SIT310 – Task 2
Using ROS

Overview

This task will introduce you to the core concepts of ROS. You will use catkin workspaces and ROS tools to create your first ROS program. You will create a simple publish-subscribe program using Python.

Task requirements

a. Ensure you have already completed task 1
b. You can complete this task using either ROS on the raspberry Pi or using a Cloud instance, or Linux on your own computer or in a virtual machine.
c. If you didn't attend the lecture this week, then either watch the lecture online and/or review the PowerPoint slides on the unit site.
d. Read the introduced concepts / theories / recommended readings below.
e. Read the task instructions
f. Complete the activities in the task guide
g. Review your progress with your lab tutor or cloud tutor.
h. Complete "Lab Assessment 1: Using ROS" – the assessment sheet is on the unit site.

Task guide

This task will guide you through creating a complete publish-subscribe ROS application. If you want further information, have a look at the "Additional resources" section below. Once you have finished this task, complete the "Lab Assessment 1" quiz on the unit site.

1. ROS Concepts

1.1. Review each of the following concepts. Try to get an idea for each concept rather than the full technical details – there is plenty of time for that.

- Nodes: A node is an executable that uses ROS to communicate with other nodes.

http://wiki.ros.org/Nodes

- Messages: ROS data type used when subscribing or publishing to a topic.

http://wiki.ros.org/Messages

- Topics: Nodes can *publish* messages to a topic as well as *subscribe* to a topic to receive messages. http://wiki.ros.org/Topics
- Master: Name service for ROS (i.e. helps nodes find each other) http://wiki.ros.org/Master
- rosout: ROS equivalent of stdout/stderr. http://wiki.ros.org/rosout
- roscore: Master + rosout + parameter server (parameter server will be introduced later). http://wiki.ros.org/roscore

2. ROS Nodes

2.1. In ROS, a node really isn't much more than an executable file within a ROS package. ROS nodes use a ROS client library to communicate with other nodes. Nodes can publish or subscribe to a Topic. Nodes can also provide or use a Service. You can develop ROS nodes using either C++ (using the roscpp package) or Python (using the rospy package). We will be using Python.

2.2. roscore

- Roscore is the core in any ROS installation. It is a collection of nodes that allow ROS nodes to communicate. It is also a ROS Master that provides naming and registration servers for other nodes. In some distributed systems architectures, you would call this the Broker or Naming Server.
- To start roscore you simply do the following.

```
$roscore
```

- You will have to do this every time you need to use ROS. It will take over the terminal – so you will always need an independent terminal for roscore.

2.3. rosnode

2.3.1. rosnode is a tool for displaying information about ROS nodes. It is mainly useful for seeing publications, subscriptions and connectors.

2.3.2. To list all the active ROS nodes running, do the following:

```
$ rosnode list
/rosout
```

2.3.3. The output shows that there is only a single node running, which is the log management node.

2.3.4. There is some extra information here: http://wiki.ros.org/rosnode

2.3.5. You can also get specific information about any active node as follows for the rosout node.

```
$ rosnode info /rosout
```

2.4. rosrun

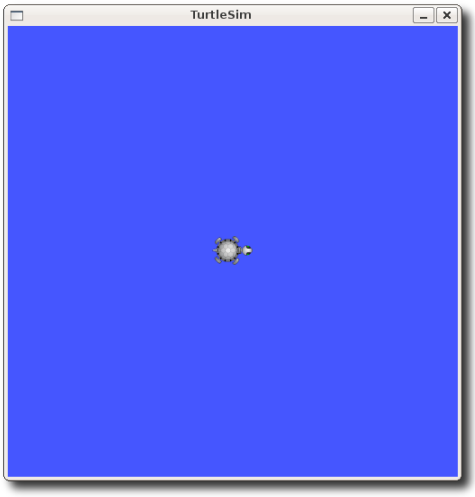
2.4.1. rosrun allows you to use the package name to directly run a node within a package (without having to know the package path). As follows.

```
$ rosrun [package_name] [node_name]
```

2.4.2. As an example, lets run the turtlesim_node from the turtlesim package.

```
$ rosrun turtlesim turtlesim_node
```

2.4.3. You will see the following, but with a random turtle.



2.4.4. In a new terminal, enter the following. This will list the active nodes, consisting of the log node (rosout) and the turtlesim node.

```
$ rosnode list
/rosout
/turtlesim
```

2.4.5. You can also name a node when you run it. For example, if you run turtlesim as follows, when you use rosnode list you will see it listed as the name.

```
$ rosrun turtlesim turtlesim_node __name:=my_turtle
(new terminal)

$ rosnode list
/my_turtle
/rosout
```

2.5. To test if a node is alive, you can use rosnode ping as follows.

```
$ rosnode ping my_turtle
Pinging /my_turtle with a timeout of 3.0s
Xmlrpc reply from http://ROS-Ubuntu:46233/ time=0.449896ms
…
```

3. ROS Topics

3.1. Nodes communicate with each other over ROS topics. Nodes can publish to a topic to send data to the topic. Nodes can subscribe to a topic to receive data from the topic. This short task will demonstrate the use of topics in ROS with the turtlesim tutorial.

