Tutorial 4

First Node, Launch Files & Advanced Programming

Ralph SEULIN

Le2i UMR CNRS 6306 – http://le2i.cnrs.fr
ralph.seulin@u-bourgogne.fr

v1.4 – January 2016

1- Introduction

This tutorial is mainly inspired from the workshop of GDR Robotique @Montpellier – July 2013.

Special thanks to Stéphane Magnenat & Francis Colas – Autonomous Systems Lab. ETH Zurich

The goal of this tutorial is to cover the following topics :

- Learn how to create a package
- Learn how to create and compile a node in Python
- Learn how to create launch files

1.1- References

- ROS Wiki - http://wiki.ros.org/
- ROS Tutorials - http://wiki.ros.org/ROS/Tutorials

1.2- Tips

- **ROS Distribution version**

  ROS Hydro is the only distribution for TurtleBot2 development during Robotics Pojects

  !!! **Always select Hydro in tutorials or documentation on the ROS wiki !!!**

- **Buildsystem**

  A buildsystem generates executable files from source files. Catkin is the only buildsystem used for new development. (Unless you need to compile "old fashioned" packages with rosbuild)

  !!! **Always select catkin in tutorials or documentation on the ROS wiki !!!**

  Consult the ROS Wiki for further information: http://wiki.ros.org/catkin_or_rosbuild

2- Create a package

In order to be integrated to ROS, a ROS node have to be included in a package. We will use the *catkin* build system in this tutorial.

2.1- To Do

- Study the tutorial on creating a ROS package:

  http://wiki.ros.org/ROS/Tutorials/CreatingPackage

!!! ROS environment is already set on the computer: do not create again a workspace !!!

- Move to the source folder of your catkin workspace

- Create a package named *beginner_tutorials*, with the following dependencies:
  - *std_msgs (for string messages)*
  - *rospy* (to program in Python)
  - *roscpp* (to program in C++ - for later use in tutorial 5)

- Study the different files in the new created folder with the help of the following wiki page:

  http://wiki.ros.org/ROS/Tutorials/CreatingPackage#ROS.2BAC8-Tutorials.2BAC8-catkin.2BAC8-CreatingPackage.What_makes_up_a_catkin_Package.3F

- Build the empty package (in the catkin workspace) and study the new created folders and files in the catkin workspace with the help of the following wiki page:

  http://wiki.ros.org/catkin/workspaces

3- Subscriber and Publisher node

3.1- To Do

- Study the ROS tutorial on Subscriber and Publisher in Python on the ROS wiki website:
  - http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber%28python%29
- Download the scripts talker.py & listener.py files within the *beginner_tutorials/scripts* folder
- Don't forget to make the nodes executable
- Build the nodes
- Study the ROS tutorial on “Examining the Simple Publisher and Subscriber” on the ROS wiki website:
  - http://wiki.ros.org/ROS/Tutorials/ExaminingPublisherSubscriber
- Run the talker node and study published data.
- Run the listener node and study running nodes and messages with *rqt_graph*.
- Study the influence of the anonymous state on the nodes by running multiple talker and listener nodes
  - **anonymous=True -** The anonymous keyword argument is mainly used for nodes where you normally expect many of them to be running and don't care about their names (e.g. tools, GUIs). Unique names are more important for nodes like drivers, where it is an error if more than one is running. If two nodes with the same name are detected on a ROS graph, the older node is shutdown.

