Lab #4: ROS2 using RCLPY in Julia

Abdelbacet Mhamdi
Senior-lecturer, Dept. of EE
ISET Bizerte — Tunisia
a-mhamdi

fadi manai
Dept. of EE
ISET Bizerte — Tunisia
fadimanai

ahmed mahmoudi
Dept. of EE
ISET Bizerte — Tunisia
MahmoudiEI2

You are required to carry out this lab using the REPL as in Figure 1.

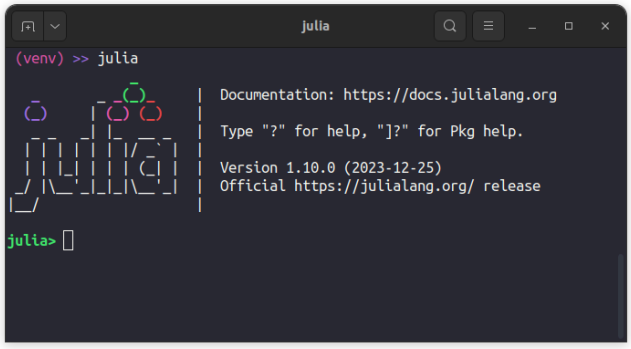

Figure 1: Julia REPL

in this lab they are two part in the first part ("Application") we gonna use also julia REPL to write down ROS2s codes as the figure (1) and in the second part ("clarafication") we gonna explain every commandes and her function
We begin first of all by sourcing our ROS2 installation as follows:

```
source /opt/ros/humble/setup.zsh
```

Secondly, we're going to open up a Julia terminal and write down the codes below. Alternatively, we can open it from our folder "infodev/codes/ros2".

Always start by sourcing ROS2 installation in any newly opened terminal.

Open a *tmux* session and write the instructions provided at your Julia REPL.

```
using PyCall
# Import the rclpy module from ROS2 Python
rclpy = pyimport("rclpy")
str = pyimport("std_msgs.msg")

# Initialize ROS2 runtime
rclpy.init()

# Create node
node = rclpy.create_node("my_publisher")
rclpy.spin_once(node, timeout_sec=1)

# Create a publisher, specify the message type and
the topic name
pub      =      node.create_publisher(str.String,
"infodev", 10)

# Publish the message `txt`
for i in range(1, 100)
    msg = str.String(data="Hello, ROS2 from Julia!
($(string(i)))")
    pub.publish(msg)
    txt = "[TALKER] " * msg.data
    @info txt
    sleep(1)
end

# Cleanup
rclpy.shutdown()
node.destroy_node()
```

The second program is the subscriber code. After writing down both programs, we need to execute each of them in a newly opened terminal. Then the subscriber will listen to the message broadcasted by the publisher

```
using PyCall

rclpy = pyimport("rclpy")
str = pyimport("std_msgs.msg")

# Initialization
rclpy.init()

# Create node
node = rclpy.create_node("my_subscriber")

# Callback function to process received messages
function callback(msg)
    txt = "[LISTENER] I heard: " * msg.data
    @info txt
end

# Create a ROS2 subscription
```

```
sub     =     node.create_subscription(str.String,
"infodev", callback, 10)

while rclpy.ok()
      rclpy.spin_once(node)
end
# Cleanup
node.destroy_node()
rclpy.shutdown()
```

The second program is the subscriber code. Once we've written down both programs, we'll need to execute each of them in a newly opened terminal. Afterward, the subscriber will listen to the message broadcasted by the publisher

```
source /opt/ros/humble/setup.zsh
rqt_graph
```

