INFO 802

Master Advanced Mechatronics

Luc Marechal

UNIVERSITÉ SAVOIE MONT BLANC

POLYTECH ANNECY-CHAMBÉRY

2019

ROS

Introduction

Course 2

Create first node *Hello World*

with **roscpp** (C++ Client Library)

```cpp
#include <ros/ros.h>

int main(int argc, char** argv)
{
  ros::init(argc, argv, "hello_world");
  ros::NodeHandle nodeHandle;
  ros::Rate loopRate(10);

  unsigned int count = 0;
  while (ros::ok()) {
    ROS_INFO_STREAM("Hello World " << count);
    ros::spinOnce();
    loopRate.sleep();
    count++;
  }

  return 0;
}
```

- ROS main header file include
- `ros::init(…)` has to be called before calling other ROS functions
- The node handle is the access point for communications with the ROS system (topics, services, parameters)
- `ros::Rate` is a helper class to run loops at a desired frequency
- `ros::ok()` checks if a node should continue running
  Returns `false` if `SIGINT` is received (`Ctrl` + `C`) or `ros::shutdown()` has been called
- `ROS_INFO()` logs messages to the filesystem
- `ros::spinOnce()` processes incoming messages via callbacks

More info

http://wiki.ros.org/roscpp
http://wiki.ros.org/roscpp/Overview

ROS C++ Client Library (*roscpp*)
Node Handle

- There are four main types of node handles

1. Default (public) node handle:
   `nh_ = ros::NodeHandle();`
2. Private node handle:
   `nh_private_ = ros::NodeHandle("~");`
3. Namespaced node handle:
   `nh_eth_ = ros::NodeHandle("eth");`
4. Global node handle:
   `nh_global_ = ros::NodeHandle("/");`

Recommended

Not recommended

For a ***node*** in ***namespace*** looking up ***topic***, these will resolve to:

```
/namespace/topic

/namespace/node/topic

/namespace/eth/topic

/topic
```

More info
http://wiki.ros.org/roscpp/Overview/NodeHandles

Create first node *Hello World*

with **rospy** (Python Client Library)

```
/*
 * hello.py
*/

#!/usr/bin/env python
# -*- coding utf-8 -*-

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

import rospy
rospy.init_node('hello')

rate = rospy.Rate(10)

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

```
> cd ~/catkin2_ws/src/beginner_tutorials/src
> sudo gedit hello.py
```

Give execution permissions to the file

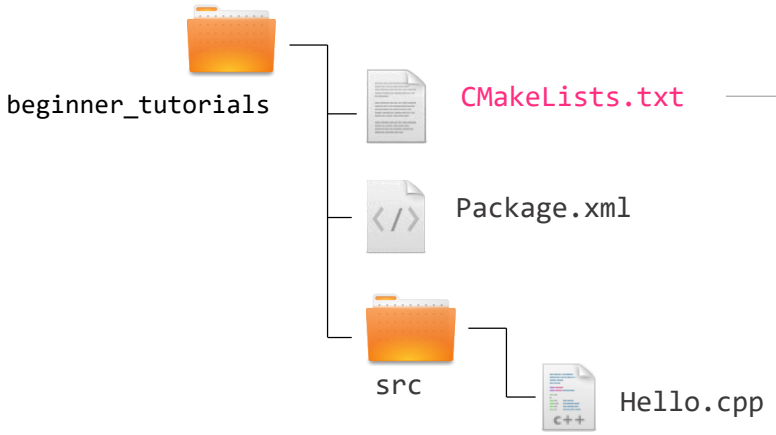
```
> chmod+x hello.py
```

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

Building first node *Hello World*

Customizing CMakeLists.txt

- Before building your node, you should modify the CMakeLists.txt in the package




```
cmake_minimum_required(VERSION 2.8.3)
project(beginner_tutorials)

## Find catkin macros and libraries
find_package(catkin REQUIRED COMPONENTS roscpp rospy
std_msgs)

## Declare ROS messages and services
# add_message_files(FILES Message1.msg Message2.msg)
# add_service_files(FILES Service1.srv Service2.srv)

## Generate added messages and services
# generate_messages(DEPENDENCIES std_msgs)

## Declare catkin package
catkin_package()

## Specify additional locations of header files
include_directories(${catkin_INCLUDE_DIRS})

## Declare a cpp executable
add_executable(hello src/hello.cpp)

## Specify libraries to link a library or executable
target against
target_link_libraries(hello ${catkin_LIBRARIES})
```

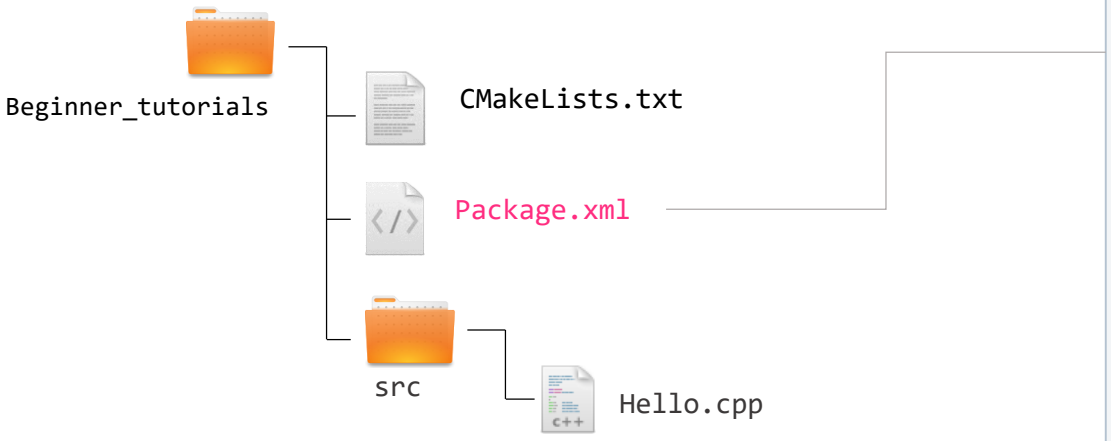
specifies the executable to be created after the build

links libraries and executables

Building first node *Hello World*

Customizing package.xml

- Before building your node, you should modify the generated package.xml in the package




