EE405A Robotics Operating System (ROS) - 2

(TA) Hyunki Seong
School of Electrical Engineering
KAIST

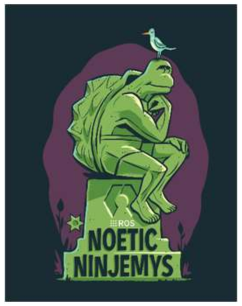
March 17, 2021

hynkis@kaist.ac.kr

Experiment Objectives

In this week, you will do the following:

- Understand how to use the ROS Tools (rviz, rosbag)
- Learn ROS Programming
- Programming Assignment :
  - Programming ROS topic publisher & subscriber.
  - Use ‘roslaunch’ to run your publisher & subscriber node together.



ROS Tools

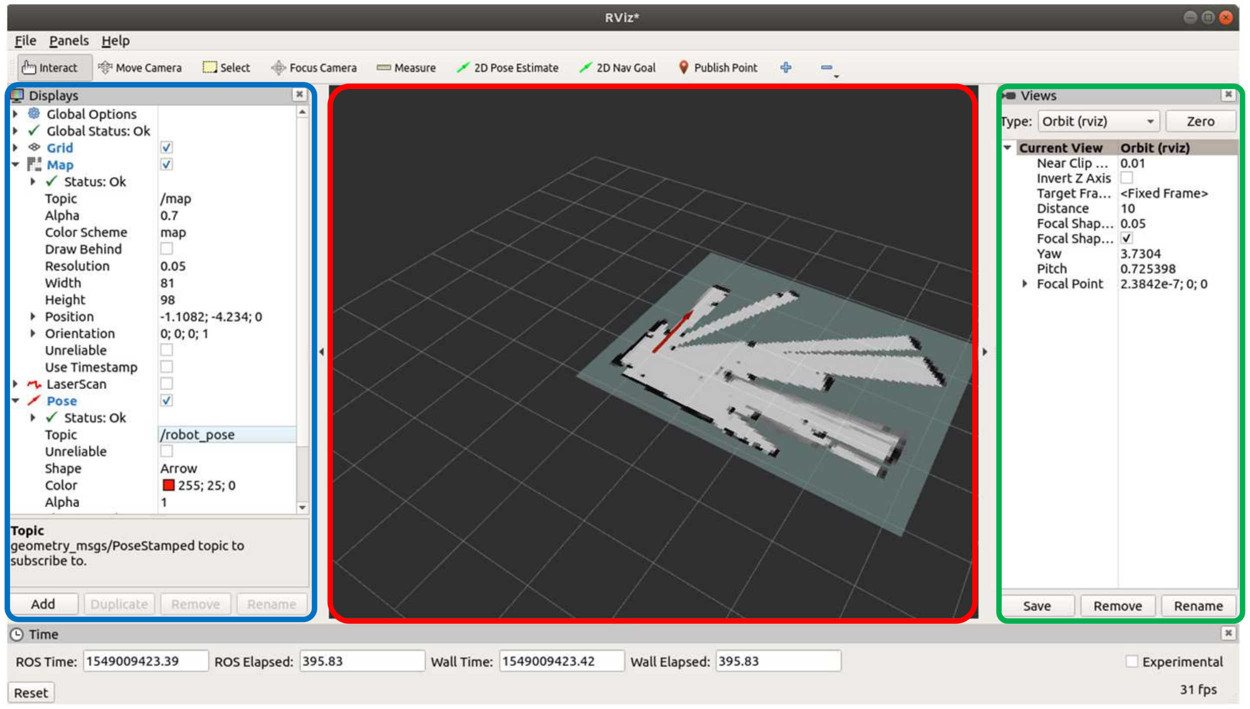
ROS Tools

- **Rviz**
  - Rvis is a **3D visualization tool** for ROS applications.
  - It provides a view of your robot model, capture sensor information from robot sensors, and replay captured data.
  - It can **display data** from camera, lasers, from 3D and 2D devices including pictures and point clouds.

- **Rosbag**
  - This is a set of tools for **recording from and playing back to ROS topics.**
  - The rosbag package provides a command-line tool for working with bags as well as code APIs for reading/writing bags in C++ and Python.
  - It can **record** a bag, **republish** the messages from one or more bags, summarize the contents of a bag, etc.

ROS Tools

- **Rviz**
  - ❑ Start command : **rosrun rviz rviz** (or simply, **rviz**)



Displays	
Global Options	
✓ Global Status: Ok	
Grid	✓
Map	✓
✓ Status: Ok	
Topic	/map
Alpha	0.7
Color Scheme	map
Draw Behind	☐
Resolution	0.05
Width	81
Height	98
Position	-1.1082; -4.234; 0
Orientation	0; 0; 0; 1
Unreliable	☐
Use Timestamp	☐
LaserScan	☐
Pose	✓
✓ Status: Ok	
Topic	/robot_pose
Unreliable	☐
Shape	Arrow
Color	🟥 255; 25; 0
Alpha	1

Topic
geometry_msgs/PoseStamped topic to subscribe to.

Type: Orbit (rviz)

Current View	Orbit (rviz)
Near Clip …	0.01
Invert Z Axis	☐
Target Fra…	<Fixed Frame>
Distance	10
Focal Shap…	0.05
Focal Shap…	✓
Yaw	3.7304
Pitch	0.725398
Focal Point	2.3842e-7; 0; 0

ROS Time: 1549009423.39 ROS Elapsed: 395.83 Wall Time: 1549009423.42 Wall Elapsed: 395.83

Displays
- **Add** messages to display.
- **Topic name** should be matched.
- **Toggle** message displaying on/off.

Views
- Change the view type.
- Go to initial zero point view.

3D View
- Display 3D data.
- Rotate and shift the 3D view.

ROS Tools

- **Rosbag**
  - **rosbag –h**

    : display the sub-commands for ‘rosbag’ command.
  - **rosbag record [topic_name_1] [topic_name_2] … [topic_name_ n]**

    : record a bag file with the contents of specific topics.

    i.e., rosbag record /image/raw /scan /imu /odom
  - **rosbag info [.bag file]**

    : summarize the contents of one or more bag files.

    i.e., rosbag info test_210312.bag
  - **rosbag play [.bag file]**

    : play back the contents of one of more bag files with time-synchronization.

