TP1: first steps ROS

Introduction to the environment

In the following you will practice the basics of ROS: subscribing and publishing to a **topic**, create your first **node** and controlling a robot with the keyboard.

All the ROS environment is already setup into a Docker container. To start it, enter the following command line:

```
./run.sh
```

To exit the container, use `Ctrl` + `D`, or the command line:

```
exit
```

ROS needs many terminals to run the commands we need in this TP. So, we encourage you to use a terminal multiplexer, as *terminator*.

Run `terminator` command to open a new window with **terminator**. Here some useful shortcuts:

- Toggle fullscreen: `F11`
- Split terminals horizontally: `Ctrl` + `Shift` + `O`
- Split terminals vertically: `Ctrl` + `Shift` + `E`
- Close current Panel: `Ctrl` + `Shift` + `W`
- Open new tab: `Ctrl` + `Shift` + `T`
- Move to the terminal above the current one: `Alt` + `↑`
- Move to the terminal below the current one: `Alt` + `↓`
- Move to the terminal left of the current one: `Alt` + `←`
- Move to the terminal right of the current one: `Alt` + `→`

Part 1: publishing/subscribing to a topic

The first thing you should run is the **roscore** when you ROS, with the command:

```
roscore
```

Now open a new terminal above (`Ctrl` + `Shift` + `O`) and split it in two (`Ctrl` + `Shift` + `E`). You can click or use the shortcuts to navigate between the terminals.

In one terminal, list the current published topics using

```
rostopic list
```

For more information about rostopic you can use

```
rostopic help
```

In one terminal, subscribe to the topic `/my_topic`, with the following command:

```
rostopic echo /my_topic
```

In the oher terminal, use the following command to publish a message containing a Float32 value to this topic:

```
rostopic pub /my_topic std_msgs/Float32 "data: 42.0"
```

The message sent will be shown in the first terminal. Well done, you just published your first message in ROS!

List of ros message types could be found using the following command

```
rosmsg list
```

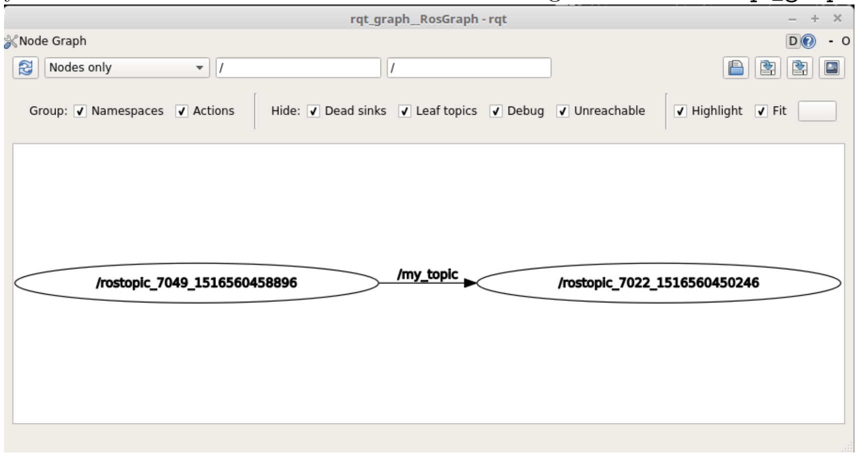
You can use the following command to get information about a message:

```
rosmsg info name_of_message_type
```

More difficult: - Use the correct command to display the information concerning the message type *std_msgs/String* - Publish a message each second on /my_second_topic with the type std_msgs/String and with your name as content - Listen to this topic

Tips:

- to kill a command you can use `Crtl` + `C`
- try to change the value of the data sent by the publisher to understand the mechanism
- you can publish a value at a specific rate, with the option `-r`. For a message sent each second: `sh rostopic pub -r 1.0 /my_topic std_msgs/Float32 "data: 42.0"`
- use `rostopic list` to list all topics available
- you can show the connection between nodes using the command `rqt_graph`


rqt_graph__RosGraph - rqt
Node Graph
Nodes only
/
/
Group: ✓ Namespaces ✓ Actions
Hide: ✓ Dead sinks ✓ Leaf topics ✓ Debug ✓ Unreachable
✓ Highlight ✓ Fit
/rostopic_7049_1516560458896
/my_topic
/rostopic_7022_1516560450246

Part 2: let's code it in Python!

Publisher node

Create a file `talker.py`.

This node called `talker` will publish a `std_msgs/Float32` message to the topic `/counter`.

Complete the following code to publish every second the current counter, incremented by one at every loop cycle:

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

# python client for ROS
import rospy
# Float32 message
from std_msgs.msg import Float32

def talker():
    # init the node: rospy.init_node(NODE_NAME)
    rospy.init_node(        )  # <--COMPLETE HERE

    # init the publisher with the method signature: rospy.Publisher(TOPIC_NAME, MESSAGE_TYP
    pub = rospy.Publisher(        )  # <-- COMPLETE HERE

    # publisher rate: 1Hz
    rate = rospy.Rate(1)
    counter = 0

    # counter loop
    while not rospy.is_shutdown():
        # COMPLETE HERE TO PUBLISH THE MESSAGE

        # apply the publishing rate
        rate.sleep()

if __name__ == '__main__':
    try:
        talker()
    except rospy.ROSInterruptException:
        pass
```

Save the code into `talker.py` file, then execute the node with the command:

```
python talker.py
```

In an other terminal use rostopic echo to display the message that you are publishing. When you manage to see your message you can move to the next section.

Subscriber node

This node called `listener` will subscribe to the topic `/counter` and print the message to the output. For this you will define a subscriber that will call a function each time a new message is received (i.e using a callback).

Complete the following code into `listener.py` file:

```
#!/usr/bin/env python
import rospy
from std_msgs.msg import Float32

def callback(counter):
    rospy.loginfo("counter value: %d", counter.data)

def listener():
    # COMPLETE HERE
    rospy.init_node(    )  # <-- COMPLETE HERE, warning each node must have an unique name

    # init the subscriber with the method signature: rospy.Subsciber(TOPIC_NAME, MESSAGE_TY
    sub = rospy.Subscriber(    )  # <-- COMPLETE HERE

    # spin() simply keeps python from exiting until this node is stopped
    rospy.spin()

if __name__ == '__main__':
    try:
        listener()
    except rospy.ROSInterruptException:
        pass
```

Execute the node with the command:

```
python listener.py
```

Part 3: controlling a robot with the keyboard

Write a node to control the **TurtleSim** robot using the keyboard. You will use: - the TurtleSim viewer with the command: `rosrun turtlesim turtlesim_node` - find the topic to publish on, with: `rosnode info NODE_NAME` - explore the message needed by the robot: `rostopic info TOPIC_NAME` - find the message specifications, with: `rosmsg show MSG_NAME`

Tips

- code for ↑ is key 65
- code for ↓ is key 66
- code for → is key 67
- code for ← is key 68
- you can access to the key code with: `python   import sys   import tty   tty.setcbreak(sys.stdin)   while True:   key = ord(sys.stdin.read(1))`