```
<?xml version="1.0"?>

<package format="2">
  <name>beginner_tutorials</name>
  <version>0.1.0</version>
  <description>The beginner_tutorials package</description>

  <maintainer email="luc@univ-smb.fr">Luc Mare</maintainer>
  <license>BSD</license>
  <url type="website">http://wiki.ros.org/beginner_tutorials</url>
  <author email="luc@univ-smb.fr">Luc Marechal</author>

  <buildtool_depend>catkin</buildtool_depend>

  <build_depend>roscpp</build_depend>
  <build_depend>rospy</build_depend>
  <build_depend>std_msgs</build_depend>

  <exec_depend>roscpp</exec_depend>
  <exec_depend>rospy</exec_depend>
  <exec_depend>std_msgs</exec_depend>

</package>
```

Building first node *Hello World*

Build package

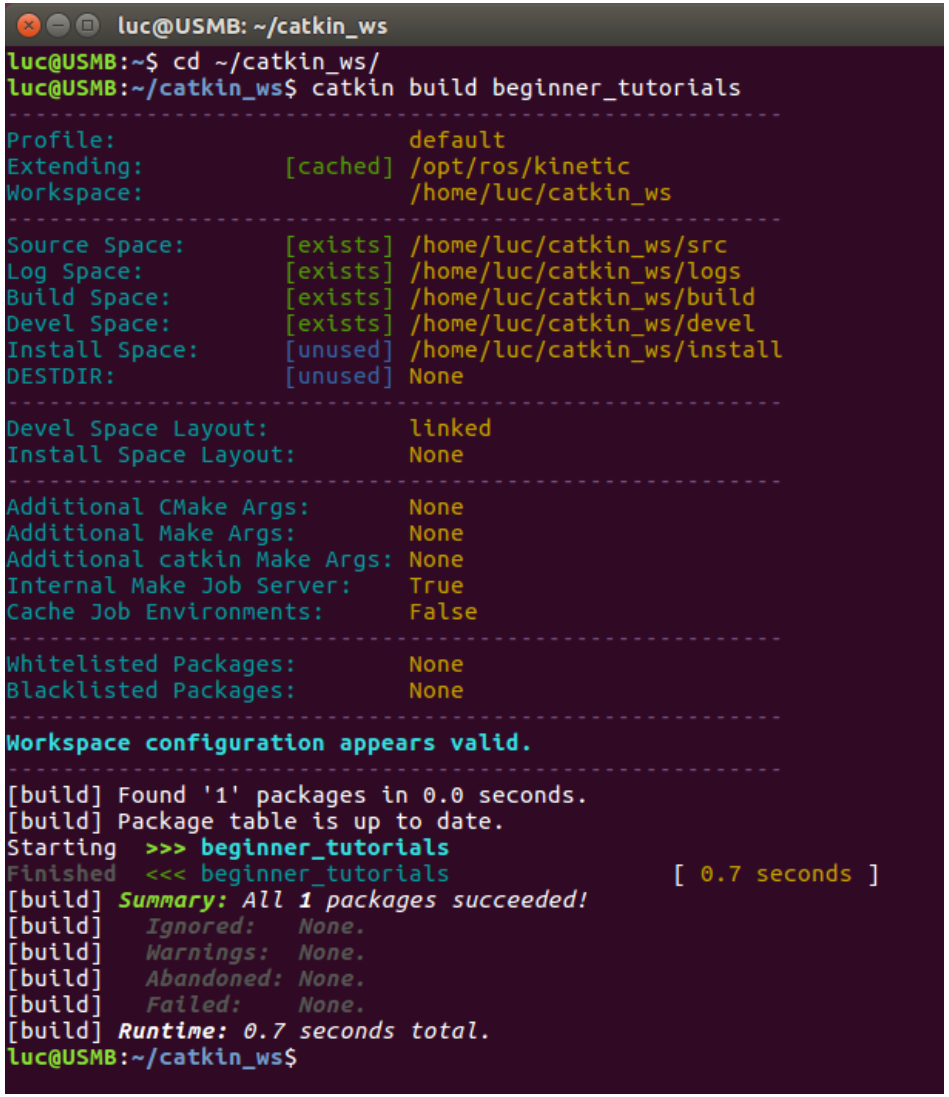
```
> cd ~/catkin_ws
> catkin_make beginner_tutorials
```

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
```



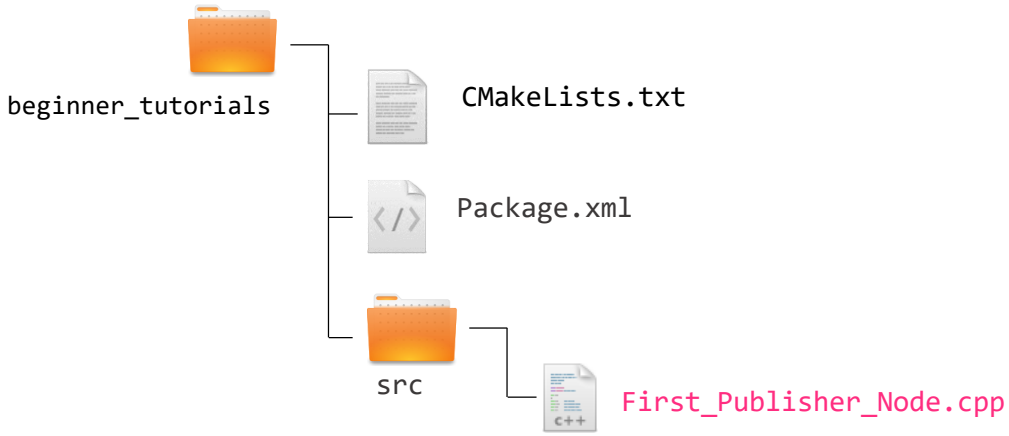
Creating a Publisher and a Subscriber Node (C++)

Writing the publisher Node

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

Edit a cpp file

```
> cd ~/catkin_ws/beginner_tutorials/src
> gedit First_Publisher_Node.cpp
```




```
#include "ros/ros.h"
#include "std_msgs/Int32.h"
#include <iostream>
int main(int argc, char **argv)
{
  ros::init(argc, argv,"First_Publisher_Node");
  ros::NodeHandle node_obj;
  ros::Publisher number_publisher =
node_obj.advertise<std_msgs::Int32>("numbers",10);
  ros::Rate loop_rate(10);
  int number_count = 0;
  while (ros::ok())
  {
    std_msgs::Int32 msg;
    msg.data = number_count;
    ROS_INFO("%d",msg.data);
    number_publisher.publish(msg);
    ros::spinOnce();
    loop_rate.sleep();
    ++number_count;
  }
return 0;
}
```

Creating a Publisher and a Subscriber Node (C++)

Examining the publisher Node

Includes all the headers necessary to use the most comon public pieces of ROS system.