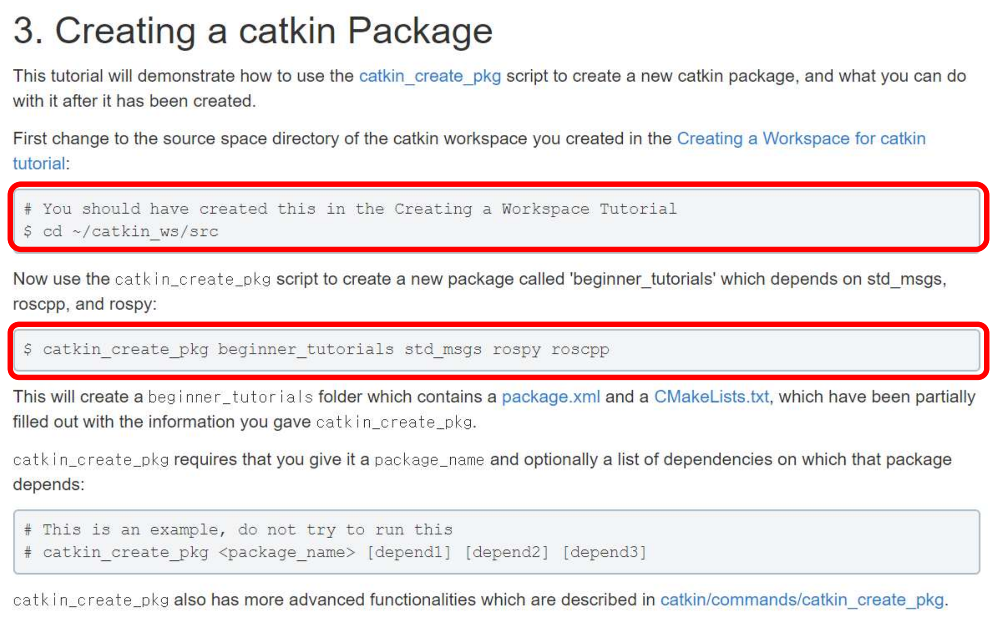
    i.e., rosbag play test_210312.bag

ROS Programming

ROS Programming

➢Creating a catkin package

Reference :
http://wiki.ros.org/catkin/Tutorials/CreatingPackage



3. Creating a catkin Package

This tutorial will demonstrate how to use the catkin_create_pkg script to create a new catkin package, and what you can do with it after it has been created.

First change to the source space directory of the catkin workspace you created in the Creating a Workspace for catkin tutorial:

```
# You should have created this in the Creating a Workspace Tutorial
$ cd ~/catkin_ws/src
```

Now use the catkin_create_pkg script to create a new package called 'beginner_tutorials' which depends on std_msgs, roscpp, and rospy:

```
$ catkin_create_pkg beginner_tutorials std_msgs rospy roscpp
```

This will create a beginner_tutorials folder which contains a package.xml and a CMakeLists.txt, which have been partially filled out with the information you gave catkin_create_pkg.

catkin_create_pkg requires that you give it a package_name and optionally a list of dependencies on which that package depends:

```
# This is an example, do not try to run this
# catkin_create_pkg <package_name> [depend1] [depend2] [depend3]
```

catkin_create_pkg also has more advanced functionalities which are described in catkin/commands/catkin_create_pkg.

➢ **Go to the catkin workspace directory**

cd ~/catkin_ws/src

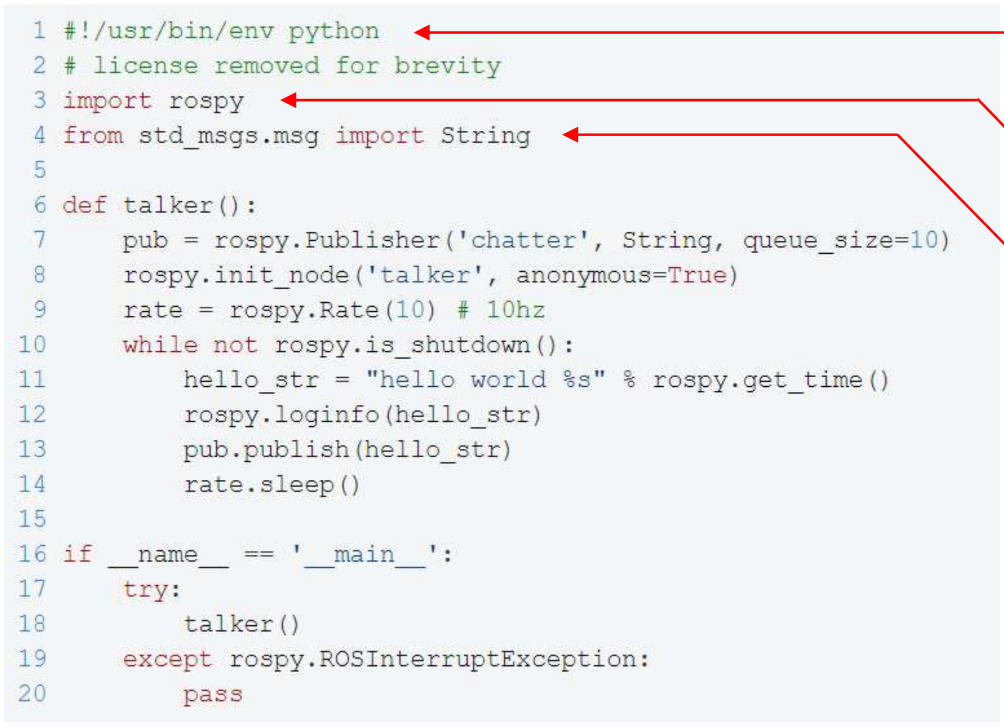
➢ **Create a ROS package**

catkin_create_pkg beginner_tutorial std_msgs rospy

※ catkin_create_pkg [package_name] [dependent] [dependent] …

ROS Programming

➢Writing a Publisher Node



```python
#!/usr/bin/env python
# license removed for brevity
import rospy
from std_msgs.msg import String

def talker():
    pub = rospy.Publisher('chatter', String, queue_size=10)
    rospy.init_node('talker', anonymous=True)
    rate = rospy.Rate(10) # 10hz
    while not rospy.is_shutdown():
        hello_str = "hello world %s" % rospy.get_time()
        rospy.loginfo(hello_str)
        pub.publish(hello_str)
        rate.sleep()

if __name__ == '__main__':
    try:
        talker()
    except rospy.ROSInterruptException:
        pass
```

Every Python ROS node will have this declaration at the top.
(This make sure your script is executed as a Python script.)