  More infos: http://wiki.ros.org/rospy/Overview/Initialization%20and%20Shutdown
- Additional infos:
  - The **queue_size** argument is New in ROS hydro and limits the amount of queued messages if any subscriber is not receiving the them fast enough. In older ROS distributions just omit the argument.

talker.py

```
1  #!/usr/bin/env python
2  # Software License Agreement (BSD License)
3
4  import rospy
5  from std_msgs.msg import String
6
7  def talker():
8      pub = rospy.Publisher('chatter', String, queue_size=10)
9      rospy.init_node('talker', anonymous=True)
10     rate = rospy.Rate(10) # 10hz
11     while not rospy.is_shutdown():
12         hello_str = "hello world %s" % rospy.get_time()
13         rospy.loginfo(hello_str)
14         pub.publish(hello_str)
15         rate.sleep()
16
17 if __name__ == '__main__':
18     try:
19         talker()
20     except rospy.ROSInterruptException:
21         pass
22
```

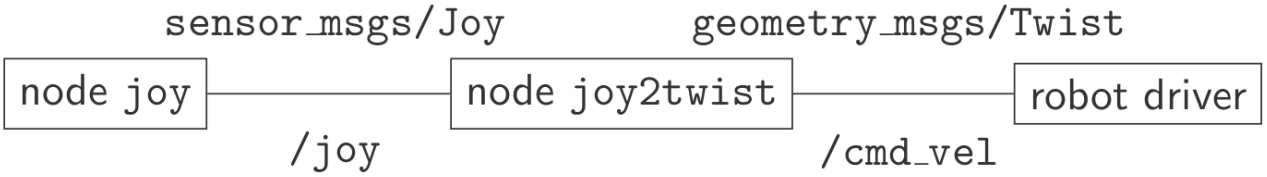
listener.py

```
1   #!/usr/bin/env python
2   # Software License Agreement (BSD License)
3
4   import rospy
5   from std_msgs.msg import String
6
7   def callback(data):
8       rospy.loginfo(rospy.get_caller_id() + "I heard %s", data.data)
9
10  def listener():
11
12      # In ROS, nodes are uniquely named. If two nodes with the same
13      # node are launched, the previous one is kicked off. The
14      # anonymous=True flag means that rospy will choose a unique
15      # name for our 'talker' node so that multiple talkers can
16      # run simultaneously.
17      rospy.init_node('listener', anonymous=True)
18
19      rospy.Subscriber("chatter", String, callback)
20
21      # spin() simply keeps python from exiting until this node is stopped
22      rospy.spin()
23
24  if __name__ == '__main__':
25      listener()
```

4- Joy to Twist Conversion

4.1- Objective

We will create a node that subscribes to the topic */joy*, reads a message of *sensor_msgs/Joy* type and depending on the joypad axes values, publishes a message of *geometry_msgs/Twist* type on the */cmd_vel* topic in order to control a differential robot (e.g. TurtleBot2).



To drive the robot with the joystick, the *joy2twist* node will:

- Subscribe to */joy* topic and read *sensor_msgs/Joy* messages: joypad axes & buttons.
- Publish *geometry_msgs/Twist* messages on */cmd vel* topic: linear & angular velocity of the robot.

We will program this exercice in Python.

The joypad axes and buttons values can be generated by a real joystick or with a bag file.

4.2- /Joy Topic

- Play the joy.bag with *rqt_bag* and publish the */joy* topic.

- Check values of the */joy* topic within *rqt_bag* and with the *rostopic echo* command.

4.3- Program

- Create a package named *joy2twist*, with the following dependencies:
  - *rospy* (to program in Python)
  - *std_msgs* (for string messages)
  - *sensor_msgs* (for Joy message definition)
  - *geometry_msgs* (for Twist message definition)

- Create the script scripts/joy2twist.py file within the *joy2twist* package.

- Add a subscriber to the /joy topic and display the data (*sensor_msgs/Joy* messages) provided by the joystick in the callback function.

joy2twist.py

```python
#!/usr/bin/env python

## Simple listener demo that listens to sensor_msgs/joy published
## to the '/joy' topic

import roslib; roslib.load_manifest('joy2twist')
import rospy
from sensor_msgs.msg import Joy

def callback(data):
    # display the fully-qualified name of the node
    # + data axes values for every received message
    rospy.loginfo(rospy.get_name() + _
        _ ": Get from Joy [%.2f, %.2f,%.2f,%.2f,%.2f,%.2f]" % data.axes)

def listener():

    # Init node called "listener"
    rospy.init_node('listener', anonymous=True)

    # suscribe to topic /joy
    #and calls the callback function for every received message
    rospy.Subscriber("/joy", Joy, callback)

    # spin() simply keeps python from exiting until this node is stopped
    rospy.spin()

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

- Run the node and check that displayed data correspond to the *rostopic echo* values.

- Add a publisher to publish a *Twist* message for every received *Joy* message:
  - JoyPad axis “0” drives the “z” Robot Angular Velocity
  - JoyPad axis “1” drives the “x” Robot Linear Velocity

New joy2twist.py with publisher