We will send an integer value, so we need to include this header. Standard message definition for integer datatype.

Initialize ROS. Has to be called before other ROS functions. Specifies the name of our node (must be unique in a running system).

The *Nodehandle* object is the access for communication with the ROS system.

Tell the master that we are going to be publishing a message of type *Int32* on the topic *my_numbers*.

ros::Rate is a helper class to run loops a desired frequency (here 10 Hz).

```
#include "ros/ros.h"
#include "std_msgs/Int32.h"
#include <iostream>
int main(int argc, char **argv)
{
  ros::init(argc, argv,"First_Publisher_Node");
  ros::NodeHandle node_obj;
  ros::Publisher number_publisher =
node_obj.advertise<std_msgs::Int32>("numbers",10);
  ros::Rate loop_rate(10);
  int number_count = 0;
  while (ros::ok())
  {
    std_msgs::Int32 msg;
    msg.data = number_count;
    ROS_INFO("%d",msg.data);
    number_publisher.publish(msg);
    ros::spinOnce();
    loop_rate.sleep();
    ++number_count;
  }
return 0;
}
```

Creating a Publisher and a Subscriber Node (C++)

Examining the publisher Node

ros::ok() checks if a node should continue running
Returns false if SIGINT is received (Ctrl + C) or ros::shutdown() has been called
This is an infinite while loop, and it quits when we press Ctrl + C. The *ros::ok()* function returns zero when there is an interrupt; this can terminate this while loop

Creates an integer ROS message, and the second line assigns an integer value to the message. Here, data is the field name of the msg object:

This will print the message data. This line is used to log the ROS information.

Publish the message to the topics numbers. Broadcast the message to anyone who is connected

ROS::spinOnce() processes incoming messages via callbacks

```
#include "ros/ros.h"
#include "std_msgs/Int32.h"
#include <iostream>
int main(int argc, char **argv)
{
  ros::init(argc, argv,"First_Publisher_Node");
  ros::NodeHandle node_obj;
  ros::Publisher number_publisher =
node_obj.advertise<std_msgs::Int32>("numbers",10);
  ros::Rate loop_rate(10);
  int number_count = 0;
  while (ros::ok())
  {
    std_msgs::Int32 msg;
    msg.data = number_count;
    ROS_INFO("%d",msg.data);
    number_publisher.publish(msg);
    ros::spinOnce();
    loop_rate.sleep();
    ++number_count;
  }
return 0;
}
```

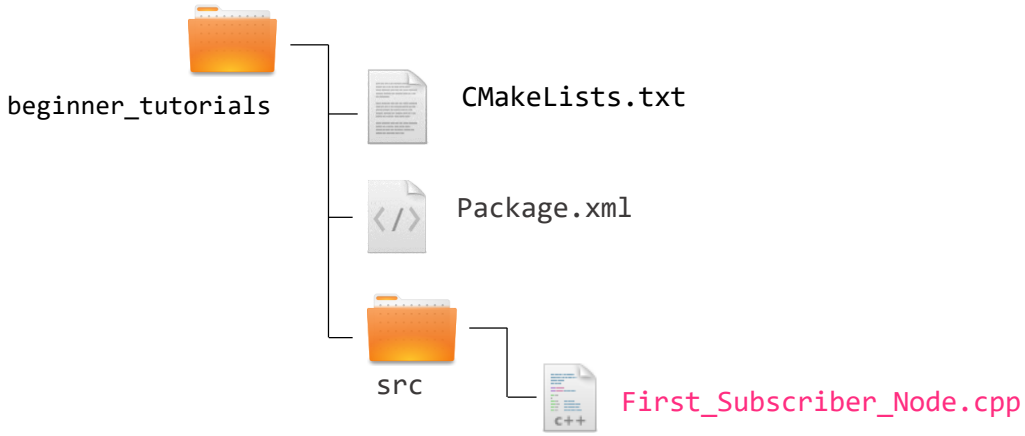
Creating a Publisher and a Subscriber Node (C++)

Writing the subscriber Node

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

Edit a cpp file

```
> cd ~/catkin_ws/beginner_tutorials/src
> gedit First_Subscriber_Node.cpp
```




```
#include "ros/ros.h"
#include "std_msgs/Int32.h"
#include <iostream>
void number_callback(const
std_msgs::Int32::ConstPtr& msg) {
ROS_INFO("Received [%d]",msg->data);
}

int main(int argc, char **argv) {
  ros::init(argc, argv,"First_Subscriber_Node");
  ros::NodeHandle node_obj;
  ros::Subscriber number_subscriber =
node_obj.subscribe("numbers",10,number_callback);
  ros::spin();
  return 0;
}
```

Creating a Publisher and a Subscriber Node (C++)

Examining the subscriber Node

This is a callback function that will execute whenever a data comes to the /numbers topic. Whenever a data reaches this topic, the function will call and extract the value and print it on the console.