Import ‘rospy’ library

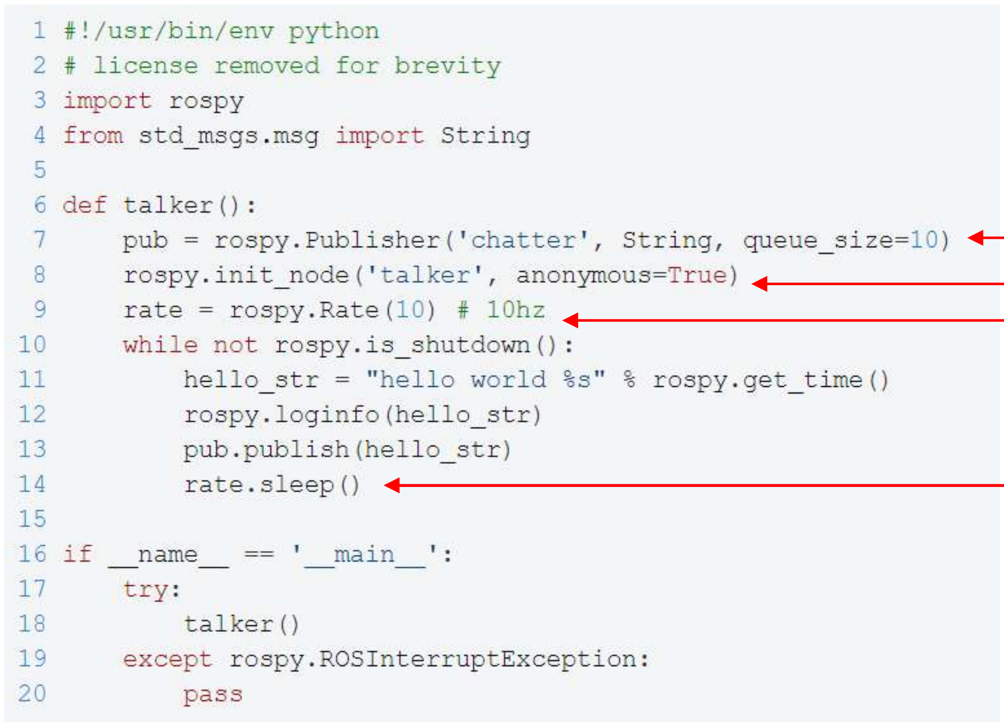
Import String message type

Reference : http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber%28python%29

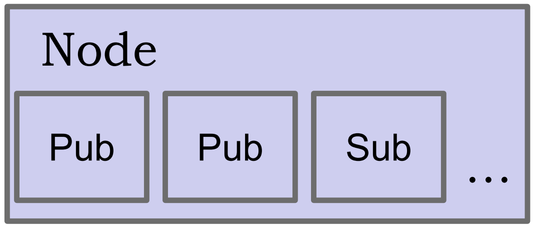
EE 405A

ROS Programming

➢**Writing a Publisher Node**



```python
#!/usr/bin/env python
# license removed for brevity
import rospy
from std_msgs.msg import String

def talker():
    pub = rospy.Publisher('chatter', String, queue_size=10)
    rospy.init_node('talker', anonymous=True)
    rate = rospy.Rate(10) # 10hz
    while not rospy.is_shutdown():
        hello_str = "hello world %s" % rospy.get_time()
        rospy.loginfo(hello_str)
        pub.publish(hello_str)
        rate.sleep()

if __name__ == '__main__':
    try:
        talker()
    except rospy.ROSInterruptException:
        pass
```



Define a publisher to publish a topic message 'chatter'

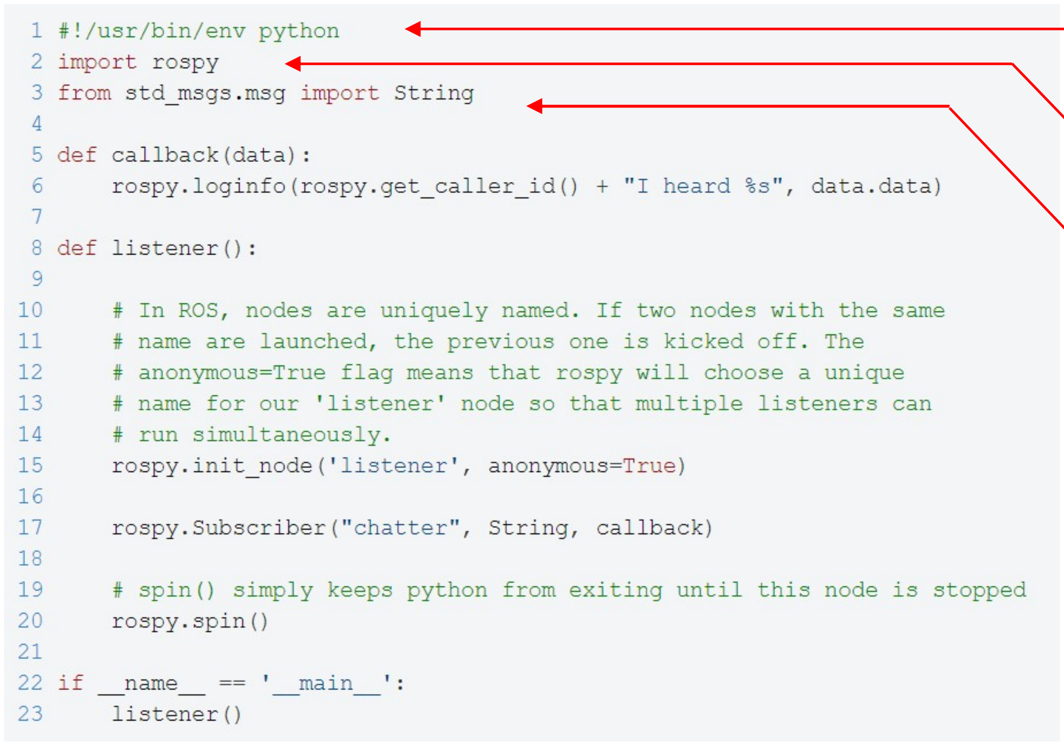
Define a node named 'talker'

Set node process rate

Reference : http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber%28python%29

ROS Programming

➢Writing a Subscriber Node



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

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

def listener():

    # 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('listener', anonymous=True)

    rospy.Subscriber("chatter", String, callback)

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

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

Every Python ROS node will have this declaration at the top. (This make sure your script is executed as a Python script.)

