

SC-635 Advanced Topics in Mobile Robotics

Experiment Module : Introduction to ROS

January 17, 2020



Systems and Control Engineering
Indian Institute of Technology Bombay

Overview

1. ROS : About
2. Workspace settings
3. ROS Concepts and Commands

Supported robots

Exhaustive list of supported robots is available at:

<https://robots.ros.org/>

What is ROS

Open source framework that facilitates

- ▶ Running multiple small programs (nodes) concurrently
- ▶ Establish p2p communication between these programs (nodes)

What is ROS

Open source framework that facilitates

- ▶ Running multiple small programs (nodes) concurrently
- ▶ Establish p2p communication between these programs (nodes)

Interfaced to many useful softwares such as

- ▶ Gazebo : Physics simulation
- ▶ RViz : 3D robot visualization
- ▶ rqt : Framework for graphical UIX with ROS

What is ROS

Open source framework that facilitates

- ▶ Running multiple small programs (nodes) concurrently
- ▶ Establish p2p communication between these programs (nodes)

Interfaced to many useful softwares such as

- ▶ Gazebo : Physics simulation
- ▶ RViz : 3D robot visualization
- ▶ rqt : Framework for graphical UIX with ROS

Why it is wildly popular

- ▶ Robotics specific algorithms available as ROS package
- ▶ Allows write once and run everywhere (TurtleBot, PR2, Husky, AR Drone)
- ▶ Out of the box heterogeneous computing

ROS Popularity

Google Scholar

ROS

Articles About 32,30,000 results (0.03 sec)

Any time
Since 2020
Since 2019
Since 2016
Custom range...

[\[PDF\] ROS: an open-source Robot Operating System](#)
[M Quigley](#), [K Conley](#), [B Gerkey](#), [J Faust...](#) - ICRA workshop on ..., 2009 - willowgarage.com
This paper gives an overview of **ROS**, an opensource robot operating system. **ROS** is not an operating system in the traditional sense of process management and scheduling; rather, it provides a structured communications layer above the host operating systems of a ...

☆ ⓘ Cited by 6561 Related articles All 18 versions ⌕

6561 Citations of the original ROS paper¹

¹ source scholar.google.com

Binary Downloads by rosdistro

The fraction of packages downloaded per rosdistro

- Hydro and earlier: 0.00 %
- Indigo: 5.02 % (Long Term Support Release)
- Jade: 0.00 %
- Kinetic: 53.06 % (Long Term Support Release)
- Lunar: 0.85 %
- Melodic: 26.71 % (Long Term Support Release)
- Bouncy and earlier ROS 2: 0.12 %
- Crystal 0.62 %
- Dashing 1.51 % (Long Term Support Release)
- Rosdistro independent: 12.10% (packages like python-roscpp and python-roslaunch, as well as backported 3rdparty libraries like pcl and colladadom)

Source: Apache2, counting only downloads from the main repository (not the testing repository, shadow-fixed, or mirrors)

2

ROS workspace

- ▶ **Workspace**

Directory where ROS project/code is located. Can work with only one at a time.

ROS workspace

- ▶ Workspace

Directory where ROS project/code is located. Can work with only one at a time.

- ▶ Creating a workspace

```
$ catkin_init_workspace
```

```
$ catkin_make      consequences ..  
                  ../devel/  
                  ../build/
```

ROS workspace

- ▶ Workspace

Directory where ROS project/code is located. Can work with only one at a time.

- ▶ Creating a workspace

```
$ catkin_init_workspace
```

```
$ catkin_make           consequences ..  
                        ../devel/  
                        ../build/
```

- ▶ Activating a workspace

```
$ source devel/setup.bash
```

Roscore, Nodes, Topics, and Messages

Roscore

- ▶ The program that needs to run first
- ▶ Node discovery, topic discovery and other book-keeping task

Node

- ▶ A program with independent existence
- ▶ Can subscribe to a **topic** and receive a specific type of **messages**
- ▶ Can publish **message** to a **topic**

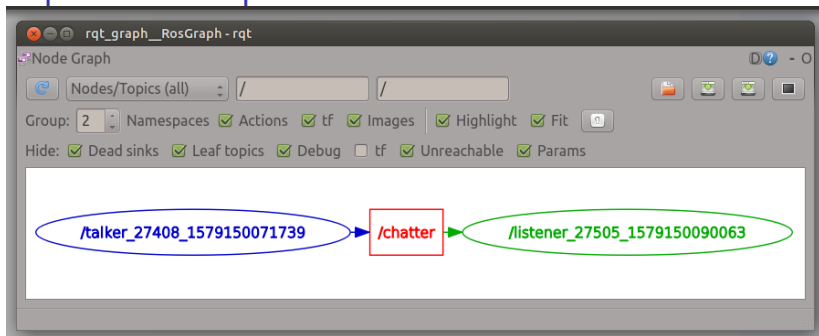
Topic

- ▶ Name for stream of messages. Nodes do not talk directly to each other, they communicate with Topics