Subscribe to the *numbers* topic with the master. ROS will call the `number_callback()` function whenever a new message arrives. The 2nd argument is the queue size, in case we are not able to process messages fast enough. In this case, if the queue reaches 10 messages, we will start throwing away old messages as new ones arrive.

ros::spin() is an infinite loop in which the node will wait in this step. The code will calling message callbacks as fast as possible. The node will quit only when we press the Ctrl+ C key.

```
#include "ros/ros.h"
#include "std_msgs/Int32.h"
#include <iostream>
void number_callback(const
std_msgs::Int32::ConstPtr& msg) {
ROS_INFO("Received [%d]",msg->data);
}

int main(int argc, char **argv) {
  ros::init(argc, argv,"First_Subscriber_Node");
  ros::NodeHandle node_obj;
  ros::Subscriber number_subscriber =
node_obj.subscribe("numbers",10,number_callback);
  ros::spin();
  return 0;
}
```

Creating a Publisher and a Subscriber Node (C++)

Building the nodes

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

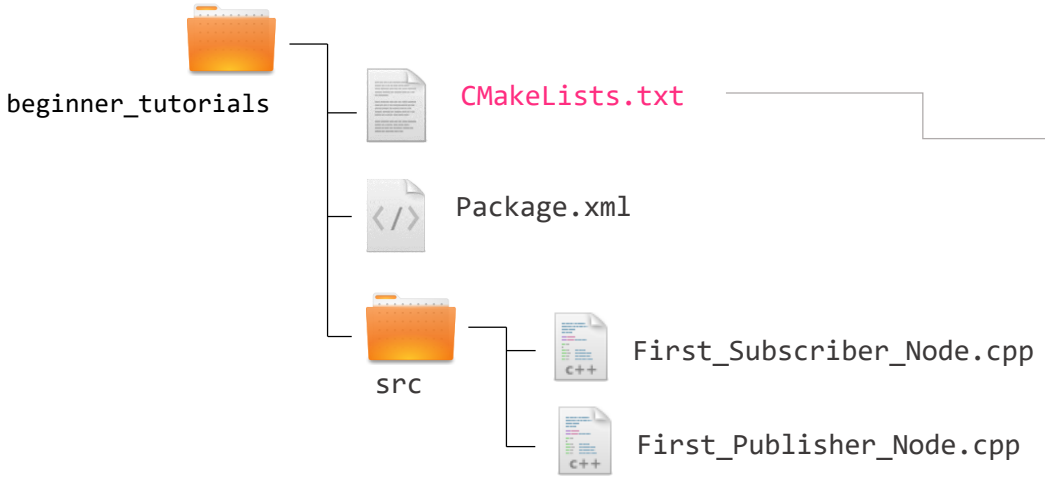
Edit a cpp file

```
> cd ~/catkin_ws/beginner_tutorials/src
> gedit First_Publisher_Node.cpp
```

Building first node *Hello World*

Building the nodes

- Before building your node, you should modify the CMakeLists.txt in the package




```
…

## Declare a cpp executable
add_executable(First_Publisher_Node src/First_Publisher_Node.cpp)
add_executable(First_Subscriber_Node src/First_Subscriber_Node.cpp)

## Dependencies
add_dependencies(First_Publisher_Node beginner_tutorials_generate_messages_cpp)
add_dependencies(First_Subscriber_Node beginner_tutorials_generate_messages_cpp)

## Specify libraries to link a library or executable target against
target_link_libraries(First_Publisher_Node ${catkin_LIBRARIES})
target_link_libraries(First_Subscriber_Node ${catkin_LIBRARIES})

…
```

Build all pakages

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

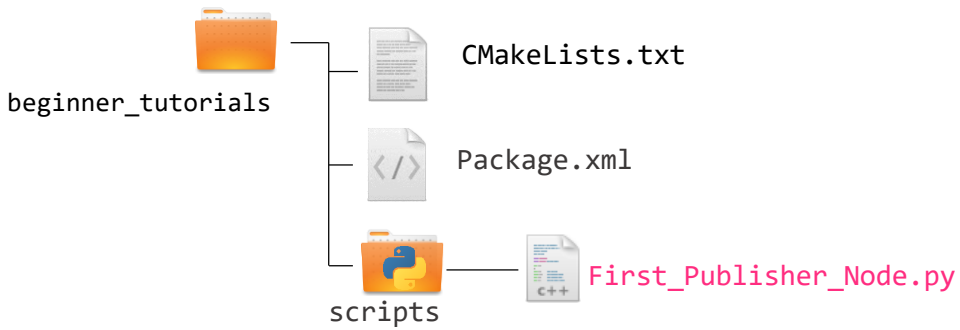
Creating a Publisher and a Subscriber Node (Python)

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/src
> mkdir scripts
> cd scripts
> sudo gedit First_Publisher_Node.py
```




```
#!/usr/bin/env python
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__':
    try:
        First_Publisher_Node()
    except rospy.ROSInterruptException:
        pass
```

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)

```
#!/usr/bin/env python
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__':
    try:
        First_Publisher_Node()
    except rospy.ROSInterruptException:
        pass
```

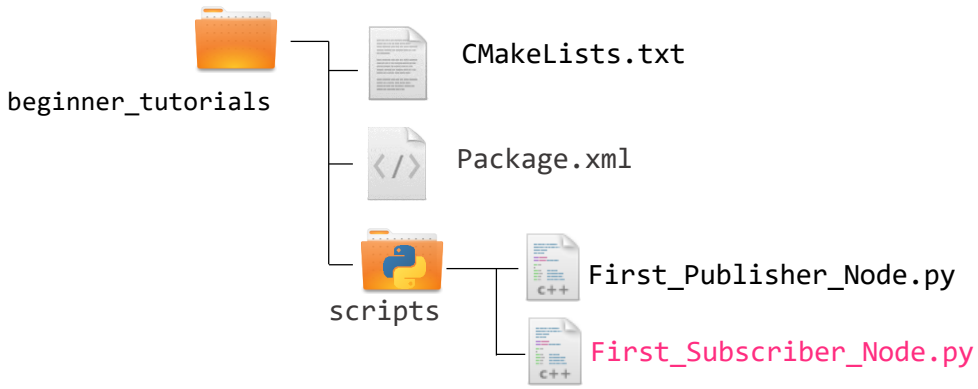
Creating a Publisher and a Subscriber Node (Python)

Writing the subscriber 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/src/scripts
> sudo gedit First_Subscriber_Node.py
```




