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
23.3.2023

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 scripting.

Goals of the laboratory

The goals of this laboratory are set to introduce you to the basics of 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 a set of open source software libraries and tools that help you build robot applications. ROS has many distributions, the 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.
- **A message** is a structured carrier of information. It can be represented by class in programming languages.

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

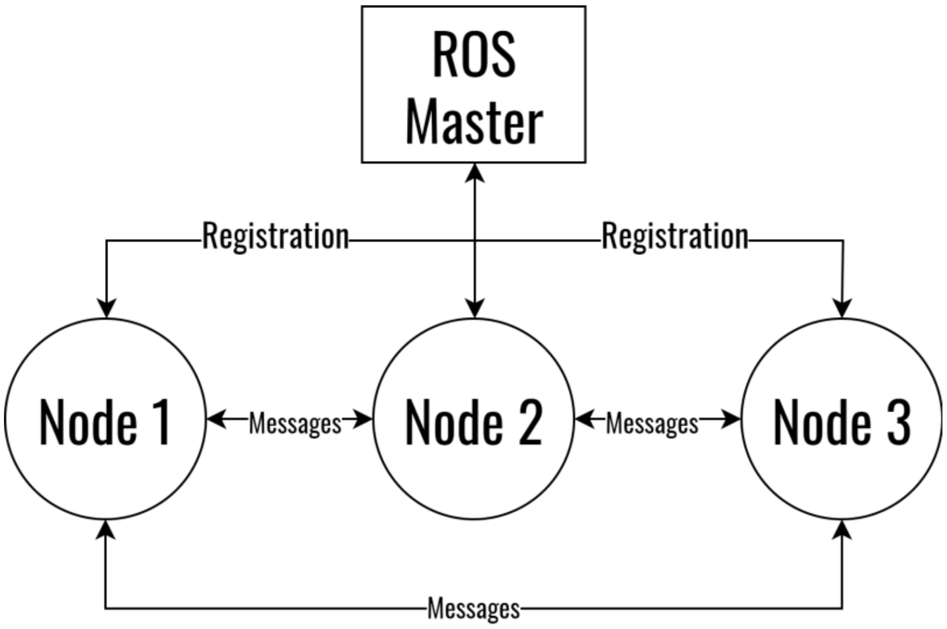


Figure 1: ROS concept

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.

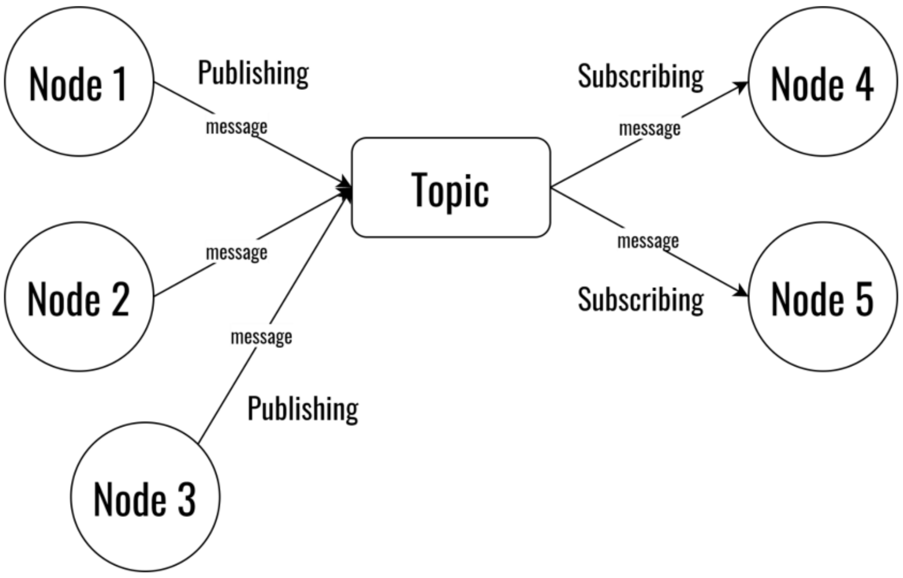


Figure 2: Communication using topics

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 a 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:

