



2019

**INFO 802**

**Master Advanced Mechatronics**

Luc Marechal



**ROS**

**Introduction  
Course 2**

# Create first node *Hello World* (C++)

## with roscpp (C++ Client Library)

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

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

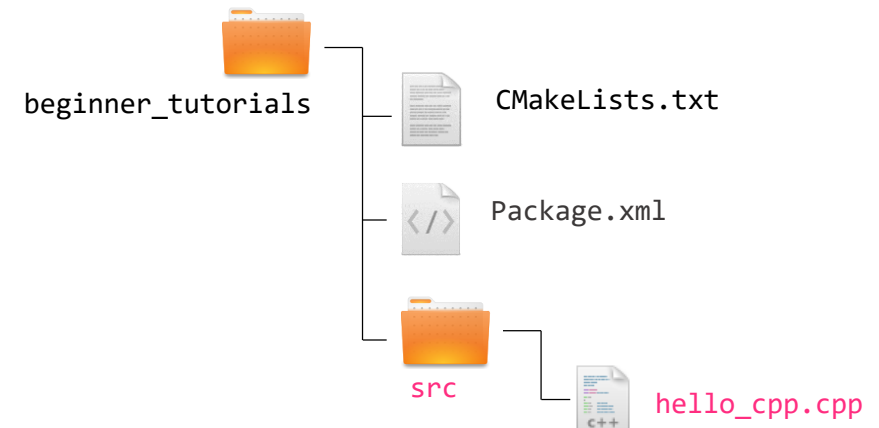
    unsigned int count = 0;
    while (ros::ok()) {
        ROS_INFO_STREAM("Hello World " << count);
        ros::spinOnce();
        loopRate.sleep();
        count++;
    }

    return 0;
}
```

ROS main header file include

### Create the node

```
> cd ~/catkin_ws/src/beginner_tutorials/src
> sudo gedit hello_cpp.cpp
```



# Create first node *Hello World*

## with roscpp (C++ Client Library)

```
#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

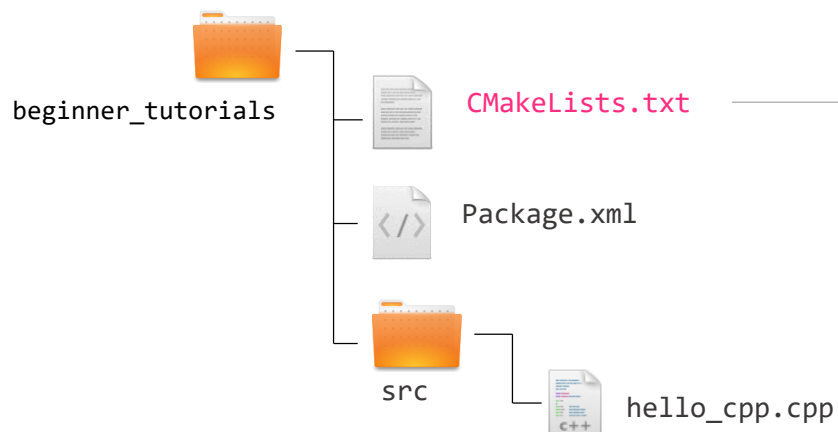
For a *node* in *namespace* looking up *topic*, these will resolve to:

`/namespace/topic``/namespace/node/topic``/namespace/eth/topic``/topic`**More info**<http://wiki.ros.org/roscpp/Overview/NodeHandles>

# Building first node *Hello World* (C++)

## Customizing CMakeLists.txt

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



specifies the executable to be created after the build

links libraries and executables

```
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})

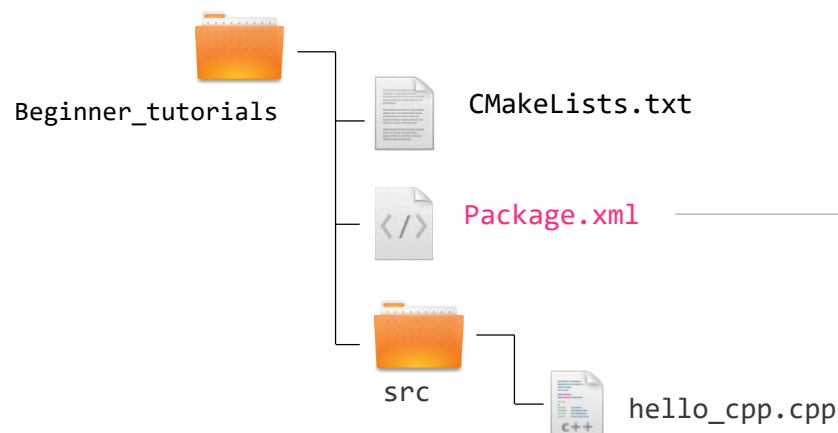
## Declare a cpp executable
add_executable(hello_cpp src/hello_cpp.cpp)

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

# Building first node *Hello World* (C++)

## 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* (C++)

## 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_cpp
```

```
luc@USMB: ~/catkin_ws  
luc@USMB:~$ cd ~/catkin_ws/  
luc@USMB:~/catkin_ws$ catkin build beginner_tutorials  
-----  
Profile: default  
Extending: [cached] /opt/ros/kinetic  
Workspace: /home/luc/catkin_ws  
-----  
Source Space: [exists] /home/luc/catkin_ws/src  
Log Space: [exists] /home/luc/catkin_ws/logs  
Build Space: [exists] /home/luc/catkin_ws/build  
Devel Space: [exists] /home/luc/catkin_ws/devel  
Install Space: [unused] /home/luc/catkin_ws/install  
DESTDIR: [unused] None  
-----  
Devel Space Layout: linked  
Install Space Layout: None  
-----  
Additional CMake Args: None  
Additional Make Args: None  
Additional catkin Make Args: None  
Internal Make Job Server: True  
Cache Job Environments: False  
-----  
Whitelisted Packages: None  
Blacklisted Packages: None  
-----  
Workspace configuration appears valid.  
-----  
[build] Found '1' packages in 0.0 seconds.  
[build] Package table is up to date.  
Starting >>> beginner_tutorials  
Finished <<< beginner_tutorials [ 0.7 seconds ]  
[build] Summary: All 1 packages succeeded!  
[build] Ignored: None.  
[build] Warnings: None.  
[build] Abandoned: None.  
[build] Failed: None.  
[build] Runtime: 0.7 seconds total.  
luc@USMB:~/catkin_ws$
```