3.2. Preparation

- You should have setup autonomic sourcing in .bashrc, if not:
  - `source /opt/ros/kinetic/setup.bash`
- Ensure you have roscore running
- Turtlesim should be already installed from the turtlesim package
- Run turtlesim as before:

```
$ rosrun turtlesim turtlesim_node __name:=my_turtle
```

3.3. Control Turtlesim using topics

- Turtlesim can be controlled using ROS topics. The turtlesim program is subscribed to a topic. So, if any program publishes messages to this topic, they can be read and acted upon by turtlesim
- You can publish messages using the turtle_teleop_key program. Run it as follows:

```
$ rosrun turtlesim turtle_teleop_key
```

- You can use the keys to drive the turtle around the window. Make sure you are in the window running the turtle_teleop_key program.

- This seems simple, but it demonstrates some very important ROS functionality. Both the turtlesim_node and turtle_teleop_key are communicating using ROS topics. Turtle_teleop_key is publishing and turtlesim_node is subscribing to the same topic

3.4. Graphing Topics

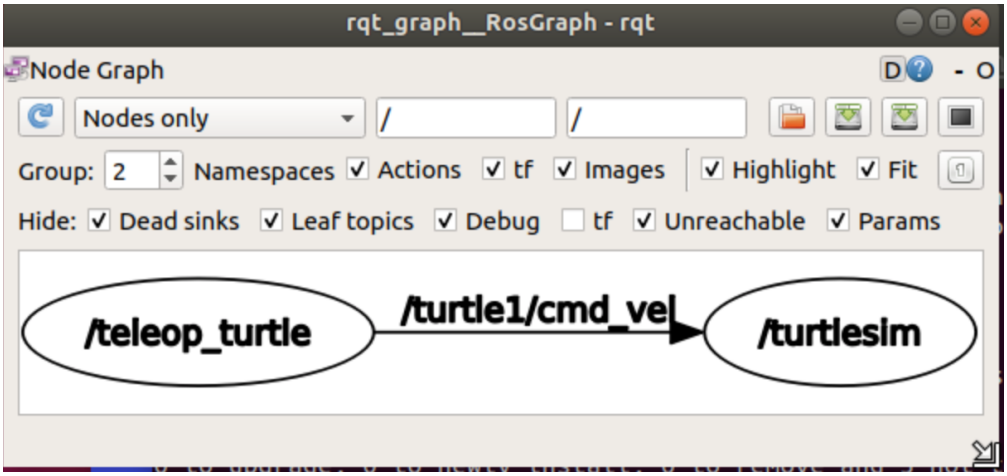
- You can use the rqt_graph tool to see whats going on in the system.
- Install it as follows

```
$ sudo apt-get install ros-kinetic-rqt
$ sudo apt-get install ros-kinetic-rqt-common-plugins
```

- Run the tool as follows.

```
$ rosrun rqt_graph rqt_graph
```

- You will be presented with the following window. You will see the turtlesim topic interaction between the different programs. This is very useful to see what parts of the system are communicating.



3.5. Using rostopic

- rostopic is a tool that allows you to get lots of information about ROS topics.
- Try running the tool as follows to see what it can do.

```
$ rostopic -h
rostopic bw     display bandwidth used by topic
rostopic delay  display delay of topic from timestamp in header
rostopic echo   print messages to screen
rostopic find   find topics by type
rostopic hz     display publishing rate of topic
rostopic info   print information about active topics
rostopic list   list active topics
rostopic pub    publish data to topic
rostopic type   print topic or field type
```

- You can also press tab after rostopic to see all the commands listed

3.6. rostopic echo

- To actually see data you can use rostopic echo – as follows:

```
rostopic echo [topic]
```

