

Laboratory focused on basics of ROS

ROBa - Laboratory number 4

<https://github.com/Adam-Fabo/ROB-laboratories>

Brno university of technology
Faculty of informatics
Adam Fabo
1.2.2023

Basics of ROS Laboratory

Welcome to the fourth laboratory. This laboratory is aimed at students with no previous experience with ROS - Robot Operating System.

Prerequisites

Prerequisites are that you attended theoretical lecture about ROS or that you are familiar with basic concepts like Nodes, topics and publisher/subscriber. Also, it is required that you know basics of bash and python programming.

Goals of the laboratory

Goals of this laboratory are set to introduce you with basic work with Robot Operating System. You will learn:

- Setting up ROS environment
- Basic ROS commands
- Creating simple Publisher and Subscriber
- Sending responses based on message

Theory

ROS

Robot Operating System (ROS) is set of open source software libraries and tools that help you build robot applications. ROS has many distributions, distribution used in this course is ROS Noetic. Main parts of ROS application are: ROS Master, Node, Topic, Message.

- **ROS Master** is a core component of the ROS architecture that coordinates communication between nodes in a ROS system. The master is responsible for maintaining a network topology, registering nodes and their communication topics, and facilitating the exchange of messages between nodes.
- **Node** is a process that performs computation. It can be publisher, subscriber or both at the same time. **Publisher** is type a node that sends messages into a topic. **Subscriber** is type of node that reads messages from topic. Each node belong to some package.
- **Topic** is a named bus over which nodes exchange messages
- **Message** is structured carrier of information. It can be represented by class in programming languages.

ROS has following concept: Firstly there needs to be an instance of ROS Master. Only after ROS master runs, nodes can be registered. Nodes exchange messages with other nodes.

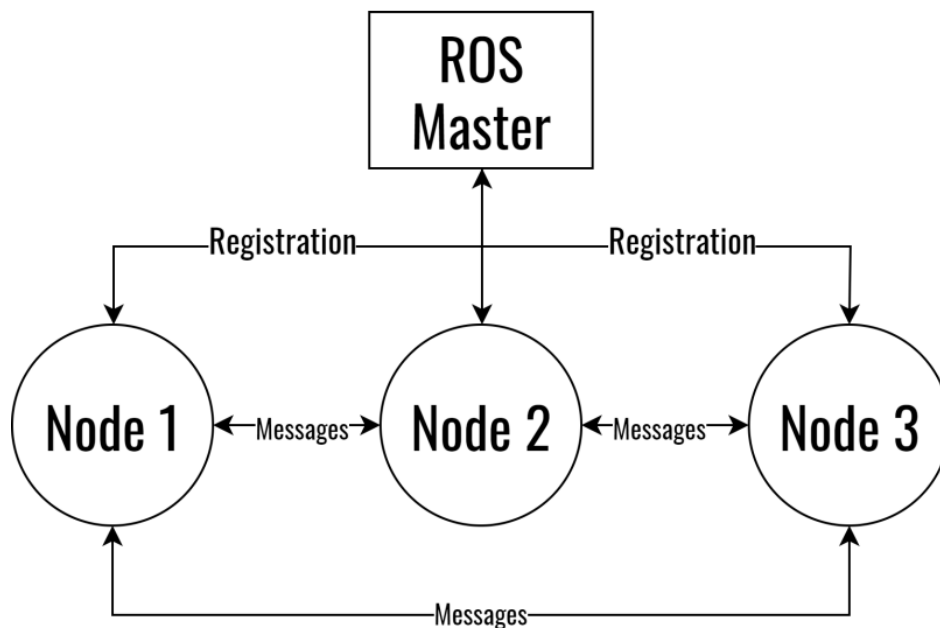


Figure 1: ROS concept

Nodes exchange messages using topics. There can be multiple nodes publishing messages to one topic and multiple nodes subscribing messages from the same topic. All the background message networking is covered by running ROS master instance.

