ECE 5463 Introduction to Robotics
Spring 2018

ROS
TUTORIAL 1

Guillermo Castillo (Wei Zhang)
Department of Electrical and Computer Engineering
Ohio State University

01/2018

Outline

- **Previous Steps**
  - Installing VM, Ubuntu 16.04.3, ROS Kinetic Distribution
  - Possible problems
- **ROS**
  - Understanding ROS Basics
  - ROS Packages
    - Creating ROS workspace and ROS Package
    - Understanding ROS Filesystem
- **ROS Program**
  - First ROS Program – Hello world
  - Python scripts, launch files.
  - Turtlesim simulation

Possible problems

- The ROS ENVIRONMENT VARIABLE is not defined properly.

```
guillermo@guillermo-pc:~$ roslaunch tutorial tutorial_launcher.launch
[tutorial_launcher.launch] is neither a launch file in package [tutorial] nor is
 [tutorial] a launch file name
The traceback for the exception was written to the log file
guillermo@guillermo-pc:~$
```

- Solution

  Run the following commands:

```
echo "source ~/catkin_ws/devel/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

Possible problems

- Installation of Anaconda changes the default path for python.
- ROS programs may not run.
- Solution

  Open the bashrc file with the following command:

  ```
  gedit ~/.bashrc
  ```

  Erase or comment the line

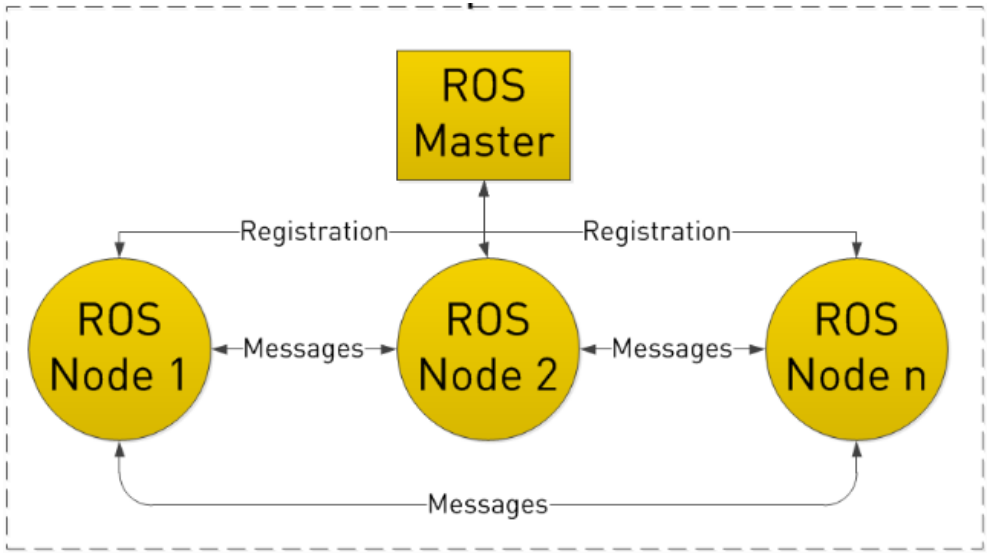
```
export PATH="/home/guillermo/anaconda3/bin:$PATH"
```

- Alternative solution

  Every time you open a new terminal run the following command:

  ```
  export $PATH="/usr/bin:$PATH„
  ```

General Overview



What is a package?

- All the files that a specific ROS program contains; all its cpp files, python files, configuration files, compilation files, launch files, and parameters files.

- Generally all those files in the package are organized with the following structure:
  - **launch** folder: Contains launch files
  - **src** folder: Source files (cpp, python)

  } Not always

  - **CMakeLists.txt**: List of cmake rules for compilation
  - **package.xml**: Package information and dependencies

Creating ROS package

- Every new project should be organized in packages
- We need to work in a very specific ROS workspace, which is known as the *catkin workspace. (catkin_ws)*
- Associate the name of the package with its functionality.

CREATING ROS WORKSPACE (Only one time)

- The default directory for ROS packages is the path: /opt/ros/kinetic/share/
- Verify it with the command: `printenv | grep ROS`
- Directory path for new ROS projects: ~/catkin_ws/src (~ = home)

```
mkdir -p ~/catkin_ws/src
cd ~/catkin_ws/
catkin_make      ---    Build the files in the workspace
```

Creating ROS package

• Always create new packages inside the SRC folder of the catkin workspace.