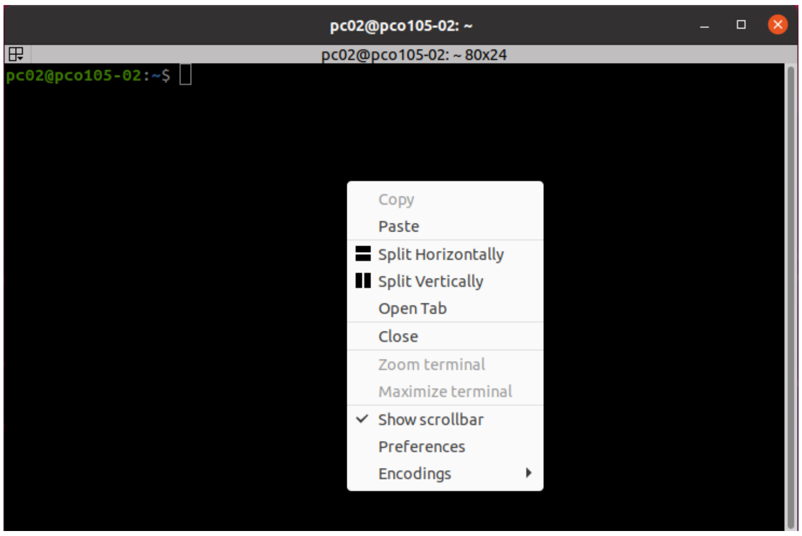
- **roscore** - Starts ROS master. Run this command in one terminal window and keep it running until you finish your work.
- **rosrun** [package] [node] - Starts node. When starting node, package of the node must be specified.

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

- **rqt_graph** - Opens new window where all Nodes and Topics are visualized.
- **rosnode** - command-line tool that displays information about ROS Nodes.
- **rostopic** - command-line tool that displays information about ROS topics.
- **rosmsg** - command-line tool that displays information about ROS messages.

Terminal tips

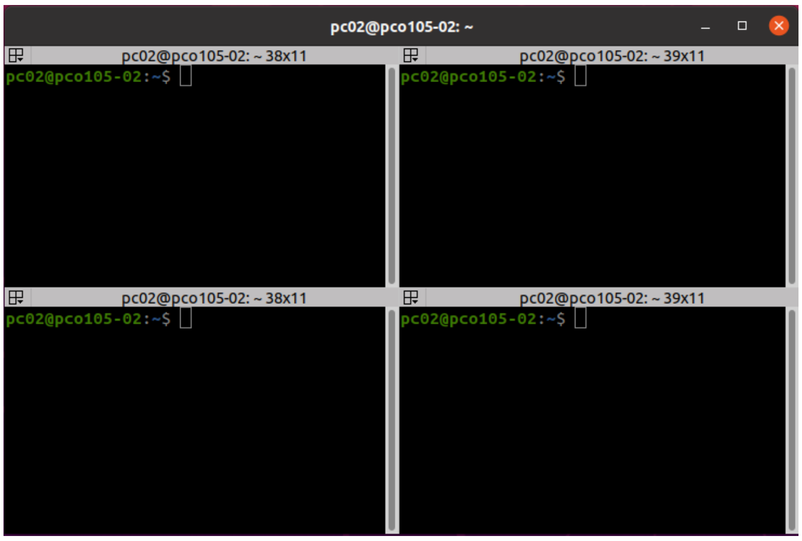
When working with ROS, there is need for having multiple terminal windows to be opened. That is why it is recommended to use application **Terminator** which allows splitting of terminal window.



```
pc02@pco105-02:~$
```

Figure 3: Terminator terminal

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



```
pc02@pco105-02: ~
pc02@pco105-02: ~ 38x11          pc02@pco105-02: ~ 39x11
pc02@pco105-02:~$                pc02@pco105-02:~$

pc02@pco105-02: ~ 38x11          pc02@pco105-02: ~ 39x11
pc02@pco105-02:~$                pc02@pco105-02:~$
```

Figure 4: Terminator splitted terminal

ROS Publisher in Python

ROS offers pure python client library call rospy. Thanks to rospy and its API, it is possible to write custom nodes. Simplest ROS publisher can be written as following:

```python
# Import everything important
import rospy
from std_msgs.msg import String

# Start of the program
if __name__ == '__main__':
    # tell ROS name of this node
    rospy.init_node('Basic_Publisher')

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

    # Set message speed to 2Hz
    r = rospy.Rate(2)

    # Infinite loop while ROS is running
    while not rospy.is_shutdown():
        # Send your own custom message
        pub.publish("Your message here")
        # Sleep for a given time
        r.sleep()
```

Figure 5: ROS Publisher

Let's 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.

```python
# Import everything important
import rospy
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()* takes one argument, which is the name of the node. You can choose any name you want, but this name cannot contain spaces.

```python
    # tell ROS name of this node
    rospy.init_node('Basic_Publisher')
```

Creating an instance of Publisher class can be done by calling *rospy.Publisher()*. It takes 3 positional arguments: **topic**, **message type** and **queue size**. Topic says to which topic will publisher be publishing messages. Message type is the 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.