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

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

`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

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

def callback(data):
    rospy.loginfo(rospy.get_caller_id() + "I heard %s",
data.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()
```

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

Creating a Publisher and a Subscriber Node (Python)

Examining the subscriber Node

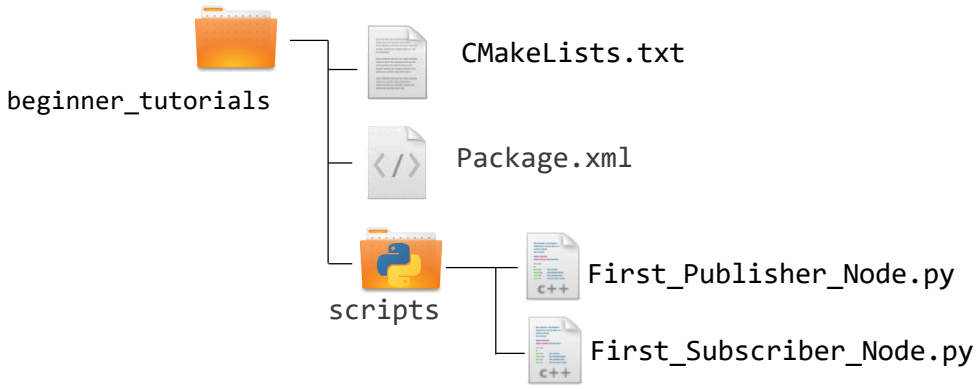
```
rospy.Subscriber(name of the topic, Message, callback function)
```

Creating a Publisher and a Subscriber Node (Python)

Building the nodes

Make the node executable (for Python only)