```
cd ~/catkin_ws/src
catkin_create_pkg <package_name> <package_dependecies>
```

• The **package_name** is the name of the package you want to create, and the **package_dependencies** are the names of other ROS packages that your package depends on.

```
catkin_create_pkg tutorial rospy
```

• The launch and src folder are not always created automatically, but we can create them manually.

```
mkdir ~/catkin_ws/src/tutorial/launch
```

• Package commands:

rospack list: Gives a list of all the packages in your ROS system.

rospack list | grep [my_package]: Filters packages that contain [my_package]

ro**scd [my_package]:** Takes you to the location of [my_package]

Compile a package

• When you create a package, you will usually need to compile it in order to make it work.

• This command will compile your whole src directory, and it needs to be called in your *catkin_ws* directory. If you try to compile from another directory, it won't work.

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

Do not forget!

• Build the package when you install or create a new one, and

• Source the workspace so that ROS can find the new package.

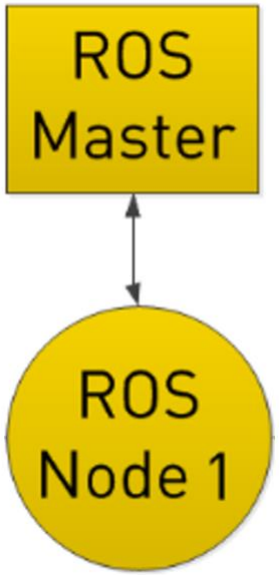
First ROS program – Hello world

- **Goal**: Create the simplest ROS program, to print “HELLO WORLD”
- Therefore, we need some “object/element” that performs the “action” of printing the desired message on the screen.
- **Answer:** Node = small program that performs an action.
- Then, we will create a ROS node that print the message on the screen. Usually this nodes are initiated by a python or cpp script.
- **Good practice:** name the script according its function (node/action).

```
cd ~/catkin_ws/src/tutorial/src
gedit printer.py
```

Million dollar question

- What is the structure of the program?



- If you can visualize the structure, you can create it!

Python script

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

# The above line will ensure the interpreter used is the first one on your
# environment's $PATH. Every Python file needs to start with this line at the top.

import rospy # Import the rospy, which is a Python library for ROS.

rospy.init_node('printer_node')    # Initiate a node called printer_node
print "\n\nHELLO WORLD - ROS TUTORIAL\n\n" # A simple Python print
```

• **NOTE:** If you create your Python file from the shell, it may happen that it is created without execution permissions. If this happens, ROS will not be able to find it. If this is your case, you can give execution permissions to the file by typing the next command: *chmod +x name_of_the_file.py*

```
chmod +x printer.py
```

Running the script

- Initiate *ROS MASTER*

```
roscore
```

- **Open another terminal's window** (CTRL + SHIFT + T)
- Run the python script using the command: *rosrun [package_name] [python_file_name]*

```
rosrun tutorial printer.py
```

- **Stop the program** (CTRL + C)

Launch file

• All launch files are contained within a <launch> tag. Inside that tag, you can see a <node> tag, where we specify the following parameters:

1. pkg="package_name" # Name of the package that contains the code
2. type="python_file_name.py" # Name of the program file to execute
3. name="node_name" # Name of the ROS node that will launch our .py file
4. output="type_of_output" # Through which channel you will print the output of the .py file

```
cd ~/catkin_ws/src/tutorial/launch
gedit node_launcher.launch
```

```
<launch>
  <node pkg="tutorial" type="printer.py" name="printer_node"
output="screen">
  </node>
</launch>
```