```python
    # Create publisher that sends messages to topic "/chatter" and message type is String
    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 a 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 an instance of a message object, but in this case, the 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.

```python
    # Set message speed to 2Hz
    r = rospy.Rate(2)

    # Infinite loop while ROS is running
    while not rospy.is_shutdown():
        # Send your own custom message
        pub.publish("Your message here")
        # Sleep for a given time
        r.sleep()
```

ROS Subscriber in Python

Simplest ROS subscriber can be written as following:

```python
# Import everything important
import rospy
from std_msgs.msg import String


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


# Start of the program
if __name__ == '__main__':
    # tell ROS name of this node
    rospy.init_node('Basic_Subscriber', anonymous=True)

    # Create subscriber that subscribes messages from topic "/chatter" of type String
    # callback function is called when message is received
    rospy.Subscriber("/chatter", String, callback)
    rospy.spin()
```

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

```python
# Import everything important
import rospy
from std_msgs.msg import String
```

It is needed to define a 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.

```python
# Callback function that is called when subscriber gets data
def callback(data):
    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 the name of a node random numbers in order to make it unique. This allows for running multiple subscribers without errors.

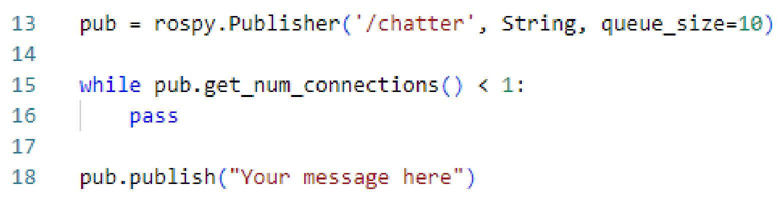
After that, subscriber can be created by calling *rospy.Subscriber()*. It needs to know from which topic to subscribe, what type of message it should expect and callback function which will be called if message is received.

```python
    rospy.init_node('Basic_Subscriber', anonymous=True)

    # Create subscriber that subscribes messages from topic "/chatter" of type String
    # callback function is called when message is received
    rospy.Subscriber("/chatter", String, callback)
    rospy.spin()
```

ROS publisher - sending one message

Sometimes it is desired to only send one message. For this can be used following code:



```python
pub = rospy.Publisher('/chatter', String, queue_size=10)