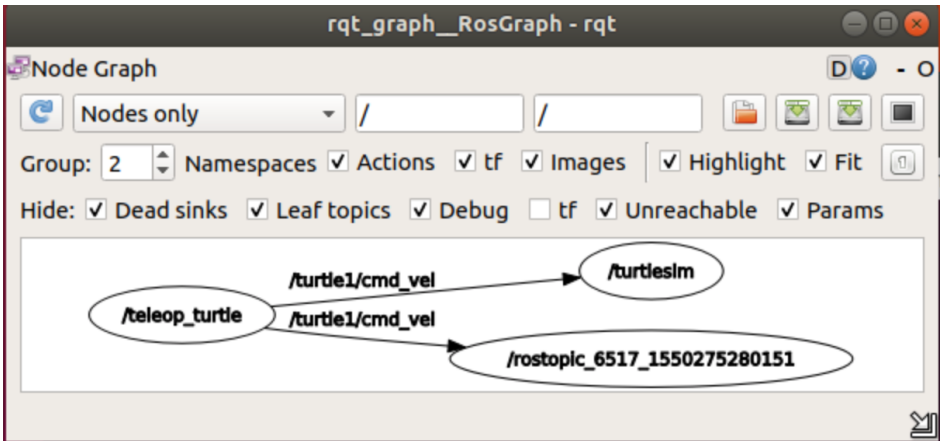
- To test it out properly, try it on the turtlesim as follows:

```
$ rostopic echo /turtle1/cmd_vel
```

- …and move your turtle around with your keys
- You will see the following…

```
kev@ROS-Ubuntu:~$ rostopic echo /turtle1/cmd_vel
linear:
  x: 2.0
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 0.0
---
linear:
  x: 2.0
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 0.0
---
```

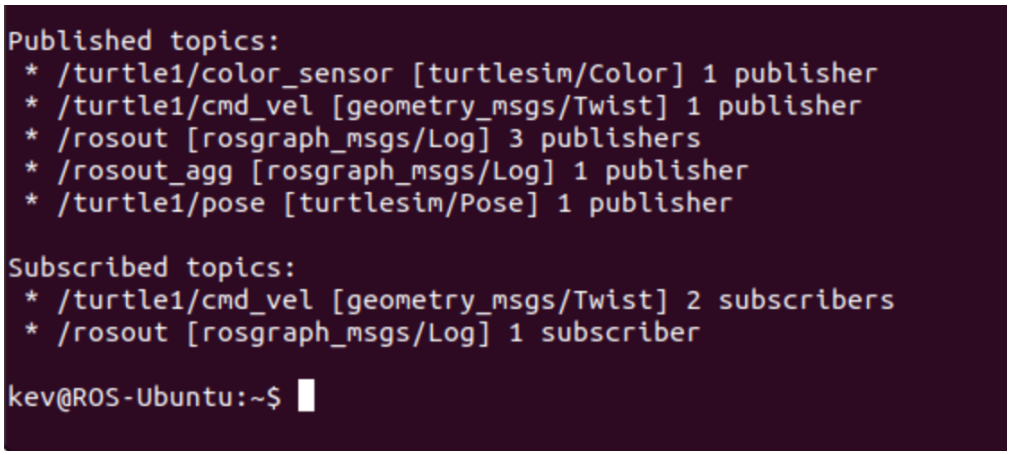
- If you look at rqt_graph, you will see the rostopic tool is also listed, as follows…



3.7. Rostopic list

- As well as seeing data on a particular topic, you can also display a list of all the topics which are currently in use.
- To display a list of topics currently in use, run the following:

```
$ rostopic list -v
```



3.8. Message type

- Like C-style languages, messages passed using ROS topics have a type. To determine the type of the topic, use the following.

```
$ rostopic type [topic]
```

- For our simturtle, try:

```
$ rostopic type /turtle1/cmd_vel
geometry_msgs/Twist
```

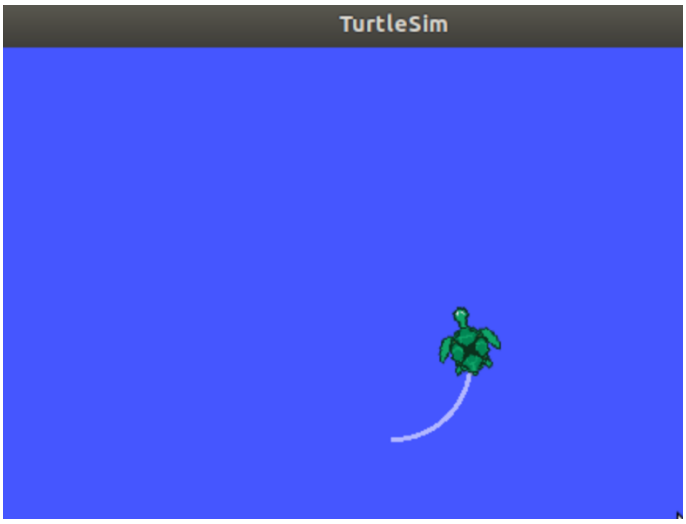
- You can then see details of the type is, as follows.

```
$ rosmsg show geometry_msgs/Twist
geometry_msgs/Vector3 linear
  float64 x
  float64 y
  float64 z
geometry_msgs/Vector3 angular
  float64 x
  float64 y
  float64 z
```