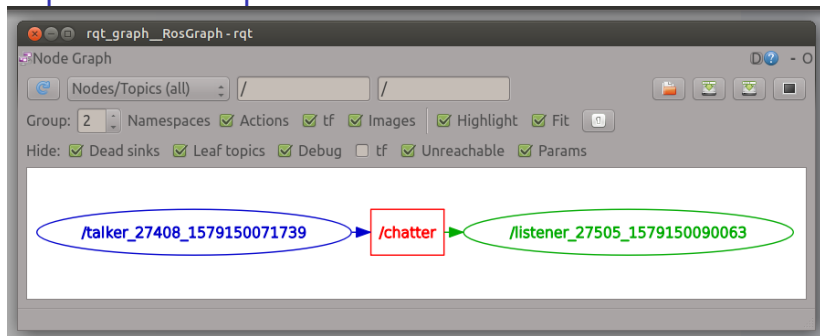
Message

- ▶ ROS messages are formatted data (akin to datatypes in programming languages)

A simple ROS-Graph



A simple ROS-Graph



Execute each command in a new terminal window/tab:

```
$ roscore
```

```
$ rosrun rospy_tutorials talker.py
```

```
$ rosrun rospy_tutorials listener.py
```

```
$ rqt_graph
```

Tip: Use **Ctrl+C** to stop the process (nodes)

Node 1: talker.py

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

Node 2: listener.py

```
1  #!/usr/bin/env python
2  import rospy
3  from std_msgs.msg import String
4
5  def callback(data):
6      rospy.loginfo(rospy.get_caller_id() + 'I heard %s',
7                      data.data)
8
9  def listener():
10     rospy.init_node('listener', anonymous=True)
11
12     rospy.Subscriber('chatter', String, callback)
13
14     # spin() simply keeps python from exiting until
15     this node is stopped
16     rospy.spin()
17
18 if __name__ == '__main__':
19     listener()
```


Creating a package

- ▶ Inside the `catkin_ws/src/` directory
\$ `catkin_create_project test_pub_sub rospy`
- ▶ Create a **script** directory inside the newly created `test_pub_sub` directory
\$ `mkdir scripts`
- ▶ Inside the **scripts** directory create two files `talker.py` and `listener.py`
\$ `touch talker.py listener.py`
- ▶ Give execute permissions to both the files
\$ `chmod +x talker.py listener.py`
- ▶ Add the code to the files
`bit.ly/2tdpTalk`
`bit.ly/2tdpListen`
- ▶ Go to `catkin_ws` directory and run
\$ `catkin_make`

Directory structure so far

```
└─ catkin_ws
   └─ build
   └─ devel
   └─ src
      └─ CMakeLists.txt
      └─ test_pub_sub
         └─ CMakeLists.txt
         └─ package.xml
         └─ scripts
            └─ listener.py
            └─ talker.py
         └─ src
```

Run the nodes

Execute each command in a new terminal window/tab:

```
$ roscore
```

```
$ rosrun test_pub_sub talker.py
```

```
$ rosrun test_pub_sub listener.py
```

```
$ rqt_graph
```

Run the nodes

Execute each command in a new terminal window/tab:

```
$ roscore
```

```
$ rosrun test_pub_sub talker.py
```

```
$ rosrun test_pub_sub listener.py
```

```
$ rqt_graph
```

Before running the nodes we need to **activate** the current workspace.

Go to `catkin_make` directory and run the following command

```
$ source devel/setup.bash
```

Frequently used ROS Commands

- ▶ `roscd`
- ▶ `rostopic`
- ▶ `roscd`
- ▶ `roslaunch`
- ▶ `roslaunch`

Launch file

- ▶ Create a **launch** directory inside `test_pub_sub` directory
\$ `mkdir launch`
- ▶ Inside **launch** directory create a **pubsub.launch** file
\$ `touch pubsub.launch`
- ▶ Add following code to the file
bit.ly/2tdpLaunchBasic
- ▶ Now we can start both the nodes by invoking the launch file.
Before that we will build the project
- ▶ Go to `catkin_ws` directory and run
\$ `catkin_make`
- ▶ Start the launch file with
\$ `roslaunch test_pub_sub pubsub.launch`

Launch file

```
1 <launch>
2   <node name="talker" pkg="test_pub_sub" type="talker
   .py" output="screen" />
3   <node name="listener" pkg="test_pub_sub" type="
   listener.py" output="screen" />
4 </launch>
```