Import 'rospy' library

Import String message type

Reference : http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber%28python%29

ROS Programming

➢Writing a Subscriber Node

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

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

def listener():

    # 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('listener', anonymous=True)

    rospy.Subscriber("chatter", String, callback)

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

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

Define a node named ‘listener’

Define a subscriber to subscribe the topic message “chatter”

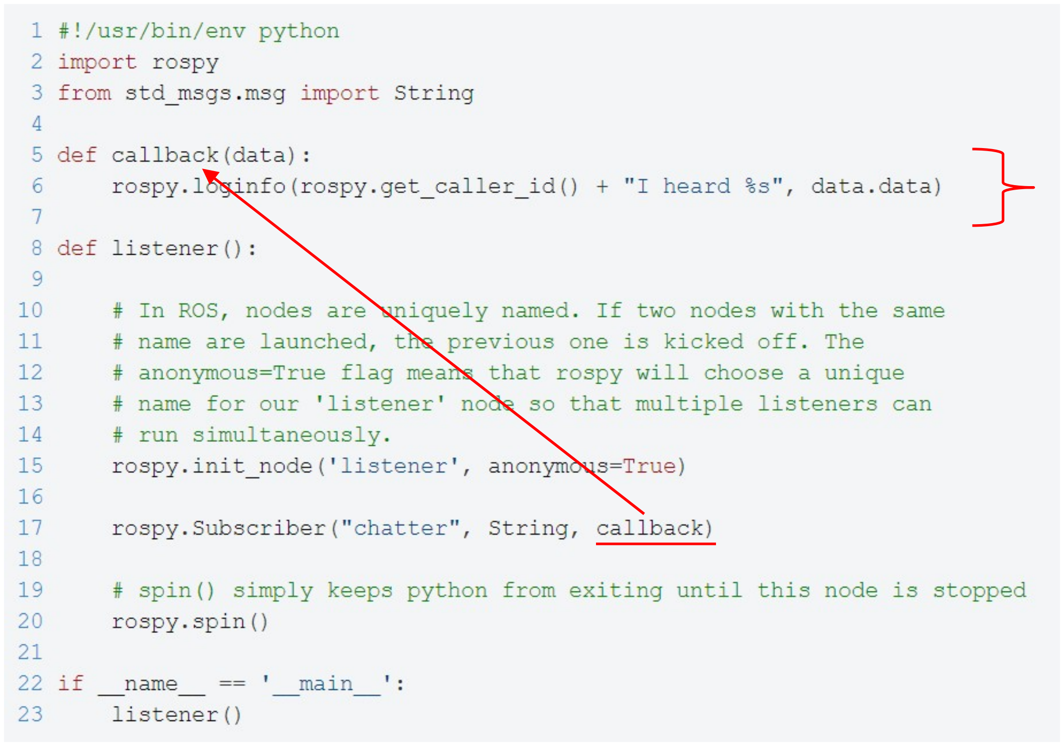
Keep the process until this node is stopped

You can change this to a **while loop with rate.sleep** similar with the ones in the Publisher node

Reference : http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber%28python%29

ROS Programming

➢Writing a Subscriber Node



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

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

def listener():

    # 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('listener', anonymous=True)

    rospy.Subscriber("chatter", String, callback)

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

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

Callback process to subscribe a topic message 'chatter'

A callback function is a function which is:
- passed as an **argument** to another **function**
- **is invoked** after some kind of **event**.

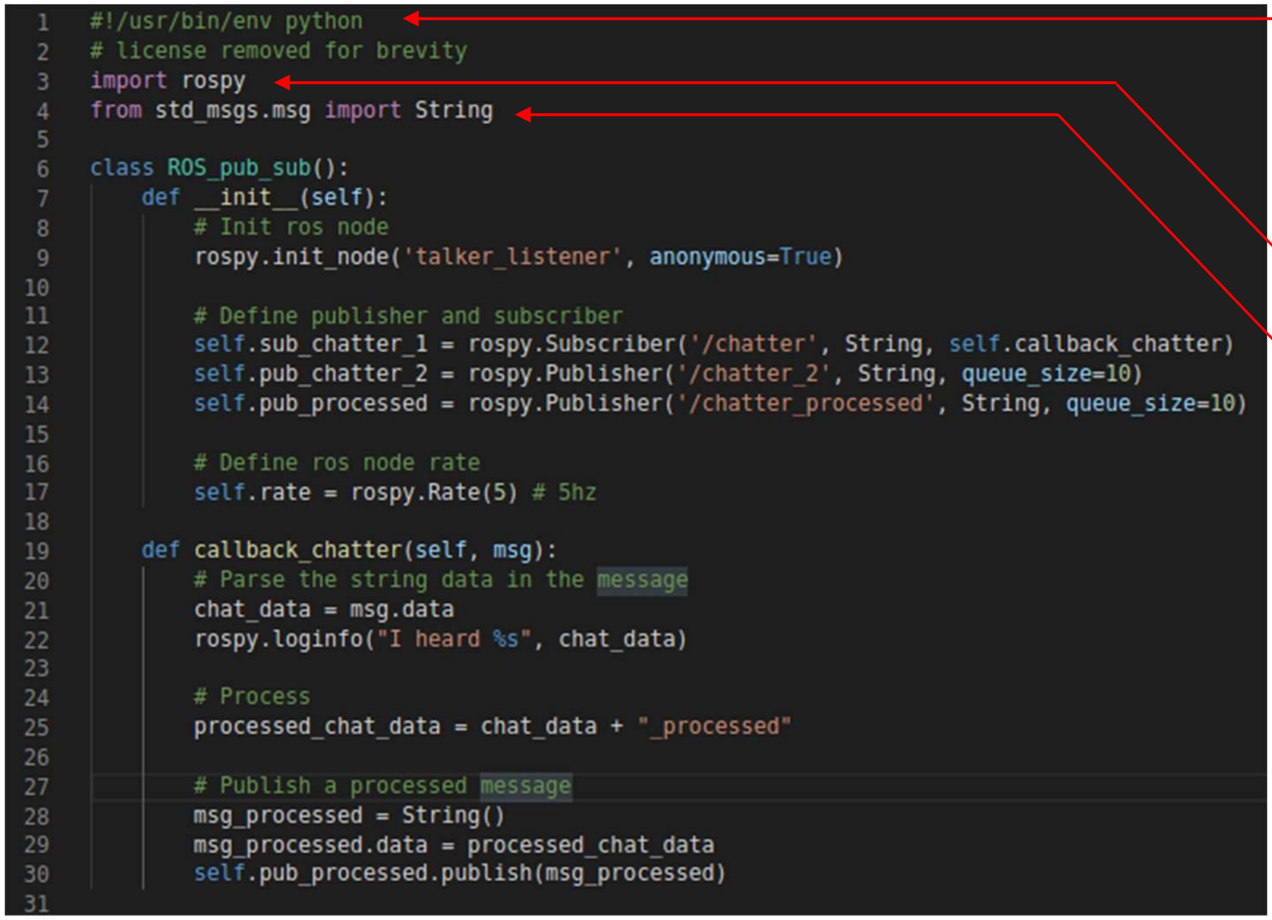
→
- passed a **topic message** to **callback function**
- **is invoked** after **receiving a message**