Running the launch file

- Use ROS command: *roslaunch [package_name] [launch_file.launch]*

```
roslaunch tutorial node_launcher.launch
```

- Now, let's check what nodes are actually running using the command:

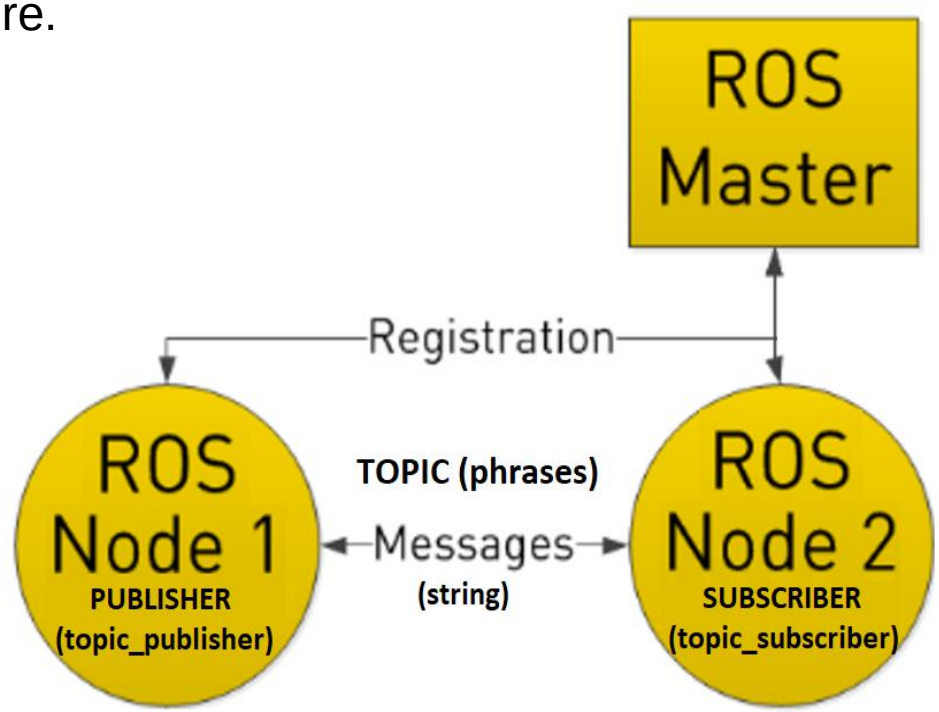
```
rosnode list
```

- The node is killed when the Python program ends.

Bigger ROS program structure

- Same program with bigger structure.

- **GOAL:** Create a ROS program with two nodes. One of them must publish a series of messages to a defined topic. The second node must listen to those messages (subscribe to the topic) and print the messages on the screen.



Publisher and Topics

• A **publisher** *is a node* that keeps publishing a message into a topic.

• A **topic** is a channel that acts as an information high way, where other ROS nodes can either publish or read information.

CREATE A PUBLISHER

```
cd ~/catkin_ws/src/tutorial/src
gedit publisher.py
```

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

import rospy                                # Import the Python library for ROS
from std_msgs.msg import String             # Import the String message from the std_msgs package

rospy.init_node('topic_publisher')          # Initiate a Node named 'topic_publisher'
pub = rospy.Publisher('phrases', String, queue_size=10)    # Create a Publisher object, that will
                                                             publish on the "phrases" topic
                                            #  messages of type String

rate = rospy.Rate(2)                        # Set a publish rate of 2 Hz
msg_str = String()                          # Create a var of type String
msg_str = "HELLO WORLD - ROS TUTORIAL"      # Initialize 'msg_str' variable


while not rospy.is_shutdown():              # Create a loop that will go until someone stops the
                                              program execution
  pub.publish(msg_str)
  rate.sleep()                              # Make sure the publish rate maintains at 2 Hz
```

Useful ROS commands related to Topics

• To verify if the topic was actually created

```
rostopic list
```

• To “listen” what the publisher is “talking”. Namely, to observe the content of the topic: *rostopic echo [name_of_the_topic]*