```
> chmod +x scripts/First_Subscriber_Node.py
> chmod +x scripts/First_Publisher_Node.py
```



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

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

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

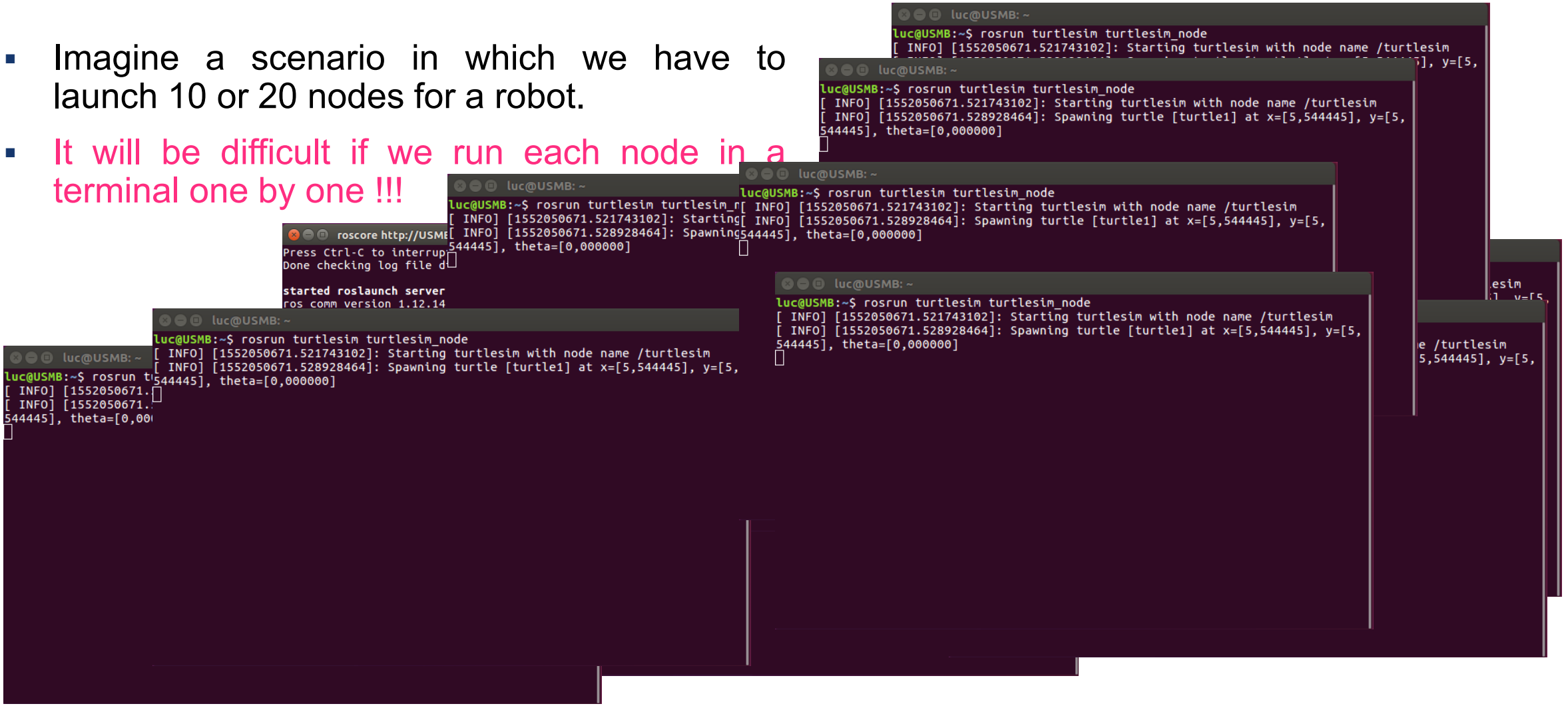
```
> cd ~/catkin_ws
> source ./devel/setup.bash
```

Run your nodes

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

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*
- If not yet running, launch automatically starts a roscore

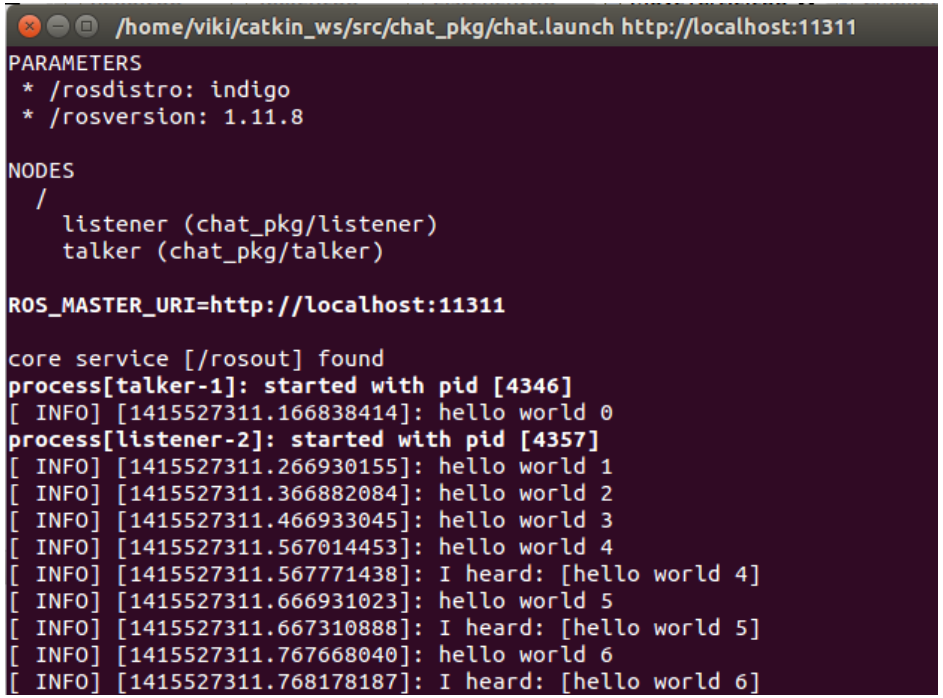
Start a launch file from a package with

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

Browse to the folder and start a launch file with

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

Example console output for:
> `roslaunch chat pkg_chat.launch`



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

ROS Launch

File Structure

! Attention when copy & pasting code from the internet

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)

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)

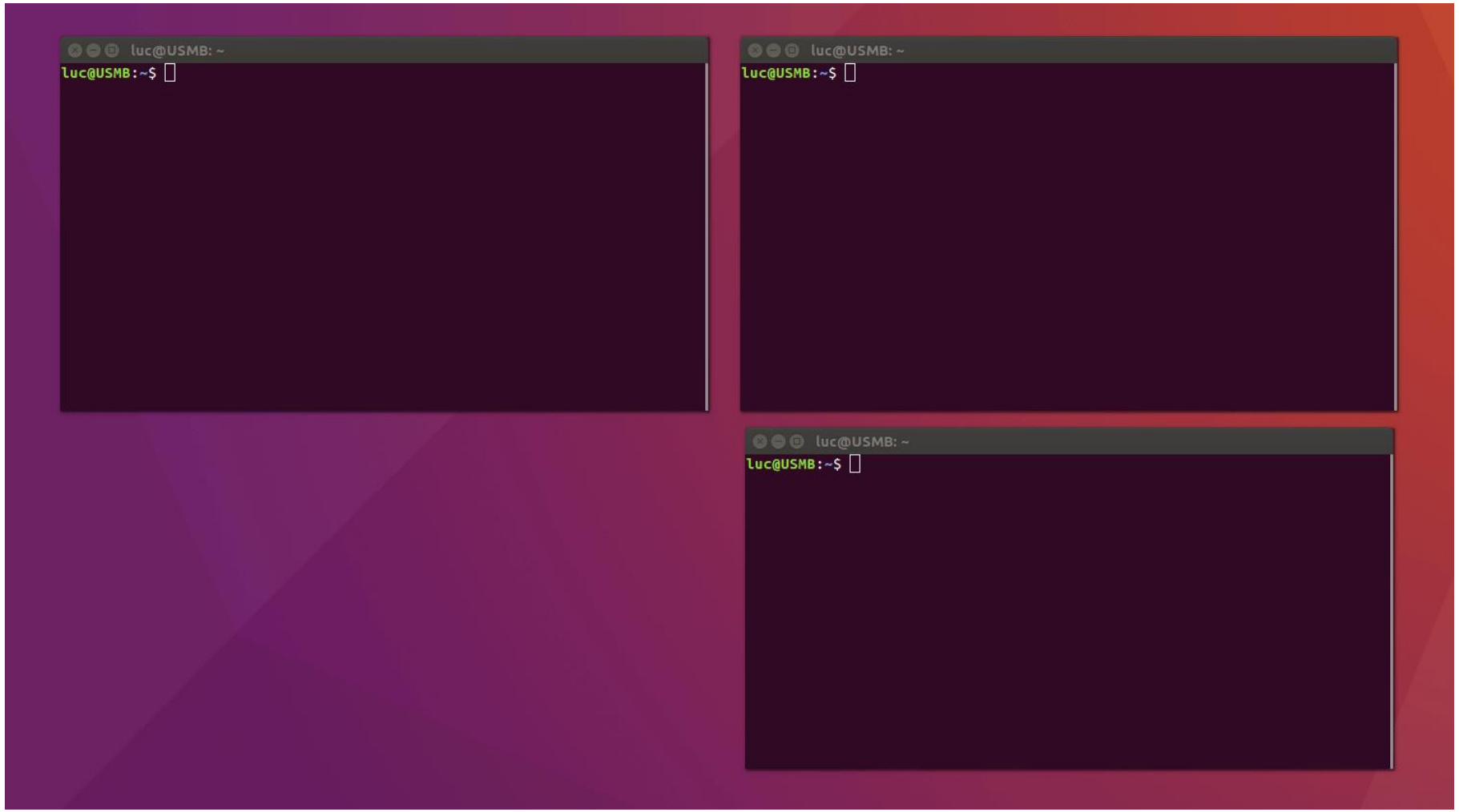
```
<?xml version="1.0"?>
<launch>
  <arg name="use_sim_time" default="true"/>
  <arg name="world" default="gazebo_ros_range"/>
  <arg name="debug" default="false"/>
  <arg name="physics" default="ode"/>