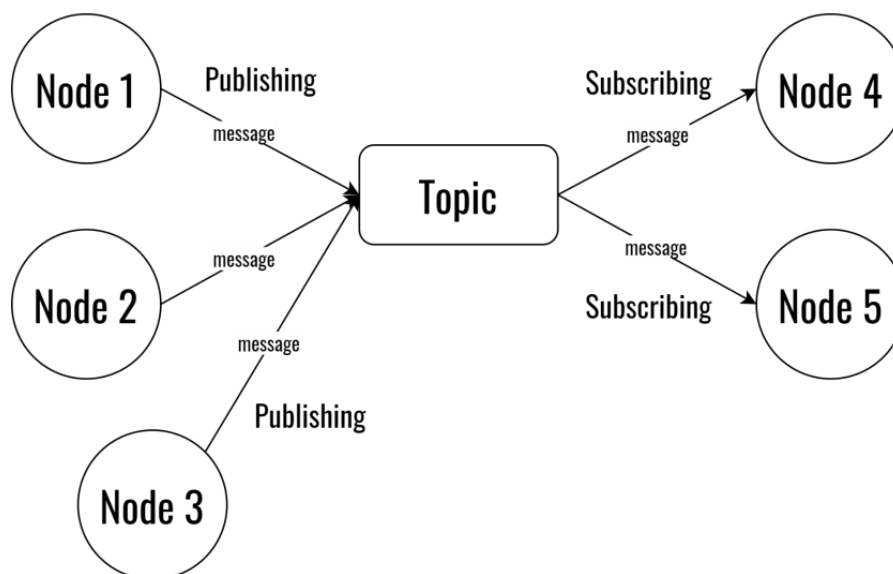


Figure 2: Communication using topics

You, as a user, can create custom nodes and topic and design your own architecture of a system.

ROS commands

When working with ROS, there is need to know basic ROS commands. ROS has over 20 commands and most of them have multiple arguments. There is no need to know all of them for this laboratory, and the needed commands for this lab are divided into 2 categories: Essential and Debug. Essential commands are following:


- **roscore** - Starts ROS master. Run this command in one terminal window and keep it running until you finish your work.
- **roslaunch** [package] [node] - Starts node. When starting node, package of the node must be specified.

Debug commands are used to display info about node/topic/message or the whole architecture of a system. Usefull debug commands for this laboratory are following:

- **rqt_graph** - Opens new window where all Nodes and Topics are visualized.
- **rostopic** - Opens new window where all Nodes and Topics are visualized.
- **rostopic** - Opens new window where all Nodes and Topics are visualized.
- **rostopic** - Opens new window where all Nodes and Topics are visualized.

sample commands

Terminal tips

When working with ROS, there is need for having multiple terminal windows to be opened. That is why it is recommended to use application **Terminator**  which allows to split terminal window.

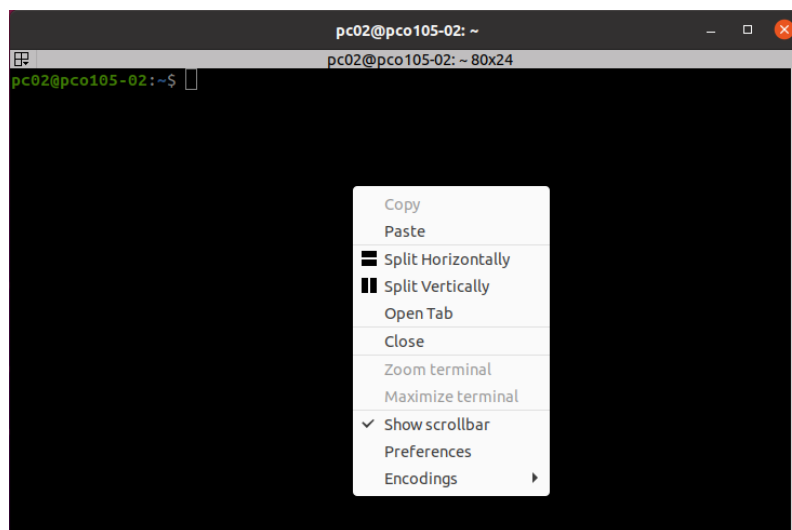


Figure 3: ROS concept

Open the dropdown option by right-clicking and create at least 4 windows by selecting Split Vertically and Split Horizontally. Your result should be like image at figure XY. If you need more terminals in the future, you can simply split more terminals.