```
rostopic echo /phrases
```

• To get information about the topic such as the message type, or the topic’s publishers and subscribers: *rostopic echo [name_of_the_topic]*

```
rostopic info /phrases
```

Messages

• Topics handle information through messages.

• In the case of the code we executed, the message type was an std_msgs/String, but ROS provides a lot of different messages.

•You can even create your own messages, but it is recommended to use ROS default messages when possible.

• To get information about a message, use the next command: *rosmsg show [message_type]*

```
rosmsg show std_msgs/String
```

• In this case, the String message has only one variable named data of type string. This String message comes from the package std_msgs.

Subscriber

• A **subscriber** *is a node* that reads information from a topic.

CREATE A SUBSCRIBER

```
cd ~/catkin_ws/src/tutorial/src
gedit subscriber.py
```

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

import rospy
from std_msgs.msg import String

def callback(msg):                                          # Define a function called 'callback' that
                                                              receives a parameter named 'msg'
    print msg.data                                          # Print the value 'data' inside the 'msg'
                                                              parameter
rospy.init_node('topic_subscriber')                         # Initiate a Node called 'topic_subscriber'
sub = rospy.Subscriber('/phrases', String, callback)        # Create a Subscriber object that will listen
                                                              to the "phrases" topic and will call the
                                                              'callback' function each time it reads
                                                              something from the topic
rospy.spin()                                                # Create a loop that will keep the program in
                                                              execution
```

Running the ROS program

• **Recall:** To run the scripts you have two ways. Using the **rosrun** command (Python file), or using the **roslaunch** command (launch file).

• Using the rosrun command:

```
rosrun tutorial suscriber.py
```

• Using launch file

```
roslaunch tutorial subscriber_launcher.launch
```

```
<launch>
  <node pkg="tutorial" type="subscriber.py"
name="topic_subscriber" output="screen">
  </node>
</launch>
```

• Notice that the subscriber will print nothing if there is no information published in the topic *“phrases”*

Alternative ways to publish and launch

- To publish directly in any topic: *rostopic pub [topic] [msg_type] [args]*

```
rostopic pub /phrases std_msgs/String “HELLO ROS”
```

- You can also launch more than one package at the same time from the same launch file.

```
<launch>
  <node pkg="tutorial" type="publisher.py" name="topic_publisher"
output="screen">
  </node>
  <node pkg="tutorial" type="subscriber.py"
name="topic_subscriber" output="screen">
  </node>
</launch>
```

```
roslaunch tutorial publisher_subscriber_launcher.launch
```

Important

- RECALL: **rosrun** is used to execute Python, cpp files, which initiate the nodes and perform actions, while **roslaunch** is used to execute the .launch file that can run one or more nodes at the same time. (This automate the above process)
- **Important commands summary**
  - rosnode list
  - rostopic list
  - rostopic info [package_name]
  - rostopic echo [package_name]
  - rosmsg show [message_type]

Turtlesim Simulation

• Start simulation by running nodes (this package uses cpp files to initiate the nodes). Run the following commands in different terminal's windows.

```
roscore
rosrun turtlesim turtlesim_node
rosrun turtlesim turtle_teleop_key
```

• Use the command learned in this tutorial to get information about the nodes, topics, messages.

```
rosnode list
rostopic list
rostopic info /turtle1/pose
rosmsg show /turtlesim/pose
rostopic echo /turtle1/cmd_vel
```

•Publish manually in *cmd_vel* topic

```
rostopic pub /turtle1/cmd_vel geometry_msgs/Twist  -- '[0.0, 0.0, 0.0]' '[0.0, 0.0, 1.6]'
```

References

- http://wiki.ros.org/urdf/Tutorials/
- http://wiki.ros.org/ROS/Tutorials/UnderstandingTopics
- http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber%28python%29
- http://wiki.ros.org/turtlesim/Tutorials
- https://www.robotigniteacademy.com
- O'Kane, Jason. A Gentle Introduction to ROS

Thanks for your attention