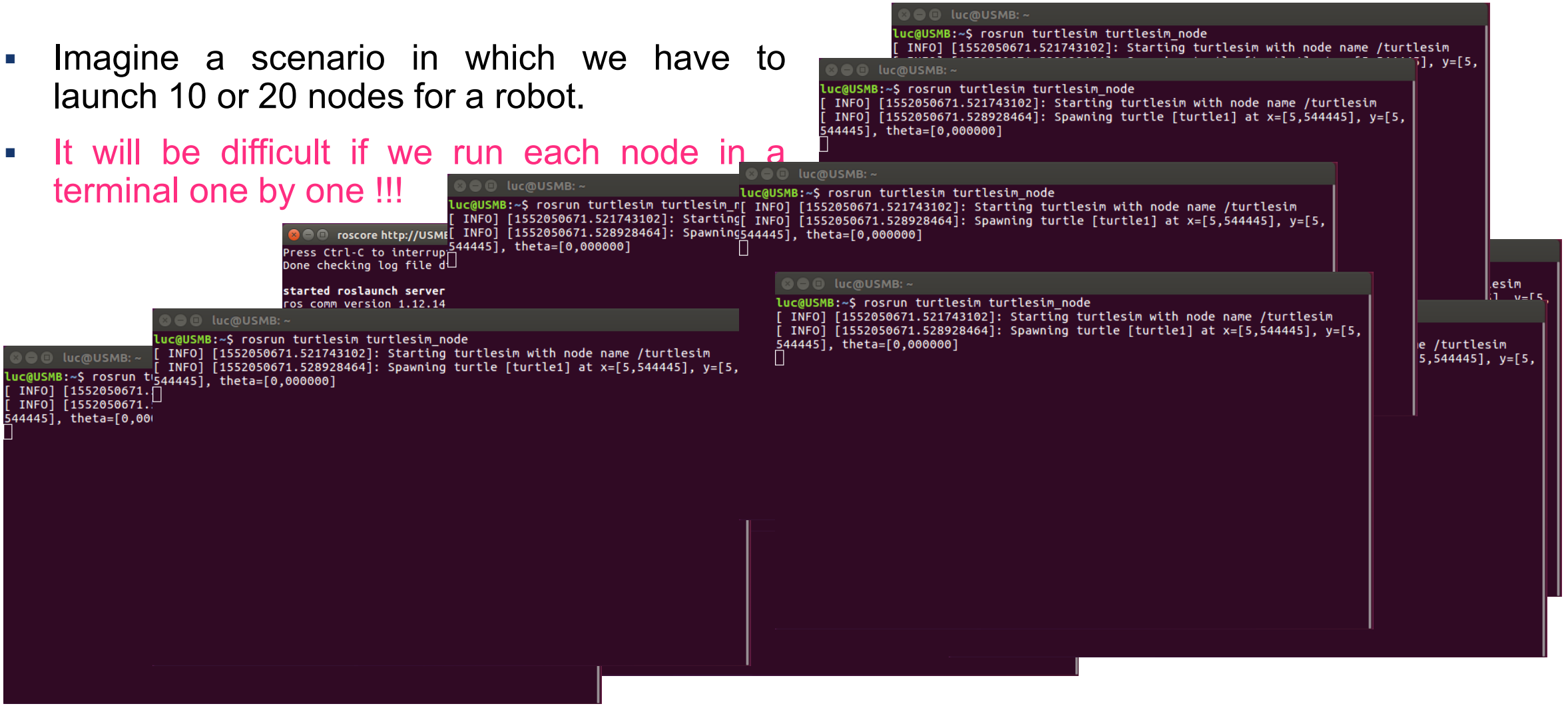
```
#!/usr/bin/env python3

Import ####
from #### import ####

def callback(msg):
    #####

def The_Node():
    rospy.init_node('The_Node', anonymous=True)

    rospy.Subscriber(topic, message_type, callback)

    rospy.spin()


if __name__ == '__main__':
    The_Node()
```