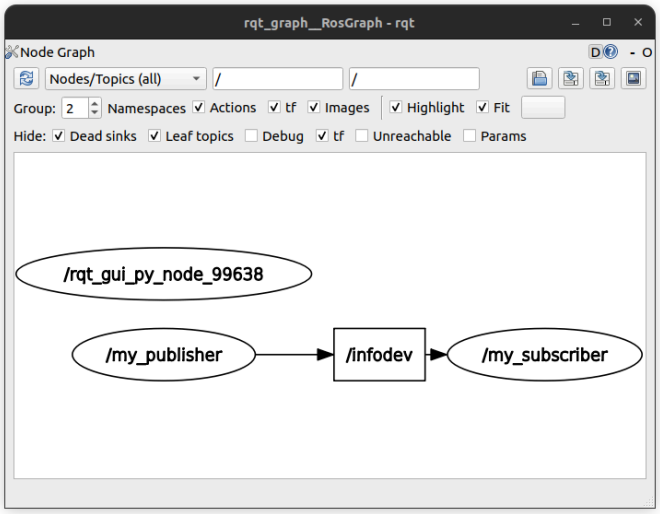

Figure 2: rqt_graph

After linking the publisher and the subscriber, the publisher will broadcast this message one hundred times to the node linked with the subscriber:
csharp Copy code [Info [TALKER] Hello, ROS2 from Julia!(1...100)] Then, the subscriber will respond in the node by:
css Copy code [Info [LISTENER] I heard Hello, ROS2 from Julia!(1...100)]
Figure 3 depicts the publication and reception of the message *"Hello, ROS2 from Julia!"* in a terminal. The left part of the terminal showcases the message being published, while the right part demonstrates how the message is being received and heard.

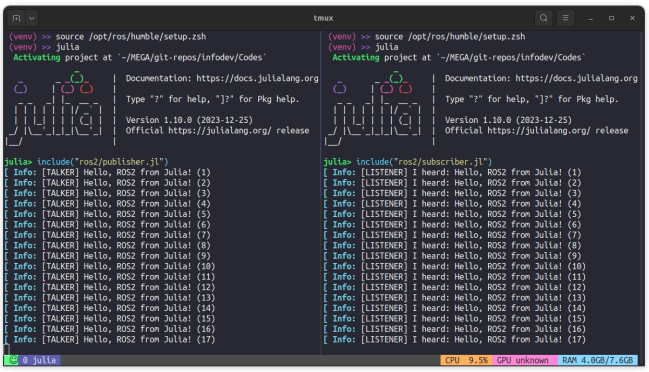

Figure 3: Minimal publisher/subscriber in ROS2

Figure 4 shows the current active topics, along their corresponding interfaces.

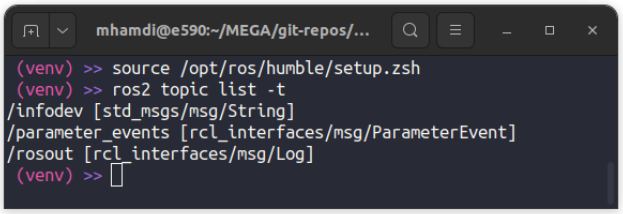

Figure 4: List of topics

First Program: Publisher Code

```
# Using PyCall to interface with Python
using PyCall

# Importing the rclpy module from ROS2 Python
rclpy = pyimport("rclpy")

# Importing the std_msgs.msg module from ROS 2
str = pyimport("std_msgs.msg")

# Initializing ROS2 runtime
rclpy.init()

# Creating a node named "my_publisher"
node = rclpy.create_node("my_publisher")

# Executing a single iteration of the ROS 2 event
loop within a given timeout period
rclpy.spin_once(node, timeout_sec=1)

# Creating a publisher within the ROS 2 node
pub      =      node.create_publisher(str.String,
"infodev", 10)

# Publishing messages to the topic
for i in 1:100
    msg = str.String(data="Hello, ROS2 from Julia!
($(string(i)))")
    pub.publish(msg)
```

```
    txt = "[TALKER] " * msg.data
    @info txt
    sleep(1)
end

# Shutting down ROS2 runtime and destroying the
node
rclpy.shutdown()
node.destroy_node()
```

Second Program: Subscriber Code

```
# Importing the rclpy module
rclpy = pyimport("rclpy")

# Importing the std_msgs.msg module
str = pyimport("std_msgs.msg")

# Creating a node named "my_subscriber"
node = rclpy.create_node("my_subscriber")

# Defining a callback function to handle received
messages
function callback(msg)
    txt = "[LISTENER] I heard: " * msg.data
    @info txt
end

# Creating a subscriber within the ROS 2 node
sub    =    node.create_subscription(str.String,
"infodev", callback, 10)

# Continuously spinning the ROS 2 node
while rclpy.ok()
    rclpy.spin_once(node)
end
```

These programs are written in Julia and utilize PyCall to interface with Python modules. The publisher code initializes the ROS2 runtime, creates a publisher node, and publishes messages to a specified topic. The subscriber code creates a subscriber node, defines a callback function to handle received messages, and continuously processes messages by spinning the node.