- We can go further and actually post message.

```
$ rostopic pub -1 /turtle1/cmd_vel geometry_msgs/Twist -- '[2.0, 0.0, 0.0]' '
[0.0, 0.0, 1.8]'
```

- This publishes some coordinates in the Twist type format to the simturtle topic.
- If all goes well, you should see the following in turtlesim.



- For more explanation about these arguments, have a look at: http://wiki.ros.org/ROS/Tutorials/UnderstandingTopics#Using_rostopic_pub

4. Creating your first ROS application

   For this tutorial we will be using Python. It is also possible to use C++ to develop ROS applications. Python programs tend to be shorter, so we will use Python.

   4.1. Create a package

   - Navigate to your carkin_ws folder

```
$ cd ~/catkin_ws/src
$ catkin_create_pkg sit310_lab2 std_msgs rospy roscpp
$ cd ~/catkin_ws
$ catkin_make
$ . ~/catkin_ws/devel/setup.bash
```

   - The second to last command runs a build operation. This is a good day of checking that everything is fully set up. The last command ensures that your packages can be found by ROS.

   - Create a scripts folder

```
$ roscd sit310_lab2
$ mkdir scripts
```

```
$ cd scripts
```

- Retrieve two files from the ROS wiki which demonstrate the basic publish/subscribe mechanism of ROS. We will use the Linux tool wget, which downloads files from web sites to the file system.

```
$ wget https://raw.github.com/ros/ros_tutorials/kinetic-devel/rospy_tutorials
/001_talker_listener/talker.py

$ wget https://raw.github.com/ros/ros_tutorials/kinetic-devel/rospy_tutorials
/001_talker_listener/listener.py
```

- Make both files executable

```
$ chmod +x talker.py

$ chmod +w listener.py
```

- Build yours ROS nodes. Even though you are using Python, you still have to build your ROS nodes.

```
$ cd ~/catkin_ws

$ catkin_make
```

4.2. Running the programs

- Run roscore as you always need to do.

```
$ roscore
```

- Navigate to your catkin workspace and call your setup.sh. You will need to create a new terminal window and do this for each ROS program you run.

```
$ cd ~/catkin_ws

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

- Run the publisher. It will begin to publish messages on a ROS topic. (Remember that tab completion works)

```
$ rosrun sit310_lab2 talker.py

[INFO] [WallTime: 1314931969.258941] /listener_17657_1314931968795I heard hel
lo world 1314931969.26

[INFO] [WallTime: 1314931970.262246] /listener_17657_1314931968795I heard hel
lo world 1314931970.26
```

- Run the subscriber. It subscribes to the ROS topic, and will begin to receive messages and print them.

```
$ rosrun sit310_lab2 listener.py

[INFO] [WallTime: 1314931969.258941] /listener_17657_1314931968795I heard hel
lo world 1314931969.26

[INFO] [WallTime: 1314931970.262246] /listener_17657_1314931968795I heard hel
lo world 1314931970.26
```

- Run rqt_graph, rostopic list, rostopic echo etc to confirm what is going on.
- Use Ctrl-C to exit each of your programs.

4.3. Publisher Node

- The publisher node is explained here, have a read of the explanation.

http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber%28python%29#rospy_tutorials.2BAC8-Tutorials.2BAC8-WritingPublisherSubscriber.Writing_the_Publisher_Node

4.4. Subscriber Node

- The subscriber node is explained here, have a read of the explanation.

http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber%28python%29#rospy_tutorials.2BAC8-Tutorials.2BAC8-WritingPublisherSubscriber.Writing_the_Subscriber_Node

5. Now that you have completed the first 2 tasks of the unit, you should complete the "Lab Assessment 1" quiz on the unit site by the deadline.

6. Additional resources

6.1. Learn about linux

- https://www.computerworld.com/article/2598082/linux/linux-linux-command-line-cheat-sheet.html
-

6.2. Learn about ROS.

- http://wiki.ros.org
- A Gentle Introduction to ROS: https://www.cse.sc.edu/~jokane/agitr/
- http://wiki.ros.org/kinetic/Installation/Ubuntu
- http://wiki.ros.org/ROS/Tutorials