# Create first node *Hello World (Python)*

## with rospy (Python Client Library)

```
#!/usr/bin/env python
# -*- 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')

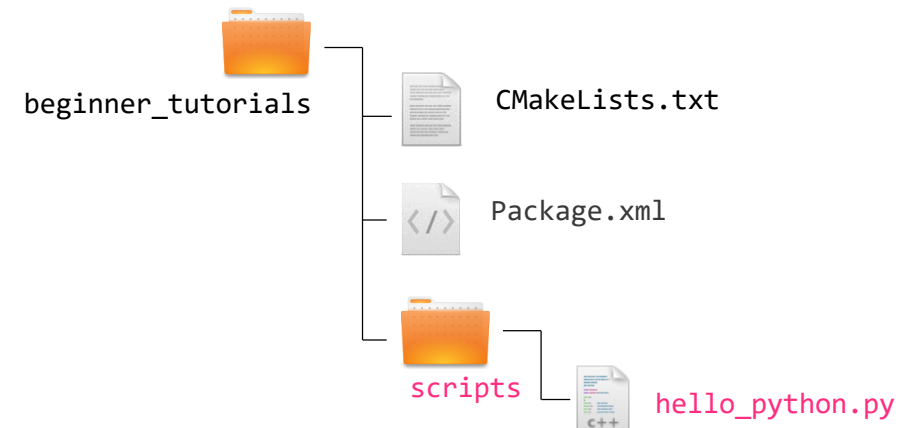
rate = rospy.Rate(10)

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

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





# Building first node *Hello World (Python)* with rospy (Python Client Library)

Make the file executable

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

→ Give execution permissions to the file



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

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

Run your node

```
> rosrn beginner_tutorials hello_python.py
```

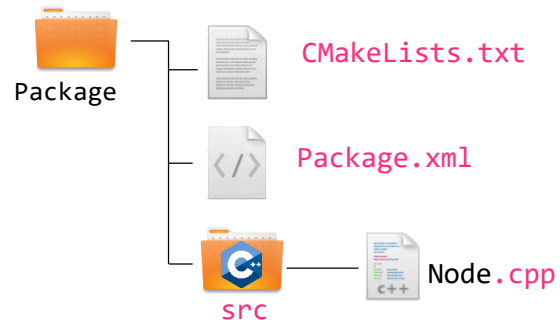
→ Extension needed

[http://www2.ece.ohio-state.edu/~zhang/RoboticsClass/docs/ECE5463\\_ROSTutorialLecture1.pdf](http://www2.ece.ohio-state.edu/~zhang/RoboticsClass/docs/ECE5463_ROSTutorialLecture1.pdf)

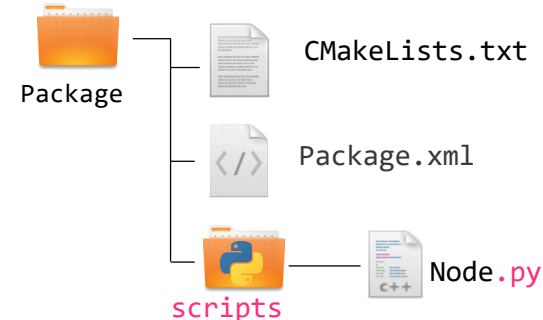
# Create Nodes Summary



- 1) Create your .cpp file in /src folder
- 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
- 2) Make the file executable
- 3) Source your workspace
- 4) Run your node with the .py extension



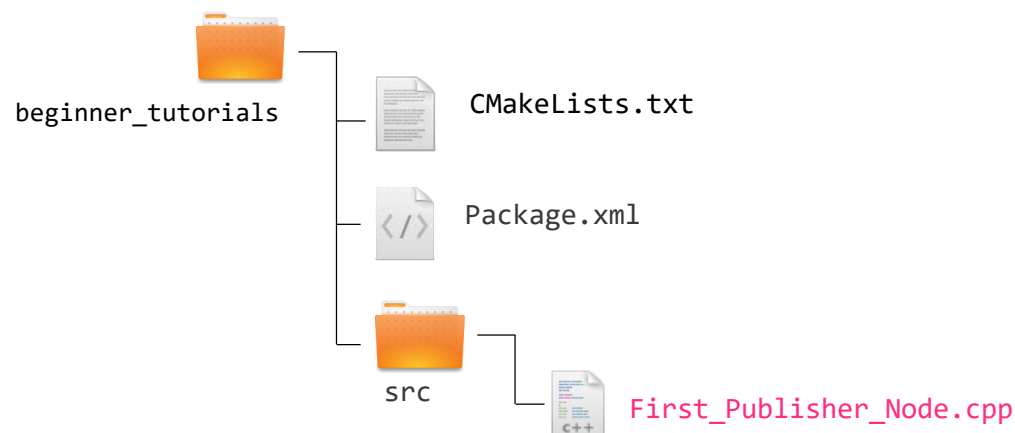
# Creating a Publisher and a Subscriber Node (C++)

## 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 common 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 at 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;
}
```

