Homework 1: The Telephone Game RO11 - Apprentissage pour la robotique

Gallego, Natalia
AST
ENSTA Paris
natalia.gallego@ensta-paris.fr

I. Introduction

In this ROS 1 project, the main goal is to implement communication between four nodes: **node A, node B, node C and node D**, each with its own publisher and subscriber. The challenge is to simulate a ”game of telephone”, where a message is transmitted from one node to another, with each node adding a letter to the original message.

This project not only helps to understand the flow of information across topics, but also to manage and organize multiple nodes within a single ROS package.

II. Shape of the node chain

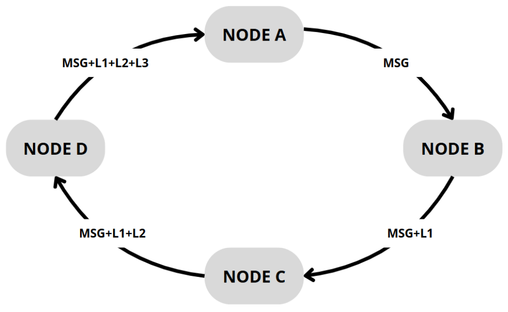


Fig. 1. Shape of the node chain

- **Node A** starts the chain by sending an initial message.
- The message is published to the topic, to which **node B** is subscribed. The **node B** modifies the message by adding a letter.
- **Node B** then publishes the new message in a topic, which is being listened to by **node C**. The **node C** receives the modified message and adds its letter.
- The updated message is published in the topic, to which **node D** is subscribed. The **node D** receives the message via the topic, and adds its letter.
- The final message is published in the topic, which is again listened to by **node A**. Then, **node A** prints the final message.

The communication chain forms a closed loop in which the message goes all the way around the four nodes, and each node modifies the message before passing it on to the next. This creates a continuous sequence of events, with a periodic stream every 2 seconds, always started by **node A**.

III. Nodes

We have two types of codes, the one for **node A** and the ones for **nodes B, C and D**. Each of these will be explained in order to understand how the system works.

It should be noted that for this project only one topic was used where the nodes publish their messages (**outgoing_LETTER**). Therefore all nodes can publish in the topic and all nodes can listen to all published messages.

To solve this, an identifier is sent within the message, which allows to know which node sent the message being received. In this way, when a node receives a message, it must identify if a message is relevant to it following the sequence **A - B - C - D**. If the message is sent by the node before it in the sequence, it will process the message according to the instructions of the task, if not, it will let the message pass.

A. Node A Code

Below is the pseudcode for node A:

Algorithm 1 Node A’s Message Communication in ROS

```
0: Start node ’node_A’.
0: Create a publisher on the ’outgoing_LETTER’ topic.
0: Create a subscriber to the ’outgoing_LETTER’ topic.
0: Define the secret message
0: Initialize a counter to 0.
0: Start a timer to publish the message every 2 seconds.
0: while ROS is running do
0:   if timer is triggered then
0:     Create a message with the format ”A: secret message - Count: counter”.
0:     Publish the message to ’outgoing_LETTER’.
0:     Increment the counter.
0:   end if
0:   if a message is received from the topic then
0:     if the message starts with ”D:” then
0:       Print the content of the message.
0:     end if
0:   end if
0: end while=0
```

In this case, **node A** publishes and receives messages in a game of ”telephone”. **Node A** sends a periodic message every 2 seconds and listens for messages back, checking to see if the message came from **node D**. If it does, it processes it and prints it out.

B. Nodes B, C and D Code

Below is the pseudcode for nodes B, C and D:

Algorithm 2 Nodes B, C and D Message Communication

```
0: Start node 'node_B, node_C or node_D'.
0: Create a publisher on the 'outgoing_LETTER' topic.
0: Create a subscriber to the 'outgoing_LETTER' topic.
0: while ROS is running do
0:   if a message is received from the topic then
0:     if the message starts with "A/B/C:" then
0:       Strip the "A/B/C:" prefix from the message.
0:       Append the "B/C/D:" prefix to the message.
0:       Add the letter "B/C/D" to the end of the message.
0:       Publish the message to 'outgoing_LETTER'.
0:     end if
0:   end if
0: end while=0
```

This code defines the behavior of **nodes B, C** and **D** in the ROS 1 project. The nodes act as intermediaries in the game of ”telephone”, where they receive a message from a node before them, modify it and relay it over the same topic to the next node in the chain.

IV. Step by step

Before you begin, make sure you are running Ubuntu 20.04. If not, and you are running other Ubuntu generations, use Docker to perform the following steps. Also, make sure you have ROS installed.

This tutorial starts after running the container on your machine.