Reference : https://stackoverflow.com/questions/824234/what-is-a-callback-function

Reference : http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber%28python%29

ROS Programming

➢Writing a Publisher + Subscriber Node



```python
#!/usr/bin/env python
# license removed for brevity
import rospy
from std_msgs.msg import String

class ROS_pub_sub():
    def __init__(self):
        # Init ros node
        rospy.init_node('talker_listener', anonymous=True)

        # Define publisher and subscriber
        self.sub_chatter_1 = rospy.Subscriber('/chatter', String, self.callback_chatter)
        self.pub_chatter_2 = rospy.Publisher('/chatter_2', String, queue_size=10)
        self.pub_processed = rospy.Publisher('/chatter_processed', String, queue_size=10)

        # Define ros node rate
        self.rate = rospy.Rate(5) # 5hz

    def callback_chatter(self, msg):
        # Parse the string data in the message
        chat_data = msg.data
        rospy.loginfo("I heard %s", chat_data)

        # Process
        processed_chat_data = chat_data + "_processed"

        # Publish a processed message
        msg_processed = String()
        msg_processed.data = processed_chat_data
        self.pub_processed.publish(msg_processed)

```

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

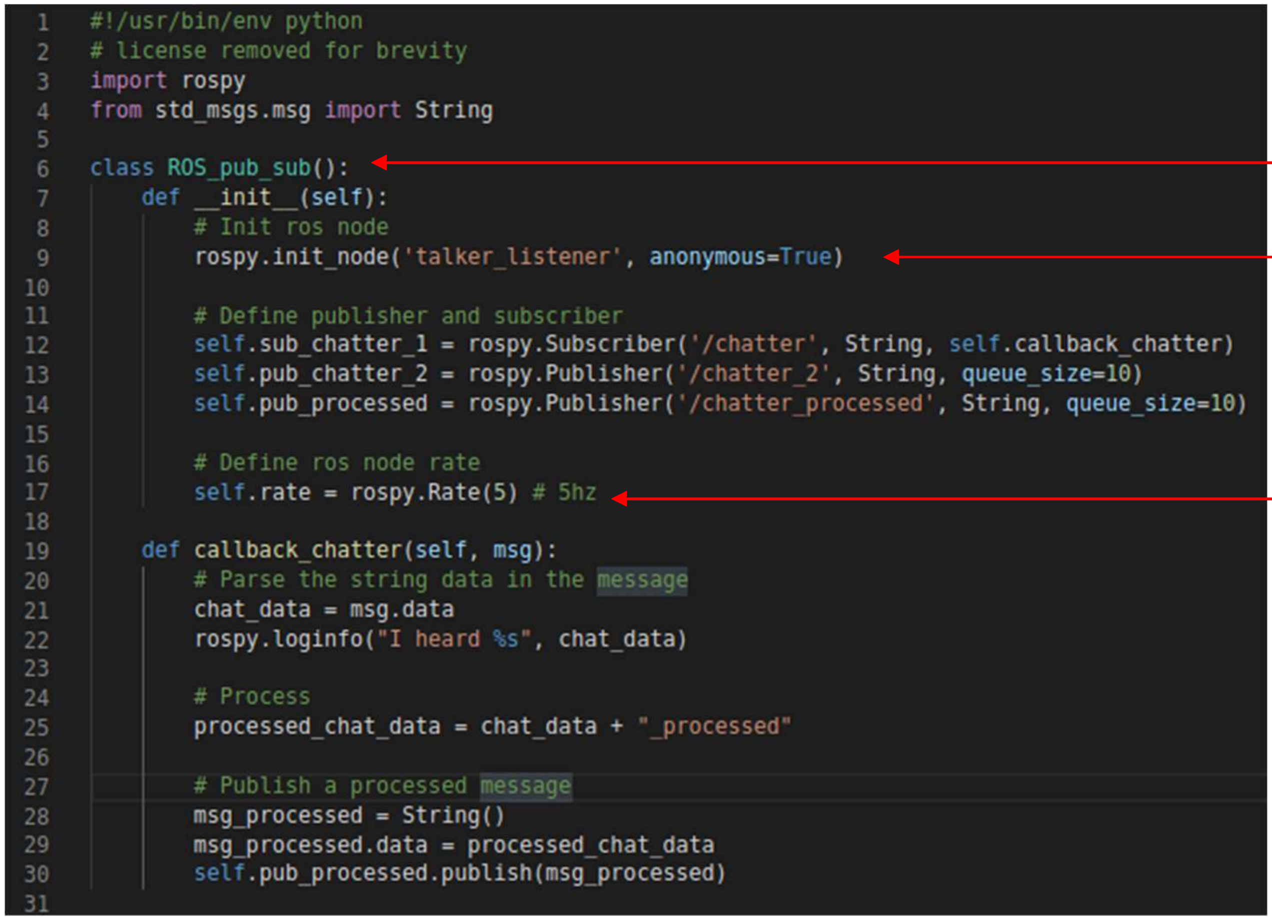
Import 'rospy' library

Import String message type

Reference code : https://github.com/hynkis/EE405A/blob/main/Week3/Materials/test_package/script/test_pub_sub.py