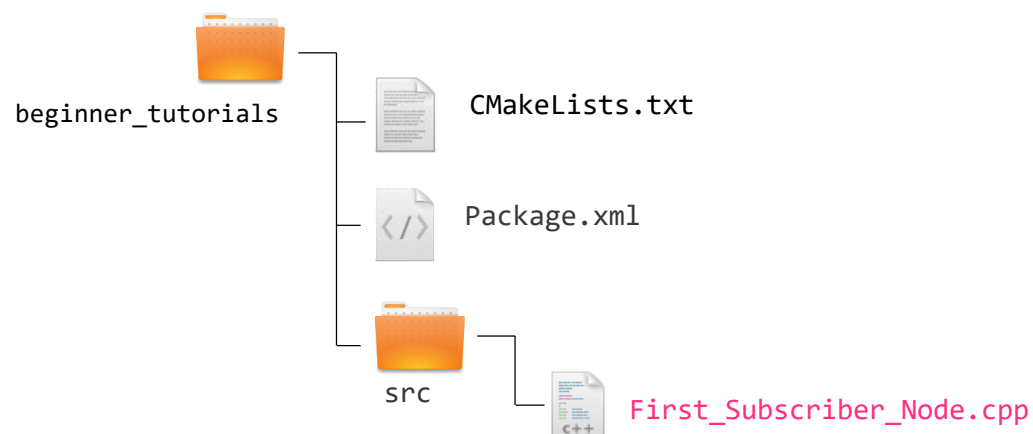
# Creating a Publisher and a Subscriber Node (C++)

## 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 call 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*

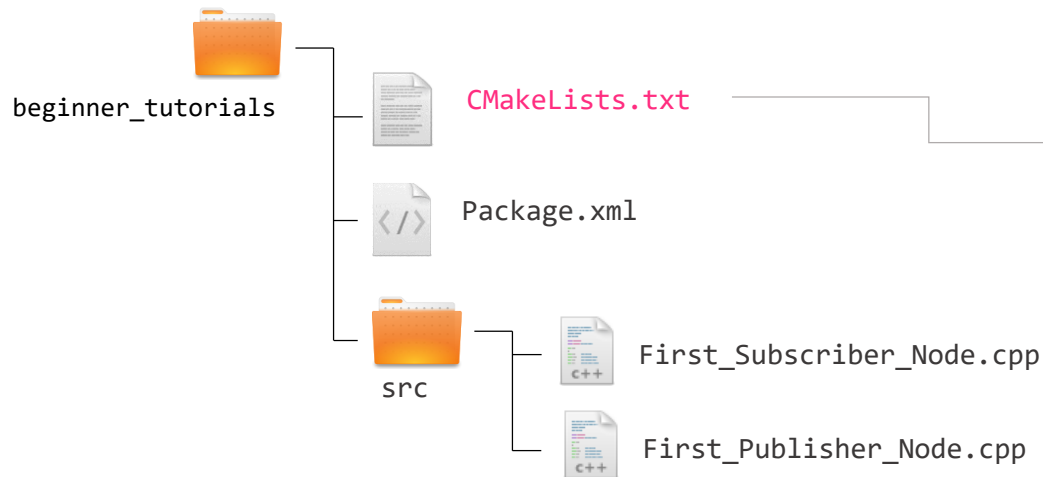
Edit a cpp file

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

# Building Publisher and a Subscriber Node (C++)

## 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 packages

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

# 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

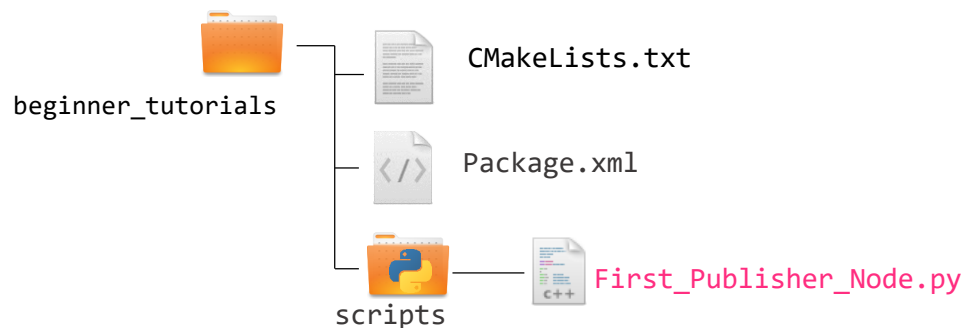
# Creating a Publisher and a Subscriber Node (Python)