```python
#!/usr/bin/env python

## Simple conversion demo that
## suscribes to sensor_msgs/joy published to the '/joy' topic
## and converts to geometry_msgs/twist
## to be published to /cmd_vel topic

import roslib; roslib.load_manifest('joy2twist')
import rospy
from sensor_msgs.msg import Joy
from geometry_msgs.msg import Twist


def callback(data):
    # display the fully-qualified name of the node
    # + data axes values for every received message
    rospy.loginfo(rospy.get_name() + _
    _": Get from Joy [%.2f, %.2f,%.2f,%.2f,%.2f,%.2f]" % data.axes)
    msg = Twist()
    msg.linear.x=data.axes[1]
    msg.angular.z=data.axes[0]
    pub.publish(msg)

def listener():
    global pub

    # Init node called "listener"
    rospy.init_node('listener', anonymous=True)

    # suscribe to topic /joy
    # and calls the callback function for every received message
    rospy.Subscriber("/joy", Joy, callback)

    # publish to topic /cmd_vel
    pub = rospy.Publisher('/cmd_vel', Twist)

    # spin() simply keeps python from exiting until this node is stopped
    rospy.spin()

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

- Run the node and check published values with the *rostopic echo* command.

4.4- Test control of simulated robot

We will test the programmed node on the simulated robot we installed during “Tutorial 3 - Packages, Simulation & Basic Control”

- Make sure *roscore* is running
- Launch the simulated TurtleBot as follows:

  roslaunch rbx1_bringup fake_turtlebot.launch
- Bring up RViz so we can observe the simulated robot in action:

  rosrun rviz rviz -d `rospack find rbx1_nav`/sim.rviz
- Check that the *joy2twist* controls the simulated robot

4.5- Control with a real joystick

Logitech Wireless Gamepad F710 Setup Memo

- **SWITCH D/X**
  - D - 6 axes - for teleop manual joy2twist
  - X - 8 axes - not used
- MODE Led
  - **Off - axes 0 & 1 active for left stick - for teleop**
  - On - axes 4 & 5 active for left stick - not used
- Turtlebot Logitech teleop
  - **LB button** - for **dead man - always activate** to perform movement

More infos:

http://gaming.logitech.com/en-us/gaming-controllers/f710-wireless-gamepad

http://support.logitech.com/en_us/product/wireless-gamepad-f710

http://support.logitech.com/article/21360

Joystick Control

- Connect the Joystick Wireless Dongle to the workstation.
- Configure the Joystick in Linux ubuntu:
  - *http://wiki.ros.org/joy/Tutorials/ConfiguringALinuxJoystick*
- Start the *joy_node* and check values of the */joy* topic with *rostopic echo.*
- Run the *joy2twist* node.
- Check that the joystick controls the simulated robot (see 4.4).

5- Setup Launch files

Launch files are used for starting many nodes at once and to manage large projects.