ROS Programming

➢Writing a Publisher + Subscriber Node



```python
#!/usr/bin/env python
# license removed for brevity
import rospy
from std_msgs.msg import String

class ROS_pub_sub():
    def __init__(self):
        # Init ros node
        rospy.init_node('talker_listener', anonymous=True)

        # Define publisher and subscriber
        self.sub_chatter_1 = rospy.Subscriber('/chatter', String, self.callback_chatter)
        self.pub_chatter_2 = rospy.Publisher('/chatter_2', String, queue_size=10)
        self.pub_processed = rospy.Publisher('/chatter_processed', String, queue_size=10)

        # Define ros node rate
        self.rate = rospy.Rate(5) # 5hz

    def callback_chatter(self, msg):
        # Parse the string data in the message
        chat_data = msg.data
        rospy.loginfo("I heard %s", chat_data)

        # Process
        processed_chat_data = chat_data + "_processed"

        # Publish a processed message
        msg_processed = String()
        msg_processed.data = processed_chat_data
        self.pub_processed.publish(msg_processed)

```

Create a class

Define a node named ‘talker_listener’

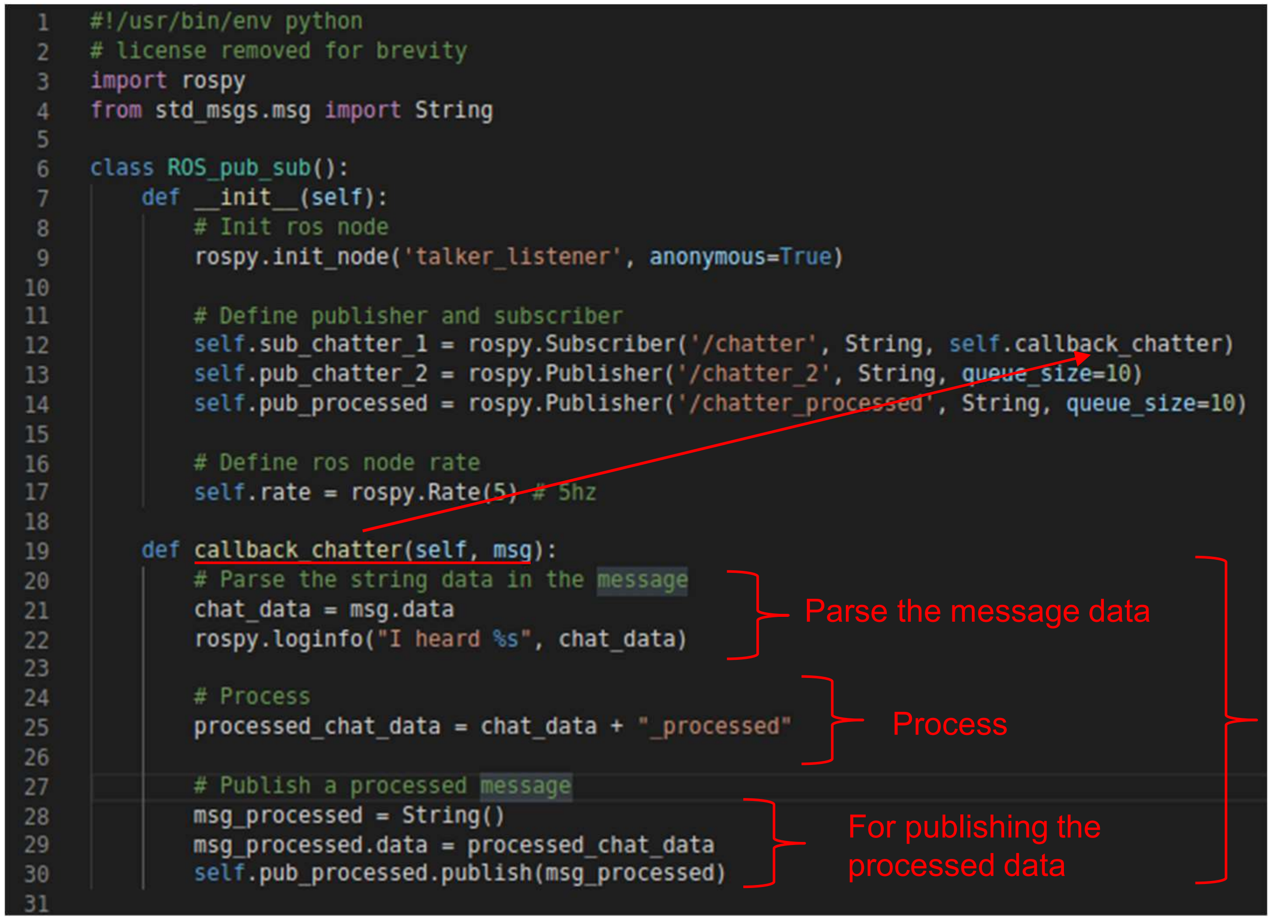
Define two publishers and a subscriber

Set node process rate

Reference code : **https://github.com/hynkis/EE405A/blob/main/Week3/Materials/test_package/script/test_pub_sub.py**