Basic structure of a publisher node

```
#!/usr/bin/env python3

Import ####
from #### import ####

def The_Node():
    rospy.init_node('The_Node', anonymous=True)

    pub = rospy.Publisher(topic, message_type, queue_size=##)

    rate = rospy.Rate(##)

    while not rospy.is_shutdown():
        pub.publish(###)
        rate.sleep()

if __name__ == '__main__':
    The_Node()
```

ROS Launch

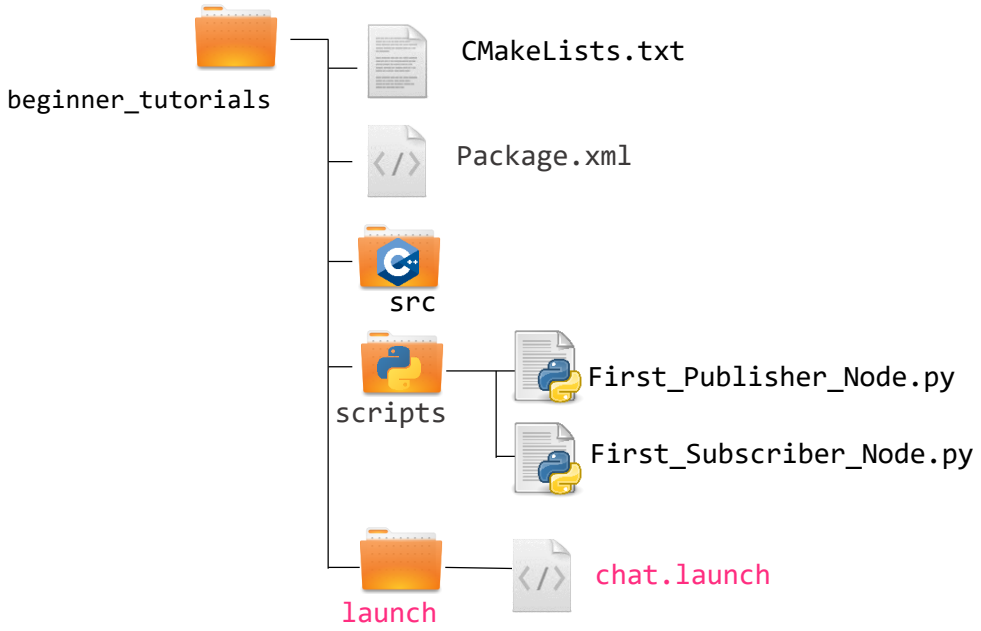
- Imagine a scenario in which we have to launch 10 or 20 nodes for a robot.
- It will be difficult if we run each node in a terminal one by one !!!



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

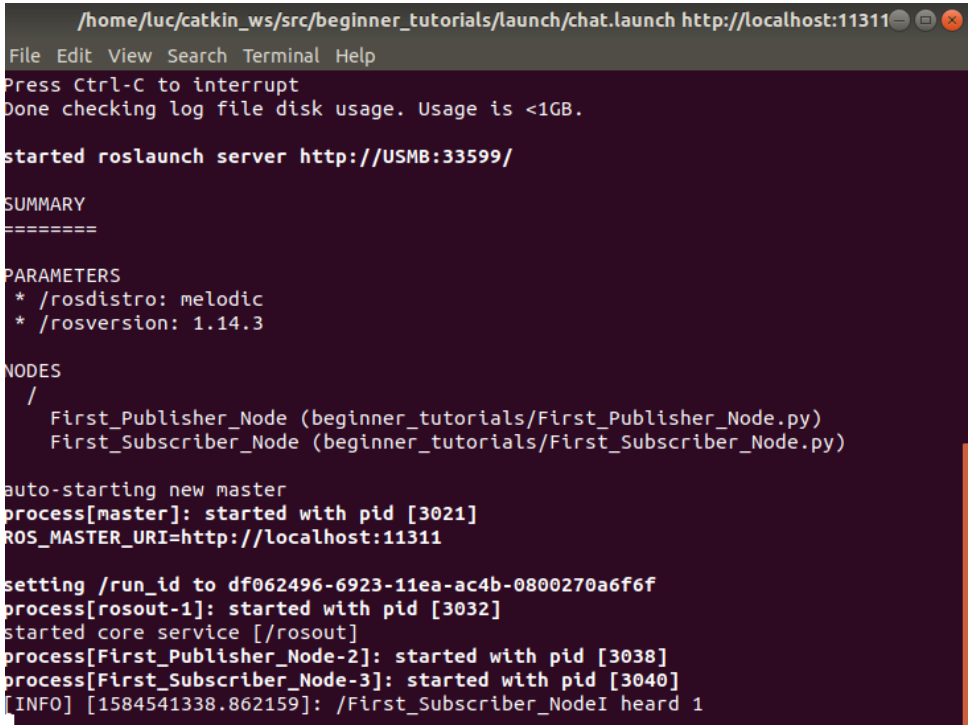
Start a launch file from a package with

```
> 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`



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" output="screen"/>
</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

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>
```