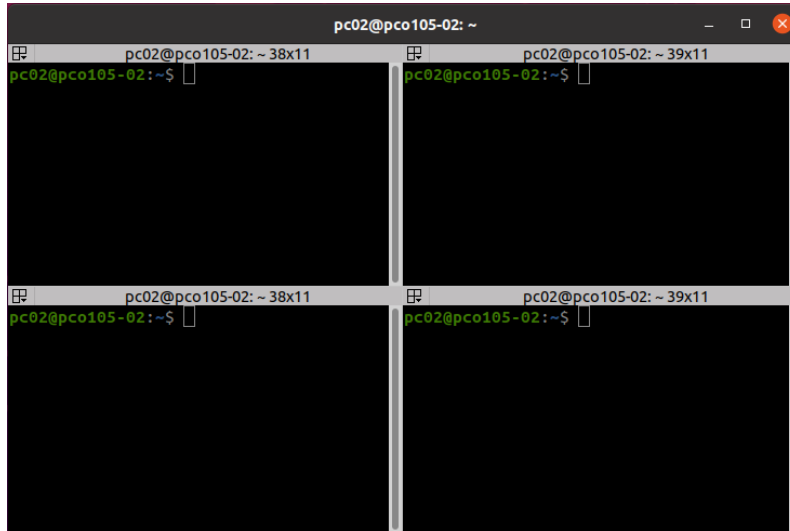


Figure 4: ROS concept

ROS and Python

ROS offers pure python client library call rospy. Thanks to rospy and its API it is possible to write custom nodes. Rospy can be imported to python by:

```
import rospy
```

Each ROS node written in python needs to be initialized - so that ROS master knows about the node. Node can be initialize by:

```
rospy.init_node('Node_name')
```

- where Node_name is custom name of the node. It cannot contain space.

Is is best convention to write this line at start of the program.

ROS Publisher in Python

Simplest ROS publisher can be written as following:

ROS publisher can be created by following:

```

5  # Import everything important
6  import rospy
7  from std_msgs.msg import String
8
9  # Start of the program
10 if __name__ == '__main__':
11     # tell ROS name of this node
12     rospy.init_node('Basic_Publisher')
13
14     # Create publisher that sends messages to topic "/chatter" and message type is String
15     pub = rospy.Publisher('/chatter', String, queue_size=10)
16
17     # Set message speed to 2Hz
18     r = rospy.Rate(2)
19
20     # Infinite loop while ROS is running
21     while not rospy.is_shutdown():
22         # Send your own custom message
23         pub.publish("Your message here")
24         # Sleep for a given time
25         r.sleep()

```

Figure 5: ROS concept

Lets take a look at the code in detail. Firstly there needs to be imported rospy library. This allows python accessing ROS API. Second import imports type of message that we will be sending. In this case it will be String.

```

5  # Import everything important
6  import rospy
7  from std_msgs.msg import String

```

At the beginning of each ROS node written in python needs to line that tells ROS name of the created node. Function `rospy.init_node()` take one argument, which is name of the node. You can choose any name you want, but this name cannot contain spaces.

```

11     # tell ROS name of this node
12     rospy.init_node('Basic_Publisher')

```

Creating instance of Publisher class can be done by calling `rospy.Publisher()`. It takes 3 positional argumets: topic, message type and queue size. Topic says to which topic will publisher be publishing messages. Message type is type of message that will be published. In this case it is String and it was imported at beginning of the code. Queue size says that how many messages are buffered in case that publisher is publishing more messages that can be sent.

```

14     # Create publisher that sends messages to topic "/chatter" and message type is String
15     pub = rospy.Publisher('/chatter', String, queue_size=10)

```

ROS messages are published periodically. Frequency of messages sent can be set by `rospy.Rate()`, which takes one argument - how many messages to send each second. In order to keep program running, there needs to be infinite loop. Ideally you want to be sending messages only when ROS master is running. For this there is function `rospy.is_shutdown()` which returns True of ROS master is not running.

For publishing messages to topic, class Publisher has method publish. This method needs to be called on instance created in previous step. It takes one required parameter - message to be sent. Usually it is instance of a message object, but in this case, message sent is String. Python string is auto converted to String object,

Final step is to pause this loop for a given time. For this can be used rospy.Rate() sleep function.

```
17     # Set message speed to 2Hz
18     r = rospy.Rate(2)
19
20     # Infinite loop while ROS is running
21     while not rospy.is_shutdown():
22         # Send your own custom message
23         pub.publish("Your message here")
24         # Sleep for a given time
25         r.sleep()
```

ROS Subscriber in Python

Simple ROS subscriber can be written as following:

```
5     # Import everything important
6     import rospy
7     from std_msgs.msg import String
8
9
10    # Callback function that is called when subscriber gets data
11    def callback(data):
12        rospy.loginfo("Received data: %s", data.data)
13
14
15    # Start of the program
16    if __name__ == '__main__':
17        # tell ROS name of this node
18        rospy.init_node('Basic_Subscriber', anonymous=True)
19
20        # Create subscriber that subscribes messages from topic "/chatter" of type String
21        # callback function is called when message is received
22        rospy.Subscriber("/chatter", String, callback)
23        rospy.spin()
```