## 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 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)

## Examining the **publisher** Node

```
rospy.Publisher(name of the topic, message type, queue size)
```

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

### More info

<http://wiki.ros.org/rospy/Overview/Publishers%20and%20Subscribers>

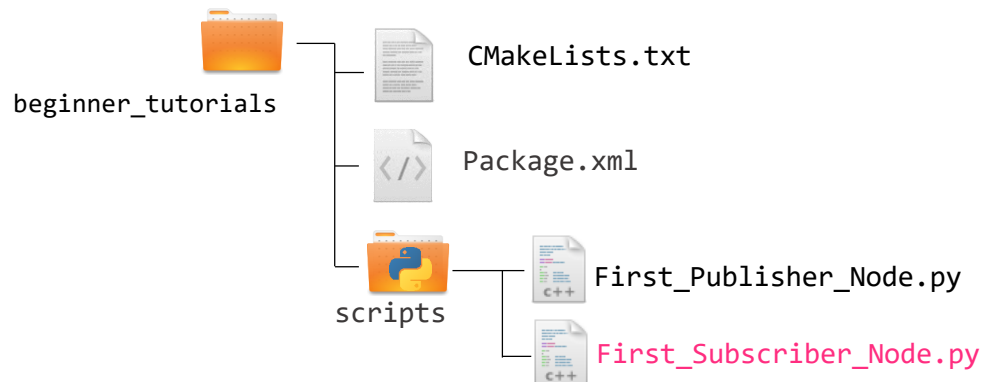
# Creating a Publisher and a Subscriber Node (Python)

## 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 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 type, callback function)
```

# Creating a Publisher and a Subscriber Node (Python)

## Building the nodes

Make the node executable (for Python only)

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

### Build package

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

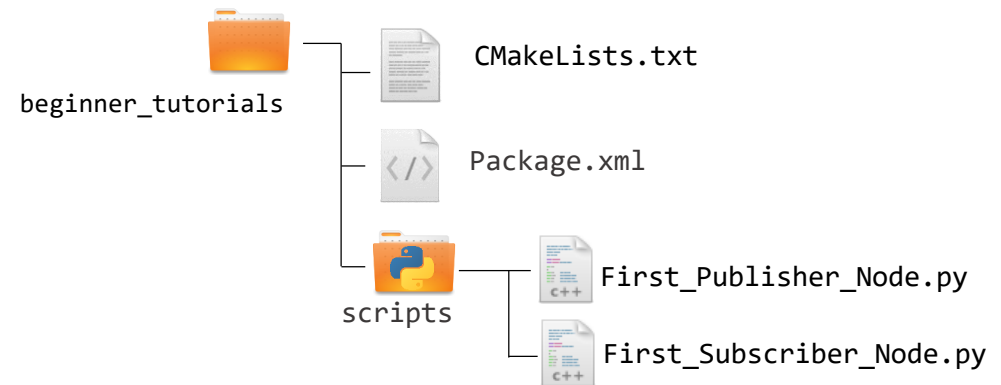
```
> cd ~/catkin_ws  
> catkin_make
```

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

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

### Run your nodes