  <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>
```

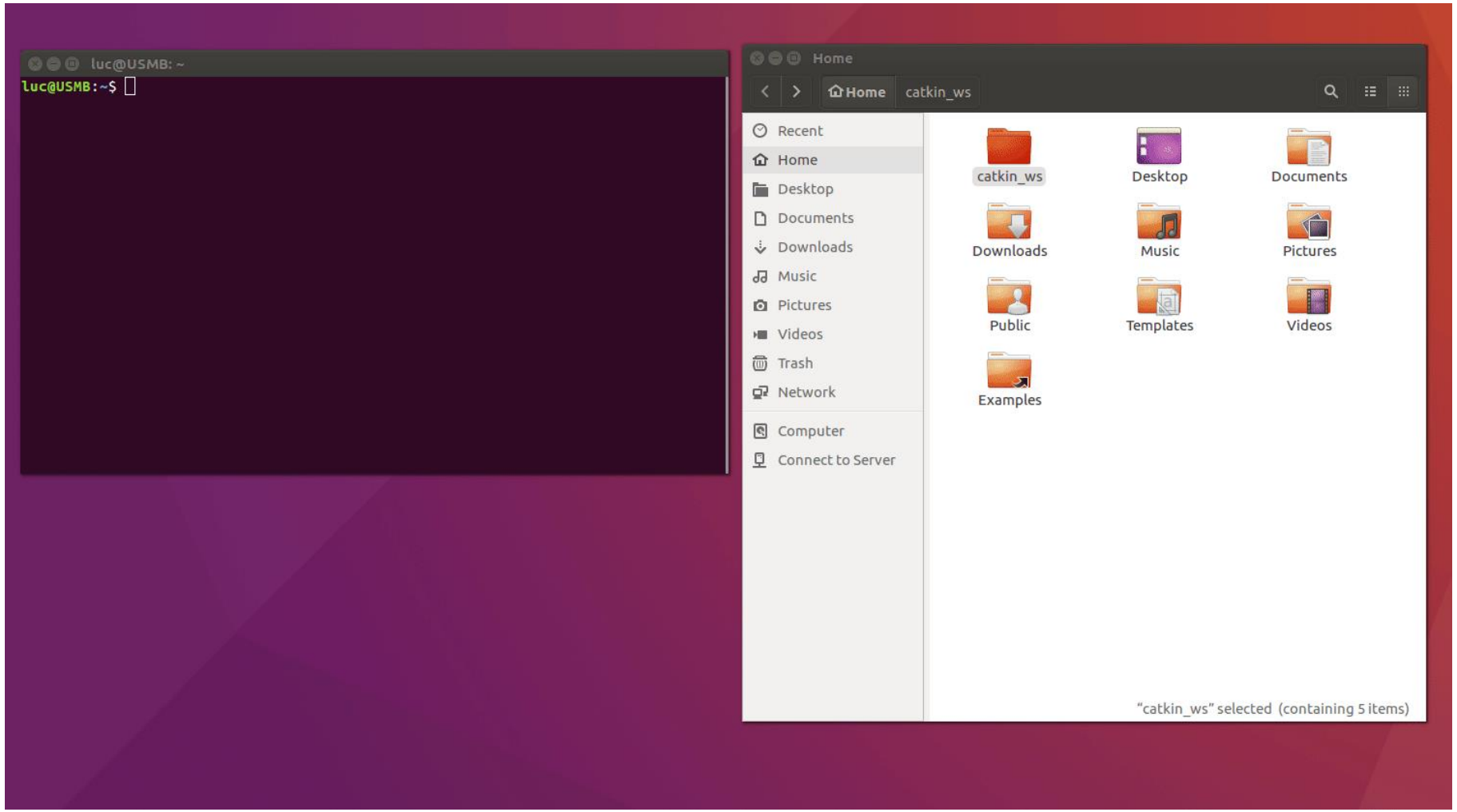
ROS Launch

Running 2 nodes from Terminal



ROS Launch

Running 2 nodes with *roslaunch*



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 ( )

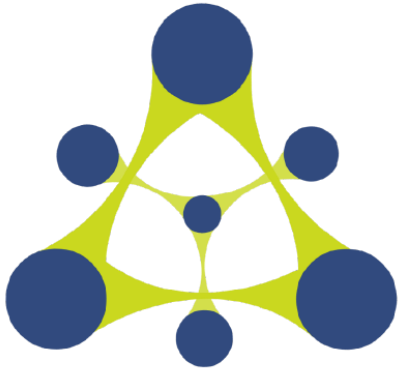
RoboWare Studio: IDE especially designed for working with ROS. The installation is quite easy, and automatically detects and loads an ROS environment without additional configurations. It has different out-of-the-box features ( )

RoboWare Installation

Go to http://www.roboware.me/ and download the latest version of the software.

Install deb file

```
> cd /path/to/deb/file
> sudo dpkg -i roboware-studio_<version>_<architecture>.deb
```



Launch RoboWare in a console

```
> roboware-studio
```

That's it !

More info
http://www.roboware.me

RoboWare Installation

Setup a project in Sublime3

Sublime 3 installation

```
> sudo add-apt-repository ppa:webupd8team/sublime- text-3
> sudo apt-get update
> sudo apt-get install sublime-text-installer
```

package_control installation

```
In Sublime : View > Show Console menu. Once open, paste the Python code

import urllib.request,os,hashlib; h = '6f4c264a24d933ce70df5dedcf1dcaee' +
'ebe013ee18cced0ef93d5f746d80ef60'; pf = 'Package Control.sublime-package'; ipp =
sublime.installed_packages_path(); urllib.request.install_opener( urllib.request.build_opener(
urllib.request.ProxyHandler()) ); by = urllib.request.urlopen( 'http://packagecontrol.io/' +
pf.replace(' ', '%20')).read(); dh = hashlib.sha256(by).hexdigest(); print('Error validating
download (got %s instead of %s), please try manual install' % (dh, h)) if dh != h else
open(os.path.join( ipp, pf), 'wb' ).write(by)
```

More info
http://schulz-m.github.io/2016/07/12/sublime-for-catkin/

Setup a project in Sublime3

Setup project

Open sublime and then simply add your catkin_ws/src folder by using

Project -> Add Folder to Project, then save it as a project file in a location of your choice by Project -> Save Project as....

Build Tag definition