ROS Programming

➢Writing a Publisher + Subscriber Node



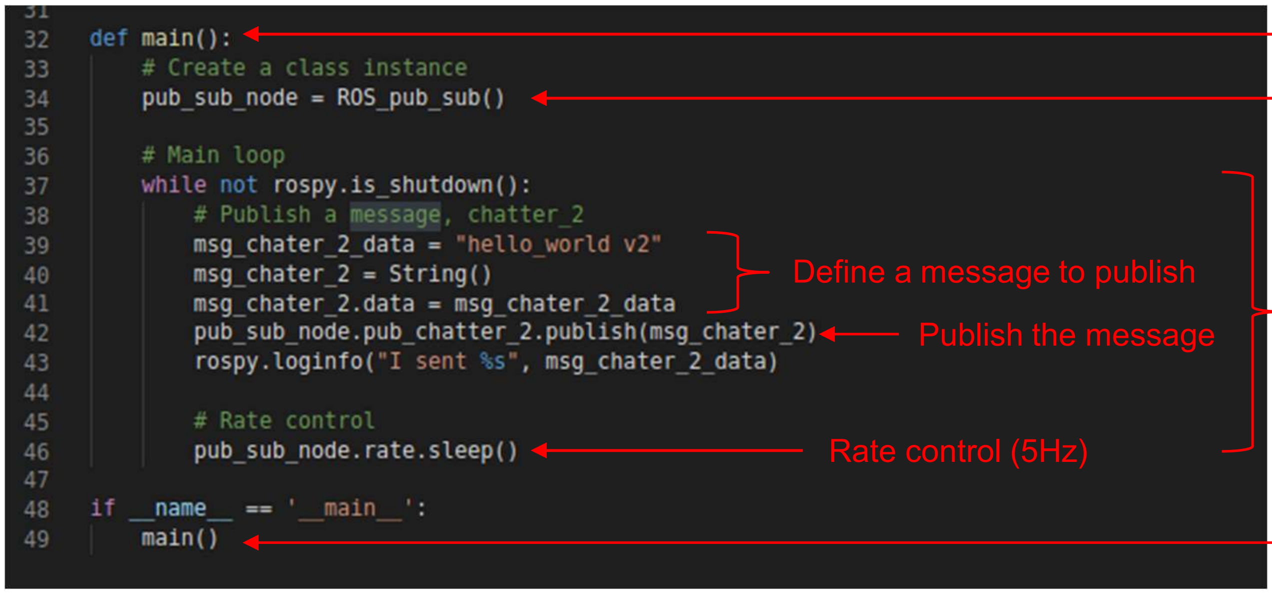
```python
#!/usr/bin/env python
# license removed for brevity
import rospy
from std_msgs.msg import String

class ROS_pub_sub():
    def __init__(self):
        # Init ros node
        rospy.init_node('talker_listener', anonymous=True)

        # Define publisher and subscriber
        self.sub_chatter_1 = rospy.Subscriber('/chatter', String, self.callback_chatter)
        self.pub_chatter_2 = rospy.Publisher('/chatter_2', String, queue_size=10)
        self.pub_processed = rospy.Publisher('/chatter_processed', String, queue_size=10)

        # Define ros node rate
        self.rate = rospy.Rate(5) # 5hz

    def callback_chatter(self, msg):
        # Parse the string data in the message
        chat_data = msg.data
        rospy.loginfo("I heard %s", chat_data)

        # Process
        processed_chat_data = chat_data + "_processed"

        # Publish a processed message
        msg_processed = String()
        msg_processed.data = processed_chat_data
        self.pub_processed.publish(msg_processed)
```

Callback process to subscribe to a topic message ‘**chatter**’ and publish a topic message ‘**chatter_processed**’

Reference code : https://github.com/hynkis/EE405A/blob/main/Week3/Materials/test_package/script/test_pub_sub.py

ROS Programming

➢Writing a Publisher + Subscriber Node



```python
def main():
    # Create a class instance
    pub_sub_node = ROS_pub_sub()

    # Main loop
    while not rospy.is_shutdown():
        # Publish a message, chatter_2
        msg_chater_2_data = "hello_world v2"
        msg_chater_2 = String()
        msg_chater_2.data = msg_chater_2_data
        pub_sub_node.pub_chatter_2.publish(msg_chater_2)
        rospy.loginfo("I sent %s", msg_chater_2_data)

        # Rate control
        pub_sub_node.rate.sleep()

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

Define a main function

Create a class instance

Define two publishers and a subscriber

Run the main function

Reference code : https://github.com/hynkis/EE405A/blob/main/Week3/Materials/test_package/script/test_pub_sub.py

ROS Programming

➢ **rosrun** / roslaunch

- ❑ **rosrun** command for running a ROS node.
- ❑ Use ‘rosrun’ command for each ROS node.
  - ✓ rosrun test_package test_publisher.py
  - ✓ rosrun test_package test_subscriber.py
  - ✓ rosrun test_package test_pub_sub.py

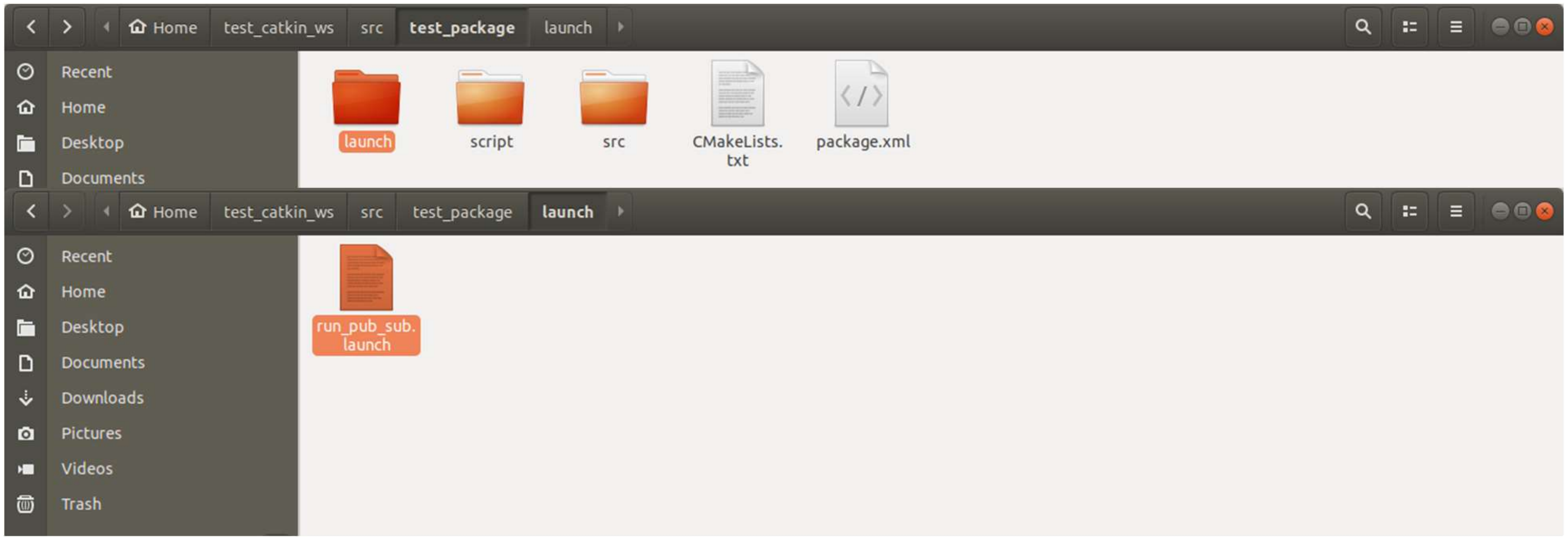
  ※ rosrun [package_name] [node_name]

Reference for ‘rosrun’ : http://wiki.ros.org/ROS/Tutorials/UnderstandingNodes

ROS Programming

➢ rosrun / **roslaunch**

- ❑ **roslaunch** command for **running multiple ros nodes** at once.
- ❑ Make a **'launch' directory** in your package folder.
- ❑ Create a **'.launch' script** in the launch folder.

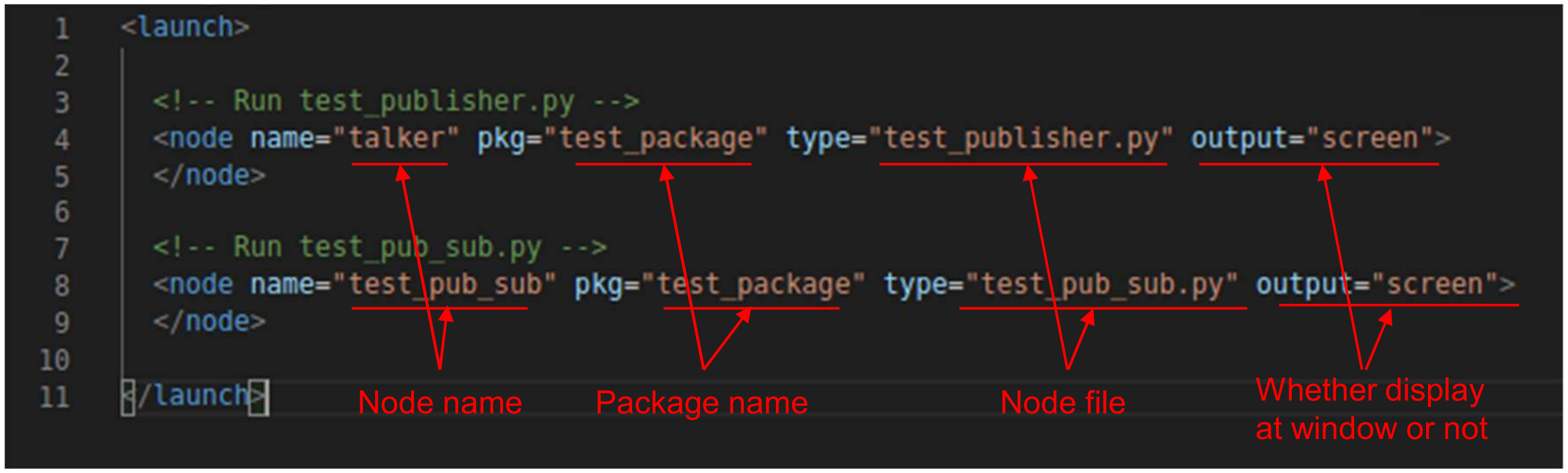


Reference for 'roslaunch' : http://wiki.ros.org/roslaunch

ROS Programming

➢ rosrun / **roslaunch**

❑ **You need to write a .launch script for using the 'roslaunch' command.**