```
> rosrn beginner_tutorials First_Publisher_Node.py  
> rosrn beginner_tutorials First_Subscriber_Node.py
```



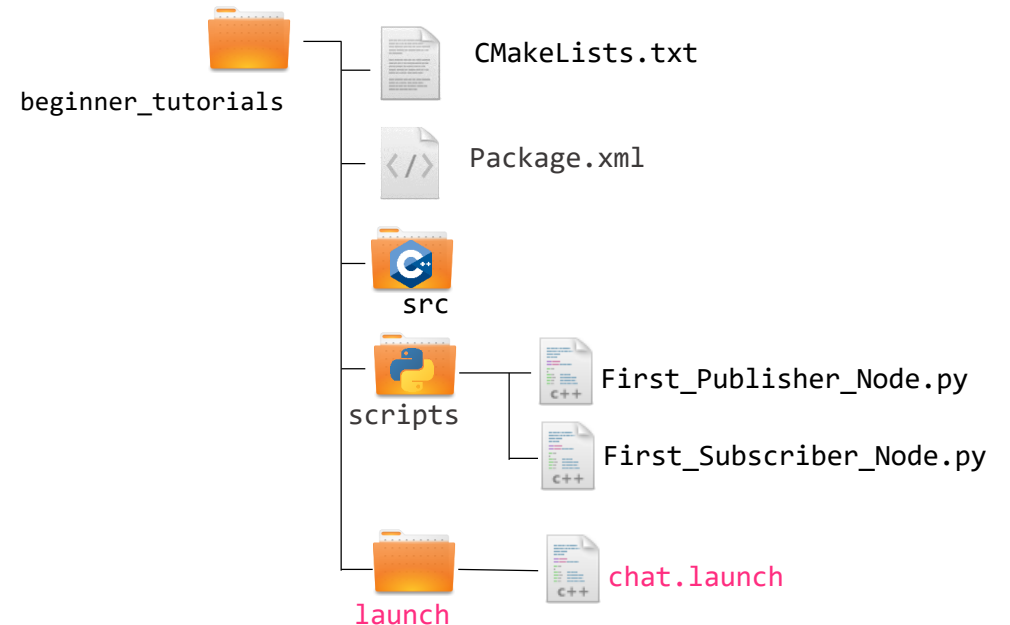


# ROS Launch

- *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 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

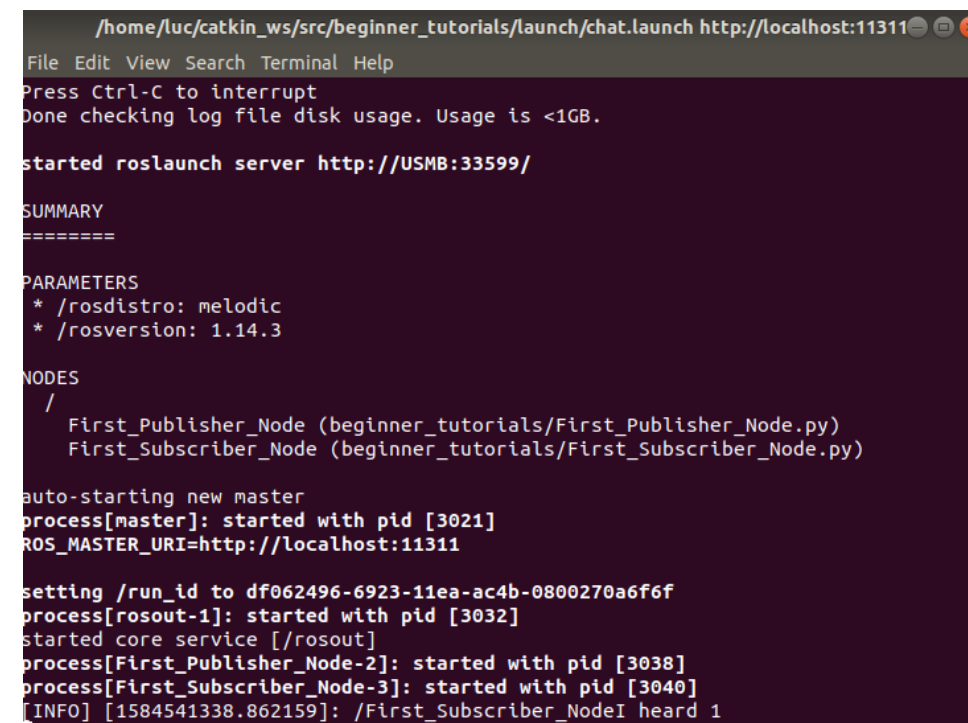
```
> roslaunch [package_name] [file_name.launch]
```

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
* /roscdistro: 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

## 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** extension)
- **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** extension)
- **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)*

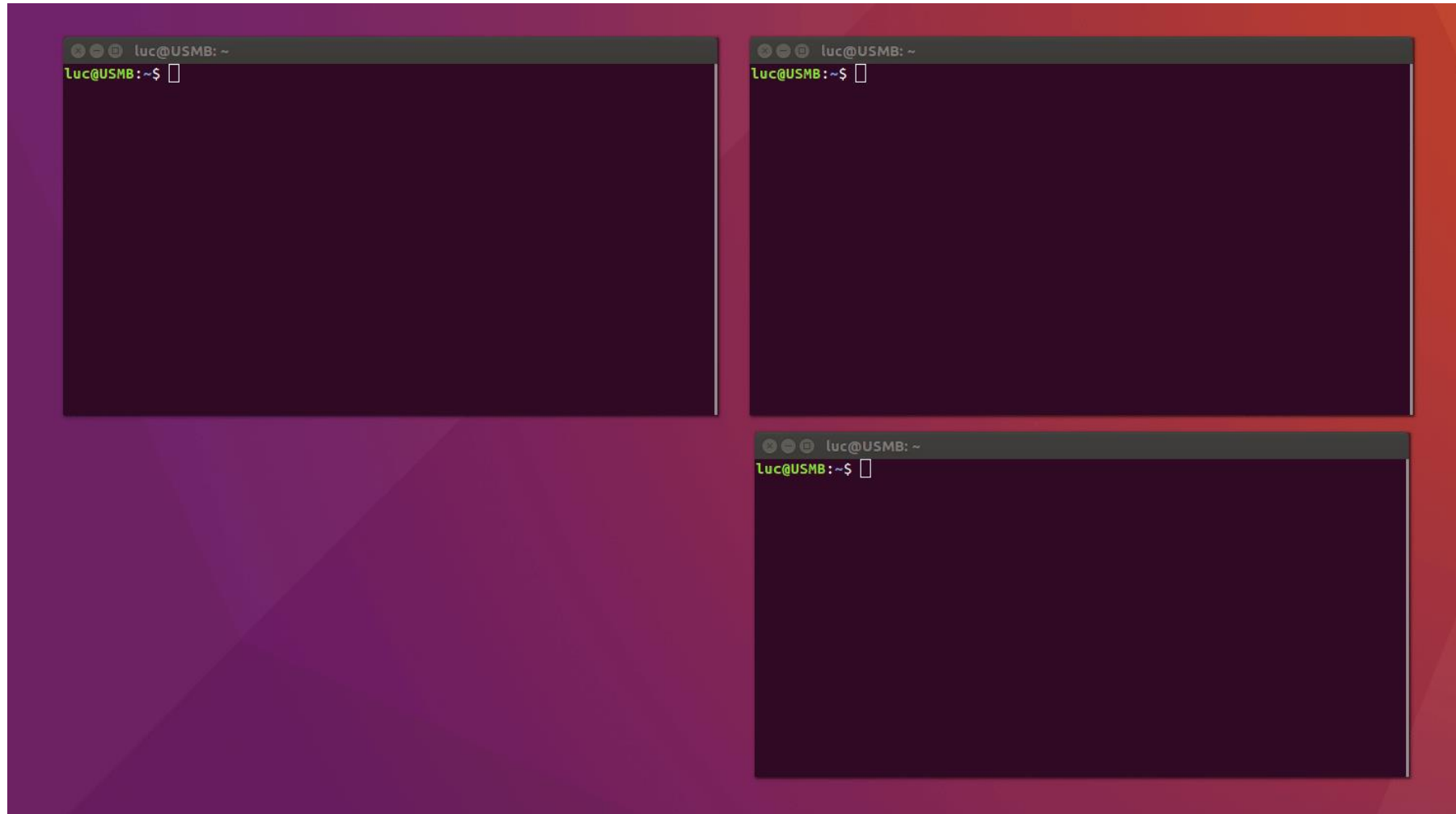
```
<?xml version="1.0"?>
<launch>
  <arg name="use_sim_time" default="true"/>
  <arg name="world" default="gazebo_ros_range"/>
  <arg name="debug" default="false"/>
  <arg name="physics" default="ode"/>

