Lab #4: ROS2 using RCLPY in Julia

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

Mohammed khalil hannechi
Dept. of EE
ISET Bizerte — Tunisia
khalilhannechi

aziz chaieb
Dept. of EE
ISET Bizerte — Tunisia
azizchaiebc

. Introduction:
This lab consists of two parts: "Application" and "Clarification." In the first part, we'll utilize Julia REPL to implement ROS2 codes as depicted in Figure 1. The second part involves explaining each command and its function in detail.

Figure 1: Julia REPL

in this lab i can't simulate ROS2 with my laptop for that i'm gonna use the simulation picture in Images/ infodev folder

I. Application

First, we'll begin by installing ROS2. Then, we'll proceed by sourcing our ROS2 installation as follows:

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

.Secondly, we'll open a Julia terminal and write down the codes below. Alternatively, we can directly open them from our folder "infodev/codes/ros2"

The first programme is the publisher code

```
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 programme is the subscriber code

After writing both programs, we'll execute each in separate terminals. Subsequently, 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()
```

. To initiate the graphical tool rqt_graph and establish data flow between the publisher and subscriber, we must connect both to a node named “infodev,” as illustrated in Figure 2. This is accomplished by executing the following lines of code:

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

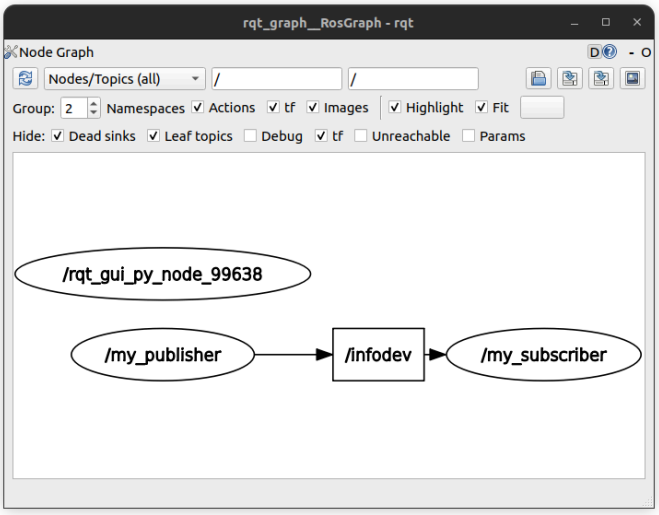

Figure 2: rqt_graph

. After linking the publisher and the subscriber, the publisher will transmit the designated message one hundred times to the node associated with the subscriber..

[Info [TALKER] Hello, ROS2 from Julia!(1...100)]

then the subscriber will respond, in the node ,by

[Info [LISTNER] I heard Hello, ROS2 from Julia!(1...100)]

as in figure 3

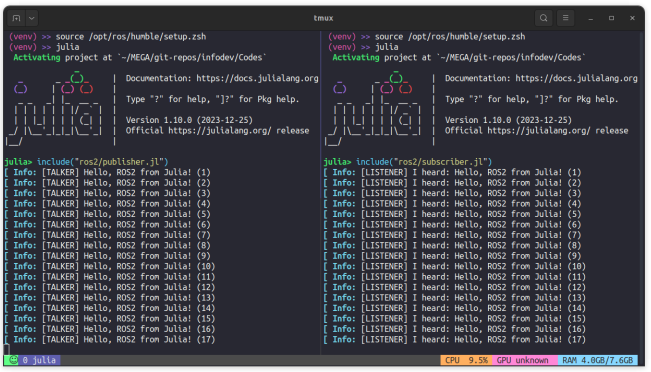

Figure 3: The communication between the publisher and the subscriber unfolds as follows

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

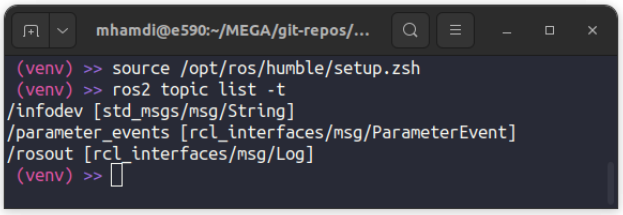

Figure 4: List of topics

II. Clarafication :

- in this part we gonna explain each code line and we gonna start with the pulisher code first :

The First programme is the spublisher code

```
using pycall
```

This package can come in handy when you aim to make use of Python’s extensive libraries or integrate Python-specific features into your Julia codebase.

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

In Julia, you can leverage PyCall to import the ‘rclpy ‘module from Python. rclpy serves as a Python client library for the Robot Operating System (ROS) 2.

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

import the std_msgs.msg module from ROS 2 into Julia

```
rclpy.init()
```

Initialize ROS2 runtime

```
node = rclpy.create_node("my_publisher")
```

create a node named “my_publisher” using the rclpy

```
rclpy.spin_once(node, timeout_sec=1)
```

. Execute a single iteration of the ROS 2 event loop within a specified timeout period using the ‘spin_once’ function from the ‘rclpy ‘module.

```
pub = node.create_publisher(str.String,
"infodev", 10)
```

- In Python, use the create_publisher function from the rclpy module to instantiate a publisher within a ROS 2 node named “node.” This publisher will be configured to publish messages of a specific type on a defined topic with a designated name.

```
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
```

. Using PyCall in Julia, establish a publisher node to communicate with a ROS 2 system. This node will publish messages to a topic named “infodev.” Each message will contain a string with the text “Hello, ROS2 from Julia!” accompanied by an incrementing number ranging from 1 to 99.

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

Deliting the rcply and destroy the node

The second programm : the subscriber code

```
rclpy = pyimport("rclpy")
```

Utilize PyCall in Julia to import the rclpy module from Python. This module, integral to the Robot Operating System 2 (ROS 2) ecosystem, offers capabilities for crafting ROS 2 nodes, publishers, subscribers, and additional functionalities.

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

Using PyCall in Julia, import the std_msgs.msg module from ROS 2. This module comprises message types frequently employed in ROS 2, including standard messages for various data types such as strings, integers, floats, and more.

```
node = rclpy.create_node("my_subscriber")
```

creat a node called my subscriber in a specific topic

```
function callback(msg)
    txt = "[LISTENER] I heard: " * msg.data
    @info txt
end
```

Create a callback function in Julia that executes when messages are received by a subscriber. This function will print the received message data, prefixed with an indication that it was received by the listener node

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

In Python, within a ROS 2 node named “node,” utilize the create_subscription function from the rclpy module to instantiate a subscriber. This subscriber will subscribe to messages of type std_msgs.msg.String on the topic “infodev” and trigger the callback function upon receiving messages.

```
while rclpy.ok()
      rclpy.spin_once(node)
end
```

. Develop a loop in Python that iteratively spins the ROS 2 node until the ROS 2 context, checked via ‘rclpy.ok()’, remains valid. This loop ensures sustained processing of messages and callbacks by the node for as long as the ROS 2 context remains valid.