while pub.get_num_connections() < 1:
    pass

pub.publish("Your message here")
```

Function *pub.publish()* cannot be called right after creation of publisher, because it takes some time for the publisher to connect with ROS master. This is why the **while** loop is needed - it halts the program until publisher is connected. Only after that message can be sent. Otherwise, message would be published, but since publisher is not connected, the message would be simply discarded.

Exercises

Running ROS master

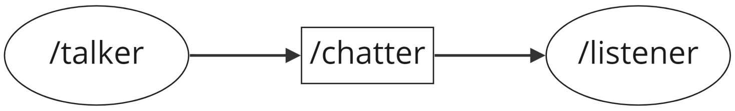
Test if your ROS master runs

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



Getting info

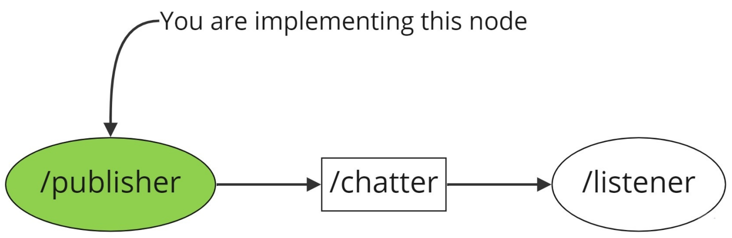
Run some simple commands that give you info about topic, nodes, messages

1. Keep the talker and listener from last exercise running
2. Run `rosnode list` to see running nodes
3. Run `rosnode info /[node name]` where node name is name of one of the node listed by previous command (hint: you can use TAB to auto-complete 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 `rosmsg list` to see list of all message types
7. Run `rosmsg show std_msgs/String` to see info about one message type

Python Publisher

Create your own publisher that sends messages to topic /chatter

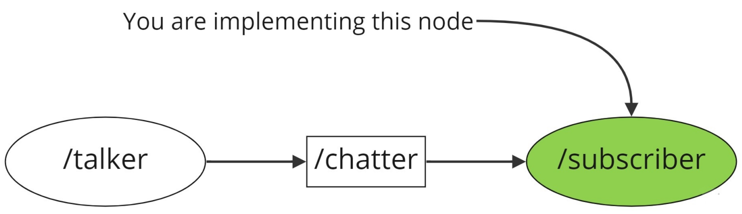
1. If you have a talker running from the last exercise, kill it. Run only listener
2. Open PyCharm by running command `pycharm` in terminal window
3. Navigate to file **basic_publisher/src/publisher.py**
4. Implement publisher that publishes String message to /chatter topic every 2 seconds.
5. Run the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



Python subscriber

Create your own publisher that subscribes messages from topic /chatter

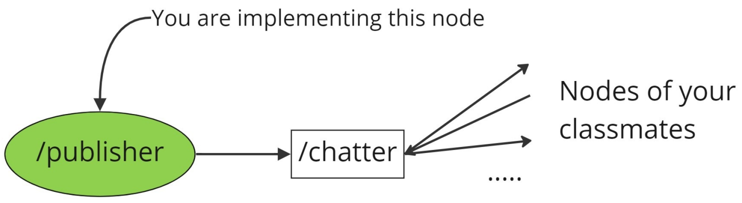
1. If you have a listener running from last exercise, kill it. Run only talker
2. Open PyCharm by running command `pycharm` in terminal window (or keep it open from the last exercise)
3. Navigate to file **basic_subscriber/src/subscriber.py**
4. Implement subscriber that subscribes String message from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



Chatting between PC - Publisher

Modify publisher in a way, that user can write custom messages. Make it so that messages are shared between all students.

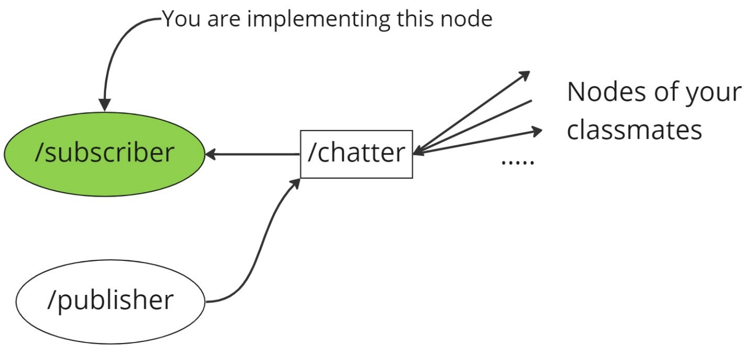
1. Continue working in your previous publisher code, or open file **basic_publisher/src/publisher_input.py**
2. Modify publisher by adding function input() that reads input from keyboard
3. Change topic to /shared_chatter
4. Call the teacher in this step, or stop running the ROS master and change ROS_MASTER_URI in **.bashrc** file
5. Run your node with `rosrun basic_publisher publisher_input.py` and write some messages



Chatting between PC - Subscriber

Modify subscriber in a way, so that you can receive messages from your classmates.

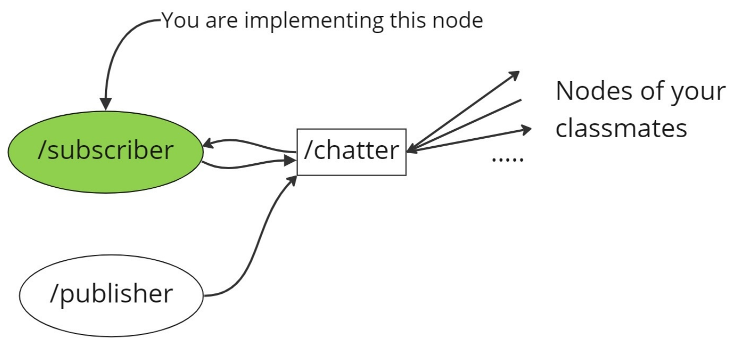
1. Continue working in your previous publisher code, or open file:
   basic_subscriber/src/subscriber_auto_reply.py
2. Change topic to /shared_chatter
3. Run your node with `rosrun basic_subscriber subscriber_auto_reply.py` and write some messages using your previous publisher. You should see your messages and messages from your classmates



Chatting between PC - Subscriber - Auto reply

Modify subscribers callback function in a way, that it sends automatic response if it detects word "Hello". Respond by word "Hello" from your own language,

1. Continue working in your previous publisher code, or open file:
   basic_subscriber/src/subscriber_auto_reply.py
2. Modify your subscriber's callback function. Check if received string contains "Hello"
3. If string contains "Hello" write single response that contains "Hello" in your own language
4. You need to create a new publisher in callback function, that sends only one message.
5. Test your implementation by writing "Hello" to the topic.


You are implementing this node
/subscriber
/chatter
Nodes of your classmates
.....
/publisher