  <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>
```

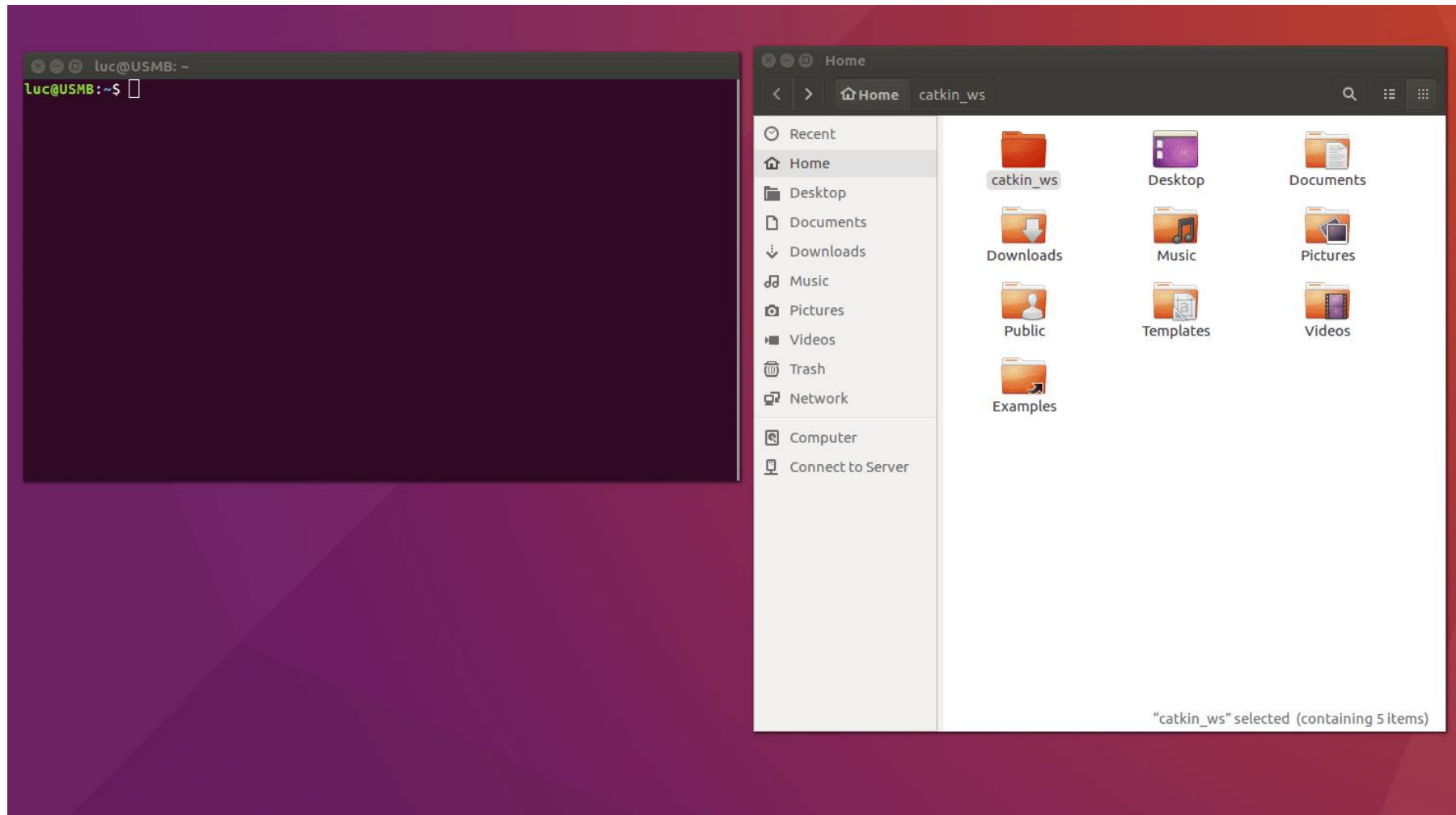
# ROS Launch

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 (🐱)

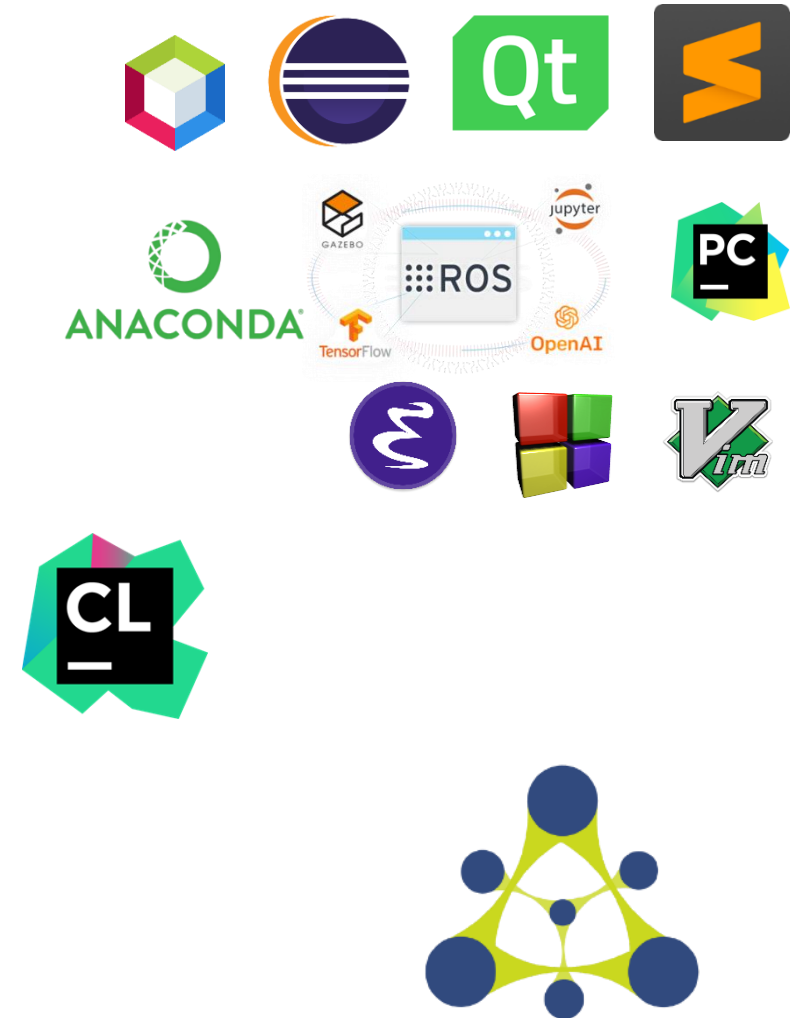
**Anaconda:** nice interface (🐍)

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 !

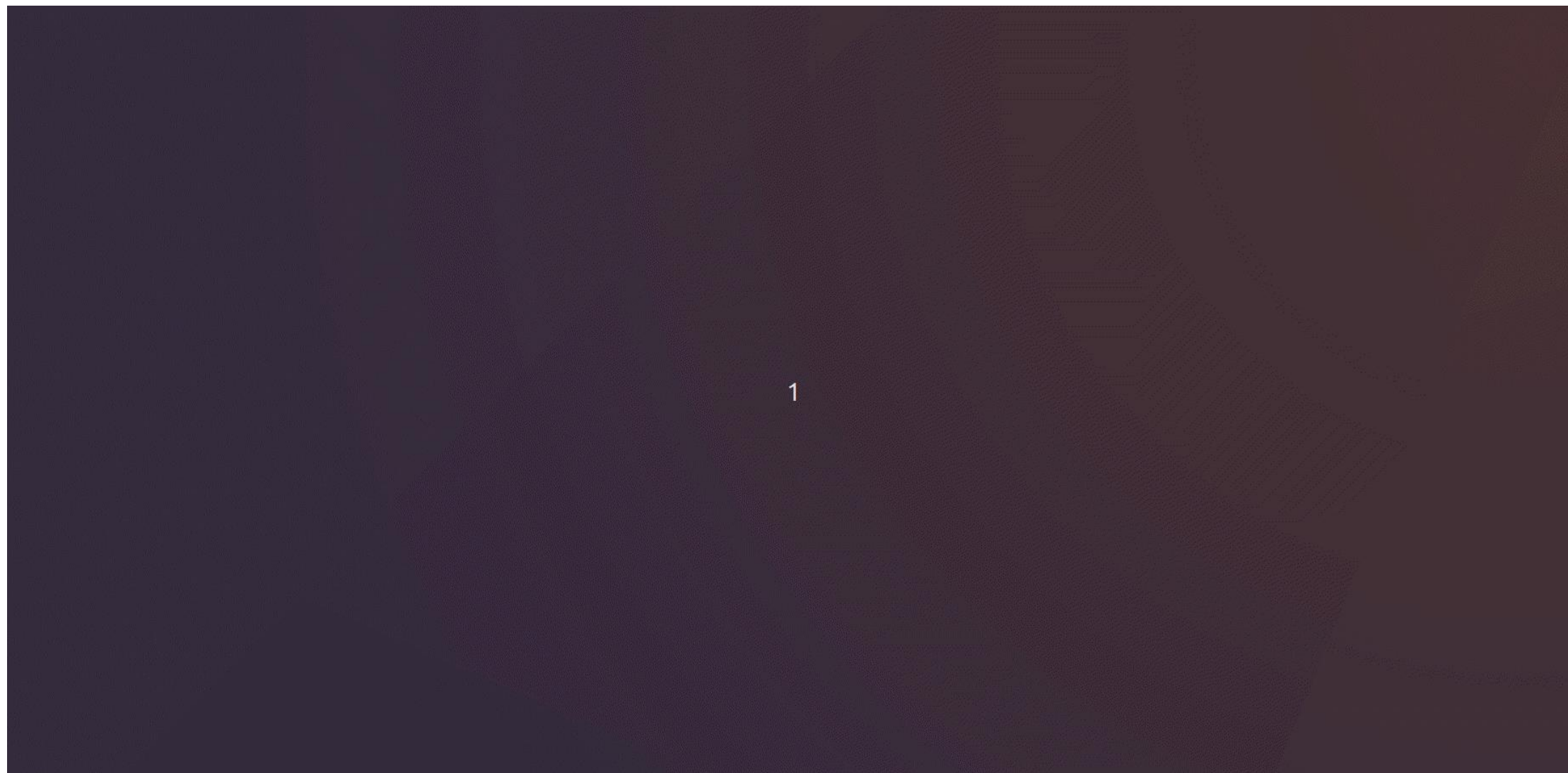