5.1- Todo

- Study the following ROS tutorials about launch files on the ROS wiki website:
  - http://wiki.ros.org/ROS/Tutorials/UsingRqtconsoleRoslaunch#Using_roslaunch
  - http://wiki.ros.org/ROS/Tutorials/Roslaunch%20tips%20for%20larger%20projects
- Create a launch folder in the joy2twist package folder
- Create a launch file that starts the following items:
  - *Bring up RViz so we can observe the simulated robot in action:*

    rosrun rviz rviz -d `rospack find rbx1_nav`/sim.rviz
  - *fake_turtlebot.launch* from *rbx1_bringup* package
  - *joy2twist* node
  - play the *joy.bag* file
- Check that the bag file playing controls the simulated robot by doing the following:
  - Launch the created launch file

- Create another a launch file that starts the following items:
  - *Bring up RViz so we can observe the simulated robot in action:*

    rosrun rviz rviz -d `rospack find rbx1_nav`/sim.rviz
  - *fake_turtlebot.launch* from *rbx1_bringup* package
  - *joy2twist* node
  - ***joy_node* node**
- Check that the real joystick controls the simulated robot by doing the following:
  - Launch the created launch file

6- Advanced programming

6.1- Remapping Arguments

Any ROS name within a node can be remapped when it is launched at the command-line or from a launch file. This is a powerful feature of ROS that lets you launch the same node under multiple configurations from the command-line or a launch file. The names that can be remapped include the node name, topic names, and parameter names.

- Study the following ROS tutorials about remapping arguments & launch files:
  - http://wiki.ros.org/Remapping%20Arguments
  - http://wiki.ros.org/roslaunch/XML
  - http://wiki.ros.org/roslaunch/XML/remap

Todo

- Try this example that starts joy2twist node to publish topic /my_velocity instead of /cmd_vel

  $ rosrun joy2twist joy2twist.py cmd_vel:=/my_velocity

  Study the published topic.
- Create a launch file to start joy2twist in order to publish an adapted topic of velocity commands to drive the Turtlebot2.

- Create another launch file to start:
  - rviz to observe the Turtlebot2 state
  - joy2twist to publish to *cmd_vel_mux/input/teleop*
  - *joy_node node*

- Check that the created launch file is working correctly by doing the following:
  - Launch the minimal.launch on TurtleBot2 Netbook
  - Connect the Joystick to the Workstation
  - Launch the launch_joy_node_realTurtlebot.launch on the workstation
  - Drive the Turtlebot2 with the Joystick

6.2- Python programming with class definition

- Study the Python documentation for detailed info :
  - http://docs.python.org/2/tutorial/classes.html
- Check examples in ROS by Example vol. 1 eBook:
  - 7.6.3 The Timed Out-and-Back Script
  - 9.4 A Voice-Control Navigation Script

Todo

- Program the “Joy to Twist Conversion” exercises with class definition and management of shutdown.

```python
#!/usr/bin/env python
import roslib; roslib.load_manifest('joy2twist')
import rospy
from sensor_msgs.msg import Joy
from geometry_msgs.msg import Twist

class NewJoy2Twist():
    def __init__(self):
        # Give the node a name
        rospy.init_node('NewJoy2Twist', anonymous=False)

        # Set rospy to execute a shutdown function when exiting
        rospy.on_shutdown(self.shutdown)

        # Publisher to control the robot's speed
        self.cmd_vel=rospy.Publisher('/cmd_vel', Twist)

        # Suscriber to Joypad commands
        rospy.Subscriber("/joy", Joy, self.callback)

        # Keeps your node from exiting until the node has been shutdown
        rospy.spin()

    def callback(self, data):
        # display values from suscriber received in callback function
        rospy.loginfo(rospy.get_name() + _
        _ ": Get from Joy [%.2f, %.2f,%.2f,%.2f,%.2f,%.2f]" % data.axes)

        # initialize message msg
        msg = Twist()

        # store linear and angular values in msg
        msg.linear.x=data.axes[1]
        msg.angular.z=data.axes[0]

        # publish the message on /cmd_vel topic
        self.cmd_vel.publish(msg)

    def shutdown(self):
        # Always stop the /cmd_vel publishing when shutting down the node
        rospy.loginfo("Stopping the node ...")

        # publish 0 values to /cmd_vel
        self.cmd_vel.publish(Twist())

        # 1 second break
        rospy.sleep(1)


if __name__ == '__main__':
    try:
        NewJoy2Twist()
    except:
        rospy.loginfo("NewJoy2Twist node terminated.")

```