```
<launch>

  <!-- Run test_publisher.py -->
  <node name="talker" pkg="test_package" type="test_publisher.py" output="screen">
  </node>

  <!-- Run test_pub_sub.py -->
  <node name="test_pub_sub" pkg="test_package" type="test_pub_sub.py" output="screen">
  </node>

</launch>
```

❑ **Use 'roslaunch' command for running multiple ROS nodes.**

✓ roslaunch test_package run_pub_sub.launch

※ roslaunch [package_name] [launch_file]

Reference for 'roslaunch' : http://wiki.ros.org/roslaunch

Reference launch script :
https://github.com/hynkis/EE405A/blob/main/Week3/Materials/test_package/launch/run_pub_sub.launch

References for ROS Tutorial

- **Official ROS Tutorials**

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

- **Programming for Robotics (ROS) Course**
  - ❑ Youtube videos for ROS introductions.

  https://www.youtube.com/watch?v=0BxVPCInS3M&list=PLE-BQwvVGf8HOvwXPgtDfWoxd4Cc6ghiP

- **The Construct: A Platform to Learn ROS-based Advanced Robotics Online**
  - ❑ A linux VM-based MOOC platform.
  - ❑ Several courses are not free.

  https://www.theconstructsim.com/

- **Hello (Real) World with ROS – Robot Operating System**
  - ❑ A MOOC course for ROS in Edx.
  - ❑ You can take the course for free by accessing to the audit track.

  https://www.edx.org/course/hello-real-world-with-ros-robot-operating-system

Programming Assignment

Programming Assignment

- **Create a ROS package with following functions:**
  - ❑ **Create a ROS package**
  - ❑ **Write a ROS publisher node (‘fake_sensor.py’)**
    - ✓ Publish a fake sensor data whose
      - topic name is ‘/vehicle_state’
      - message type is std_msgs/Float32 (You can set the data value arbitrary.)
      - rate is 30Hz
  - ❑ **Write a ROS subscriber node (‘data_processor.py’)**
    - ✓ Subscribe to the fake sensor data
    - ✓ Using the received sensor data, publish a processed data whose
      - topic name is “/processed_state”
      - message type is std_msgs/Float32 (You can set the processed data arbitrary.)
  - ❑ **Run both publisher and subscriber nodes using ‘roslaunch’**
    - ✓ Create a .launch script to run the publisher(‘fake_sensor.py’) and subscriber node(‘data_processor.py’).

Programming Assignment

- **Send followings to hynkis@kaist.ac.kr until 21.03.31 (for 2 weeks)**
  - ❑ Your **ROS package**
  - ❑ Your **Report**
    - ✓ Write **what you have learned** this week.
    - ✓ You can use both **KOR/ENG** in your report.

  - ❑ Please **zip your ROS package and Report** with the following filename.

    EE405A_[lecture_date(YYMMDD)]_[Student ID]_[Full name]

    (e.g., EE405A_210317_20215169_Hyunki_Seong.zip)

Reference for the assignment :
https://github.com/hynkis/EE405A/tree/main/Week3/Assignments

Experiment Summaries

- **Understand how to use the ROS Tools (rviz, rosbag).**
  - **rvis** is a **3D visualization tool** for ROS applications.
  - **rosbag** is a set of tools for **recording** from and **playing back** to ROS topics.

- **Learn ROS Programming.**
  - Creating a ROS package.
  - Writing a Publisher node.
  - Writing a Subscriber node.
  - 'rosrun' command for running a ROS node.
  - 'roslaunch' command for running multiple ROS nodes at once.

Experiment Objectives

Next week, you will do the following:

- Understand the type of ROS topic message
- Learn frequently-used message types

  (e.g., std_msgs, geometry_msgs, nav_msgs, …)
- Learn how to create a custom message

Q & A

Email : hynkis@kaist.ac.kr