**More info**

<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/](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....

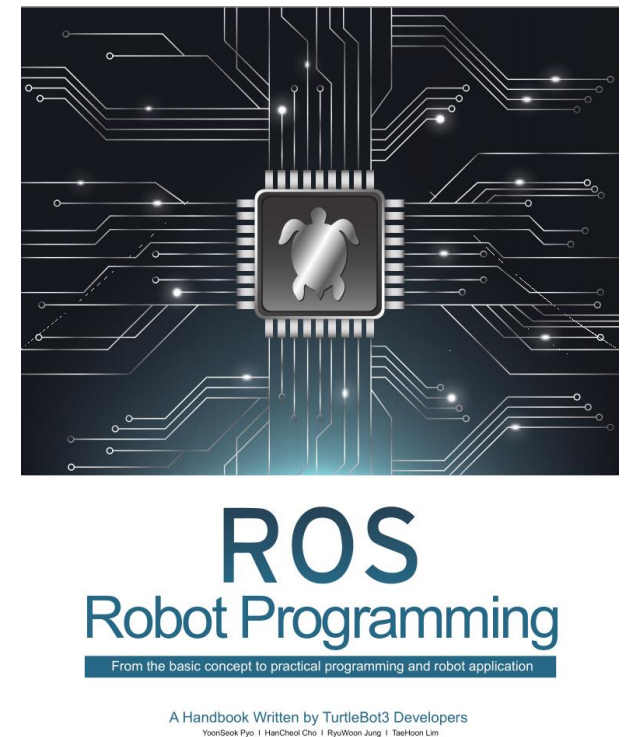
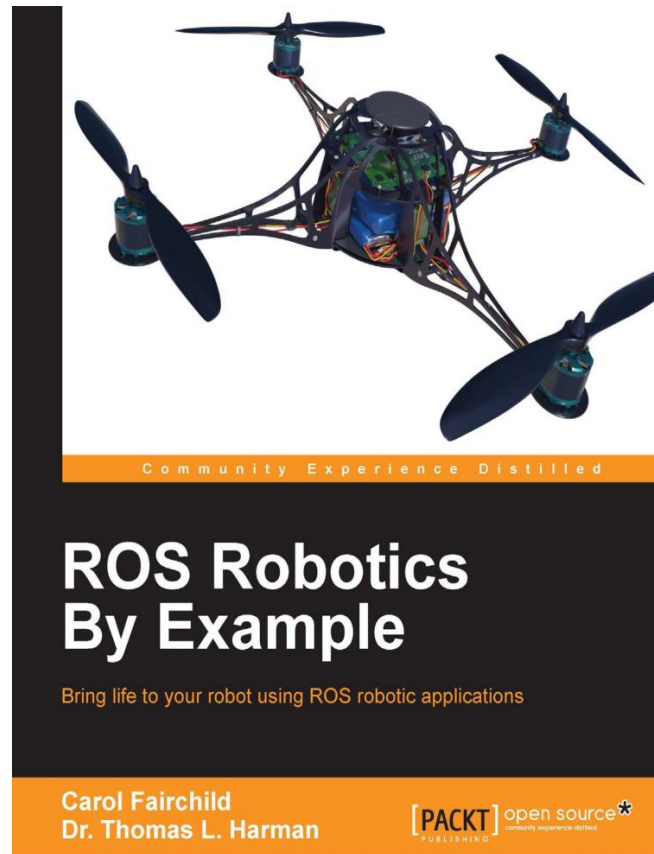
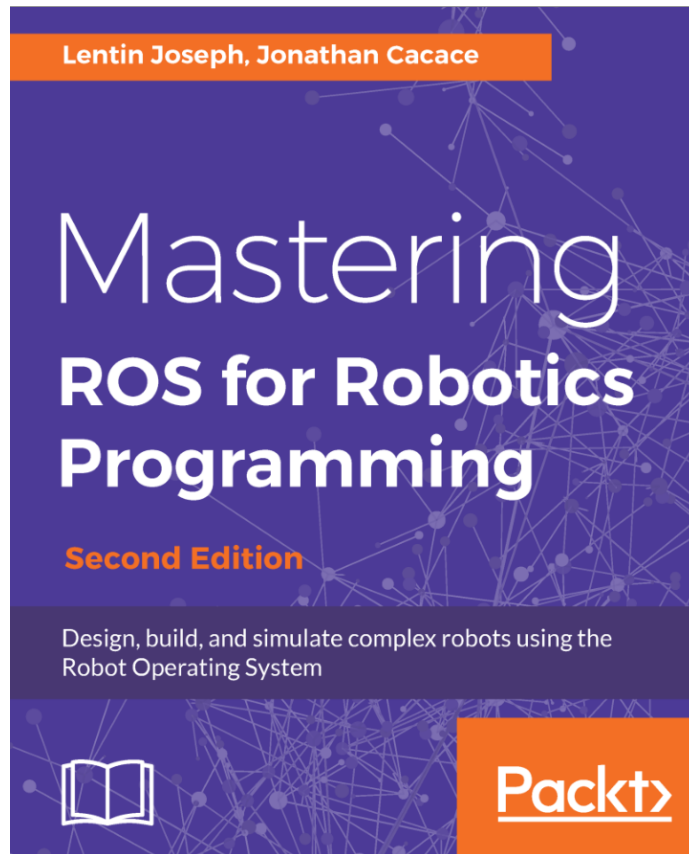
**More info**

<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/](https://kapeli.com/cheat_sheets/ROS.docset/)
- **ROS Best Practices**
  - [https://github.com/leggedrobotics/ros\\_best\\_practices/wiki](https://github.com/leggedrobotics/ros_best_practices/wiki)
- **ROS Package Template**
  - [https://github.com/leggedrobotics/ros\\_best\\_practices/tree/master/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 Chambéry  
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