```
> sudo apt-get install exuberant-ctags
```

In SublimeType Alt + Shift + P and type install. Type ctags and put enter.

More info

http://schulz-m.github.io/2016/07/12/sublime-for-catkin/

Setup a project in Sublime3

Build catkin package from Sublime

It is very straightforward to build catkin packages using catkin tools within sublime.
Simply install the catkin builder using the package manager. You can then either run it with ctrl + shift + p and then Build with: Catkin or using Tools -> Build.

ROS Msg File Syntax

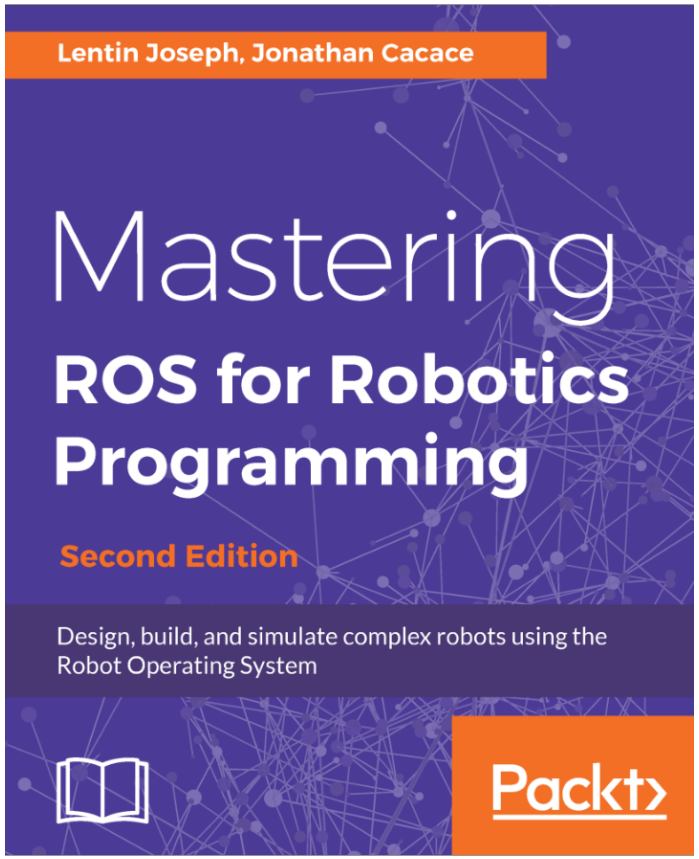
Download the syntax file from Github and copy the file into ~.config/sublime-text-3/Packages/User.
Now you can simply select *ROS message definition* with View -> Syntax -> Open all with current extension as....

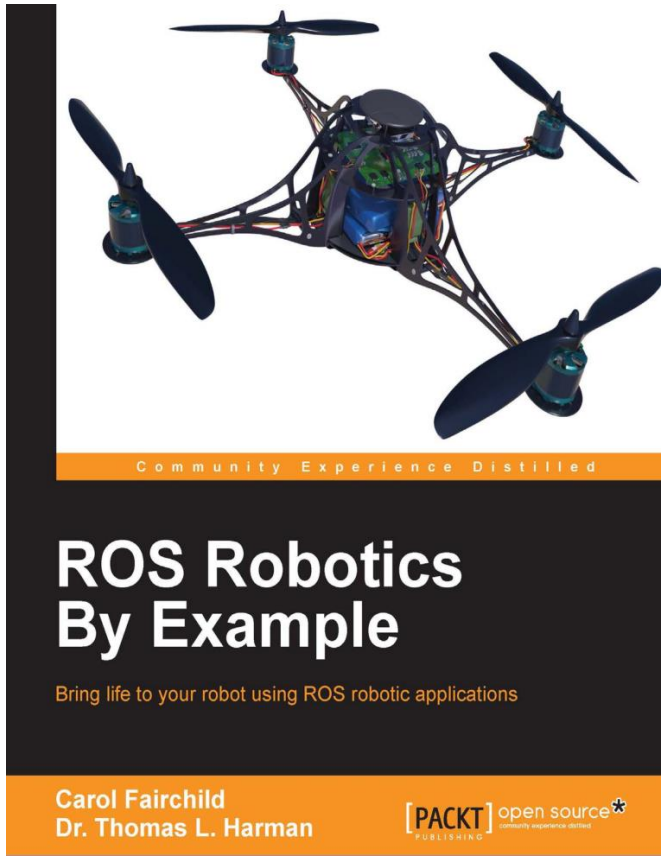
More info
http://schulz-m.github.io/2016/07/12/sublime-for-catkin/

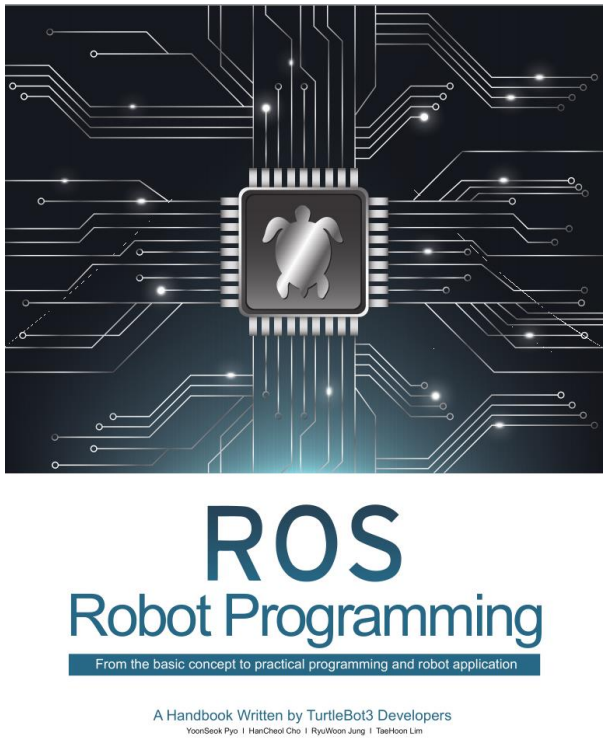
Further References

- 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)

SYMME

UNIVERSITÉ SAVOIE MONT BLANC
POLYTECH ANNECY-CHAMBÉRY