Same as in publisher, rospy library and message type needs to be imported.

```
5     # Import everything important
6     import rospy
7     from std_msgs.msg import String
```

It is needed to define callback function (does not have to be named callback). This function takes one argument, which is data passed by subscriber. Callback function is called when subscriber receives message.

```

10 # Callback function that is called when subscriber gets data
11 def callback(data):
12     rospy.loginfo("Received data: %s", data.data)

```

In main part of subscriber's code needs to be initialization of node by calling `rospy.init_node()` function. If parameter `anonymous=True`, then ROS appends to name of a node random numbers in order to make it unique. This allows for running multiple subscribers without errors.

After that subscriber can be created by calling `rospy.Subscriber()`. It needs to know from which topic to subscribe, what type of a message it should expect and callback function which will be called if message is received.

```

18     rospy.init_node('Basic_Subscriber', anonymous=True)
19
20     # Create subscriber that subscribes messages from topic "/chatter" of type String
21     # callback function is called when message is received
22     rospy.Subscriber("/chatter", String, callback)
23     rospy.spin()

```

ROS Messages in Python

ROS publisher - sending one message

```

13 pub = rospy.Publisher('/chatter', String, queue_size=10)
14
15 while pub.get_num_connections() < 1:
16     pass
17
18 pub.publish("Your message here")

```


Exercises

Running ROS master

Test if your ROS master runs

1. Open new terminal using **Terminator** app
2. Split the terminal into 4 parts as is described in chapter Terminal tips
3. Click onto top left terminal and run `roscore` command
4. You should see last line of the message say: "started core service [/rosout]"
5. Do not exit this terminal, let it run and use other terminal windows

Running simple ROS nodes

Use built in tutorial package that offers simple publisher and subscriber

1. Create publisher by running command: `roslaunch rospy_tutorials talker`
2. In another window create subscriber by running command: `roslaunch rospy_tutorials listener`
3. Now, the publisher is sending messages and subscriber is listening to them
4. Visualize this by running command: `rqt_graph`
5. New window should open where you can see visual representation of your running nodes

Getting info

Run some simple commands that give you info about topic, nodes, messages

1. Keep the talker and listener from last exercise running
2. Run `rostopic list` to see running nodes
3. Run `rostopic info /[node name]` where node name is name of one of the node listed by previous command (hint: you can use TAB to autocomplete node names)
4. Run `rostopic list` to see active topics
5. Run `rostopic info /chatter` to see type of message that this topic accepts and also publishers and subscribers that interact with this topic
6. Run `rostopic list` to see list of all message types
7. Run `rostopic show std_msgs/String` to see info about one message type

Python Publisher

Creating your own publisher that sends messages to topic /chatter

1. If you have talker running from last exercise, kill it. Run only listener
2. Open PyCharm by running command `pycharm` in terminal windows
3. Navigate to file "XYZ"
4. Implement publisher that publishes String message to /chatter topic every 2 seconds.
5. Run the node by running `roslaunch XYZ`
6. Take a look at the listener - it should be receiving your messages

Python subscriber

Creating your own publisher that subscribes messages from topic /chatter

1. If you have listener running from last exercise, kill it. Run only talker
2. Open PyCharm by running command `pycharm` in terminal windows (or keep it open from last exercise)
3. Navigate to file "XYZ"
4. Implement subscriber that subscribes String message from /chatter
5. Run the node by running `roslaunch XYZ`
6. Take a look at the output from your code - it should be receiving messages

Chatting between PC - Publisher

Modify publisher code that you created, so that you can write your own custom messages. Make it so that your messages are shared between all students.

1. Modify your subscriber by adding function `input()` that reads your input from keyboard
2. Change topic to /shared_chatter
3. Call the teacher in this step, or stop running the ROS master and change XYZ in file XYZ
4. Run your node and write some messages

Chatting between PC - Subscriber

Modify subscriber code that you created, so that you can receive messages from your classmates.

1. Change topic to /shared_chatter
2. Run your node and write some messages using your previous publisher. You should see your messages and messages from your classmates

Chatting between PC - Subscriber - Auto reply

Modify subscriber callback function that you created, in a way that it sends automatic response if it detects word Hello. Make the response be "Hello" in your own language.

1. Modify your subscriber callback function. Check if received string contains "Hello"
2. If string contains "Hello" write single response that contains "Hello" in your own language
3. You need to create new publisher in callback function.
4. Test your implementation by writing "Hello" to the topic.