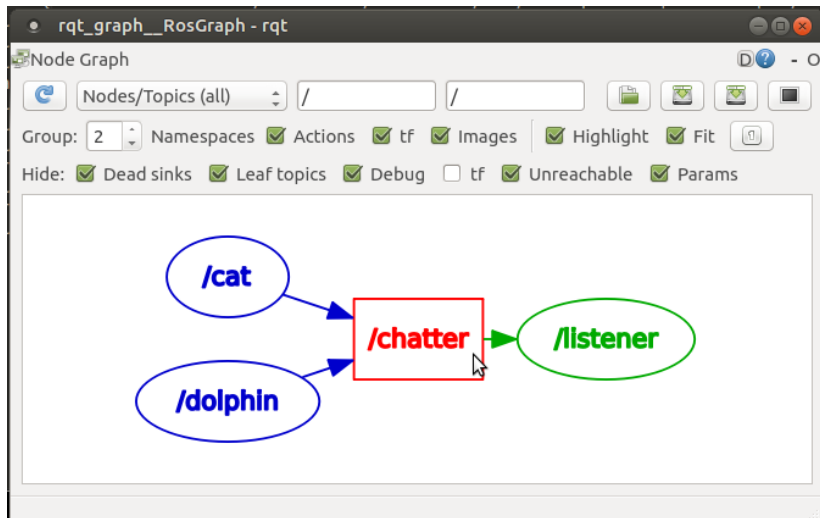
Merged outputs

```
vivek@vivek-VirtualBox: ~/ros_files/catkin_ws
File Edit View Search Terminal Tabs Help
vivek@vivek-Virtual... x roscore http://vivek... x vivek@vivek-Virtual... x vivek@vivek-Virtual... x
[INFO] [1579161128.719553]: hello world 1579161128.72
[INFO] [1579161128.720648]: /listenerI heard hello world 1579161128.72
[INFO] [1579161128.819908]: hello world 1579161128.82
[INFO] [1579161128.822130]: /listenerI heard hello world 1579161128.82
[INFO] [1579161128.929893]: hello world 1579161128.93
[INFO] [1579161128.930861]: /listenerI heard hello world 1579161128.93
[INFO] [1579161129.029166]: hello world 1579161129.03
[INFO] [1579161129.031629]: /listenerI heard hello world 1579161129.03
[INFO] [1579161129.120473]: hello world 1579161129.12
[INFO] [1579161129.122407]: /listenerI heard hello world 1579161129.12
[INFO] [1579161129.219457]: hello world 1579161129.22
[INFO] [1579161129.220468]: /listenerI heard hello world 1579161129.22
[INFO] [1579161129.321644]: hello world 1579161129.32
[INFO] [1579161129.322590]: /listenerI heard hello world 1579161129.32
[INFO] [1579161129.421741]: hello world 1579161129.42
[INFO] [1579161129.423075]: /listenerI heard hello world 1579161129.42
[INFO] [1579161129.519812]: hello world 1579161129.52
[INFO] [1579161129.521185]: /listenerI heard hello world 1579161129.52
[INFO] [1579161129.622828]: hello world 1579161129.62
[INFO] [1579161129.624495]: /listenerI heard hello world 1579161129.62
[INFO] [1579161129.723761]: hello world 1579161129.72
[INFO] [1579161129.725785]: /listenerI heard hello world 1579161129.72
[INFO] [1579161129.819839]: hello world 1579161129.82
[INFO] [1579161129.821903]: /listenerI heard hello world 1579161129.82
[INFO] [1579161129.920001]: hello world 1579161129.92
```


Passing arguments

- ▶ Multiple instances of same node with a variation

Example : Two versions of talker node sending different msg (as shown in the following ROS graph)



Launch file with arguments

```
1 <launch>
2   <arg name="dolphin_msg" default="Click_and_Squeak" />
3   <arg name="cat_msg" default="Meow_and_Purr" />
4
5   <node name="dolphin" pkg="test_pub_sub" type="
6   talker_mod.py" args="$(arg_dolphin_msg)" />
7   <node name="cat" pkg="test_pub_sub" type="
8   talker_mod.py" args="$(arg_cat_msg)" />
9   <node name="listener" pkg="test_pub_sub" type="
10  listener.py" output="screen" />
11 </launch>
```

3

Node modification

```
1  #!/usr/bin/env python
2  import sys
3  import rospy
4  from std_msgs.msg import String
5
6  def talker(greeting):
7      pub = rospy.Publisher('chatter', String, queue_size=10)
8      rospy.init_node('talker', anonymous=True)
9      rate = rospy.Rate(10)
10     while not rospy.is_shutdown():
11         hello_str = "{}".format(greeting)
12         rospy.loginfo(hello_str)
13         pub.publish(hello_str)
14         rate.sleep()
15
16 if __name__ == '__main__':
17     try:
18         talker(sys.argv[1])
19     except rospy.ROSInterruptException:
20         pass
```

4

⁴location bit.ly/2tdpTalkerMod

Assignment

- ▶ Create a new project with name in following format
NAME_ROLL_NUMBER
- ▶ Three nodes and one launch file
 - ▶ First will publish your NAME (String) to topic **name_listener**
 - ▶ Second will publish your ROLL NUMBER (Int) **roll_listener**
 - ▶ Third node will subscribe to both the above topics and print a assimilated string in following format

Student **NAME** has roll number : **ROLL_NUMBER**