- **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

turtlesim_node is NOT a python script

```
<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>
```

- **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>
```

Important Facts

Steps to create a node in pythton :

1. Create your **.py* file in /scripts folder of the package
2. Make the file executable with: `sudo chomd +x nodefile.py`
3. Source your workspace with: `source ~/catkin_ws/devel/setup.bash`
4. Run your node with the *.py* extention: `rosrun package_name nodefile.py`

`rospy.Publisher(`*`name of the topic, message_type, queue size`*`)` and publish() fonctions are used in the publisher node

`rospy.Subscriber(`*`name of the topic, message_type, callback_function`*`)` and `callback functions are used in` the publisher node

A `callback` is function that is passed as an argument to an other function

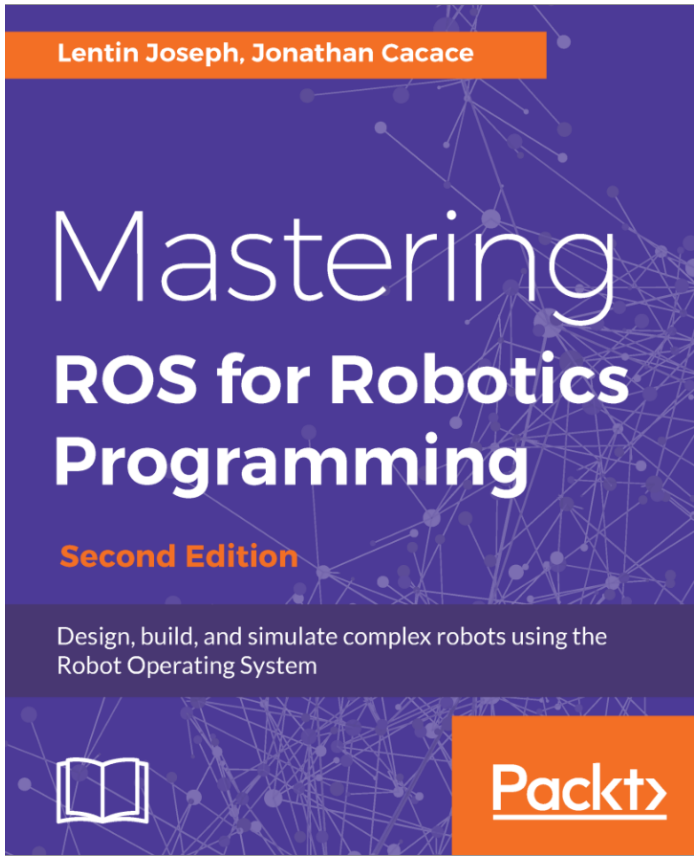
In the function `rospy.Subscriber`, the callback automatically contains the message read on the topic as its argument

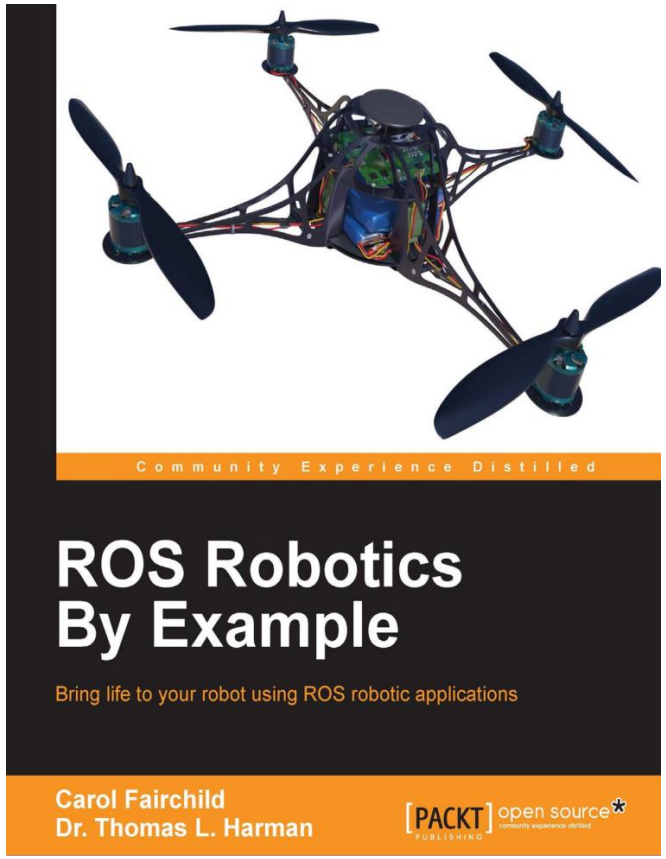
`rospy.loginfo("message") is a useful function to printout messages and variables in the Terminal`

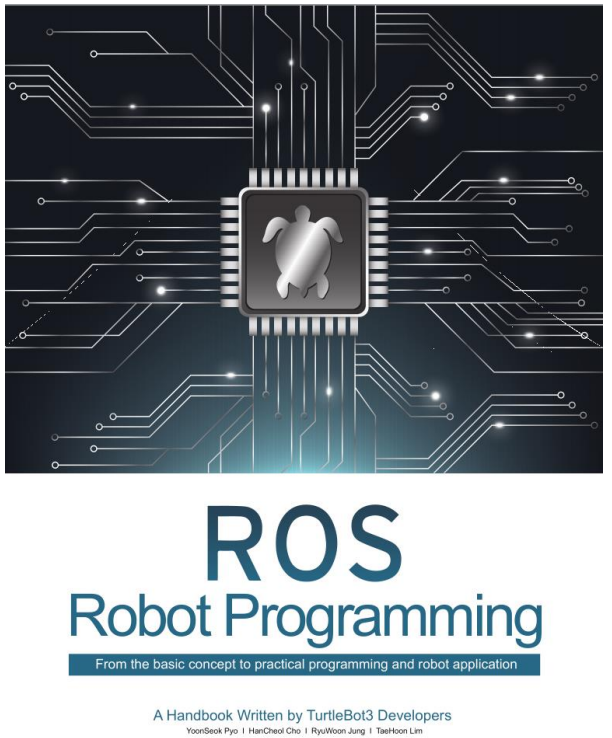
Further References

- 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://www.clearpathrobotics.com/ros-robot-operating-system-cheat-sheet/
  - https://kapeli.com/cheat_sheets/ROS.docset/
- ROS Best Practices
  - https://github.com/leggedrobotics/ros_best_practices/wiki
- ROS Package Template
  - https://github.com/leggedrobotics/ros_best_practices/tree/master/ros_package_template

Relevant books







Contact Information

Université Savoie Mont Blanc

Polytech' Annecy Chambery
Chemin de Bellevue
74940 Annecy
France

https://www.polytech.univ-savoie.fr

Lecturer

Luc Marechal (luc.marechal@univ-smb.fr)

SYMME Lab (Systems and Materials for Mechatronics)

UNIVERSITÉ SAVOIE MONT BLANC
POLYTECH ANNECY-CHAMBÉRY