You will start by creating a ROS workspace, and then creating the necessary project files and folders within it. In addition to indicating where the ROS dependencies are located, so that the project can use them. To do this, enter the following instructions in the command line:

```
mkdir -p ros_workspace/src
cd ros_workspace/src

catkin_init_workspace

source /opt/ros/noetic/setup.bash
cd ros_workspace
catkin_make
```

Once that is done, we will create the package with the ROS basic structure, for our telephone application.

```
cd ros_workspace/src
catkin_create_pkg telephone std_msgs rospy
```

Inside our package we will go to **src** and create the files that will have the python code for each of our nodes. Do this for the 4 nodes. Using the nano function, we will be able to access the inside of our programs, and edit this to add the code that will make our nodes work. For this, use the pseudocodes presented previously as a development base.

```
cd telephone/src
touch a_node.py
chmod +x a_node.py
nano a_node.py
```

Returning to the base of our package, we will edit the **CMakeList.txt** file. So that it takes into account the files that we just created.

```
cd ..
nano CMakeLists.txt
```

For this we will add the following block of code inside our **CMakeList.txt** file.

```
catkin_install_python(PROGRAMS src/a_node.py
    src/b_node.py
    src/c_node.py
    src/d_node.py
  DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION}
)
```

To finish the configuration, just go to the **root** of the project and build it. Also, tell the terminal where to look for the binaries created when building the package.

```
cd ros_workspace
catkin_make
source devel/setup.bash
```

Now to execute the codes we must, first of all, execute the command necessary to initialize ROS.

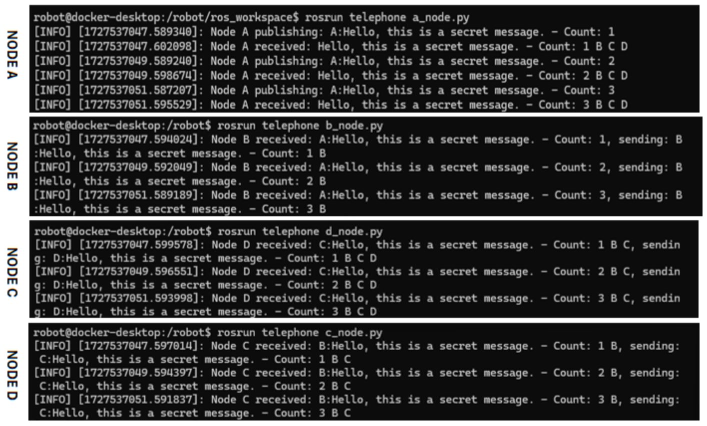
```
roscore
```

We proceed to open another 4 terminals, one for each node and enter the workspace created previously. Once inside, we proceed to run the code corresponding to each node in each terminal, to do this follow the following instructions.

```
source ros_workspace/devel/setup.bash
rosrun telephone a_node.py
```

It is recommended to run nodes B, C and D first, since node A is the one that starts the communication cycle.

V. Results



```
robot@docker-desktop:/robot/ros_workspace$ rosrun telephone a_node.py
[INFO] [1727537047.589340]: Node A publishing: A:Hello, this is a secret message. - Count: 1
[INFO] [1727537047.602098]: Node A received: Hello, this is a secret message. - Count: 1 B C D
[INFO] [1727537049.589240]: Node A publishing: A:Hello, this is a secret message. - Count: 2
[INFO] [1727537049.598674]: Node A received: Hello, this is a secret message. - Count: 2 B C D
[INFO] [1727537051.587207]: Node A publishing: A:Hello, this is a secret message. - Count: 3
[INFO] [1727537051.595529]: Node A received: Hello, this is a secret message. - Count: 3 B C D
```

```
robot@docker-desktop:/robot$ rosrun telephone b_node.py
[INFO] [1727537047.594024]: Node B received: A:Hello, this is a secret message. - Count: 1, sending: B
:Hello, this is a secret message. - Count: 1 B
[INFO] [1727537049.592049]: Node B received: A:Hello, this is a secret message. - Count: 2, sending: B
:Hello, this is a secret message. - Count: 2 B
[INFO] [1727537051.589189]: Node B received: A:Hello, this is a secret message. - Count: 3, sending: B
:Hello, this is a secret message. - Count: 3 B
```

```
robot@docker-desktop:/robot$ rosrun telephone d_node.py
[INFO] [1727537047.599578]: Node D received: C:Hello, this is a secret message. - Count: 1 B C, sendin
g: D:Hello, this is a secret message. - Count: 1 B C D
[INFO] [1727537049.596551]: Node D received: C:Hello, this is a secret message. - Count: 2 B C, sendin
g: D:Hello, this is a secret message. - Count: 2 B C D
[INFO] [1727537051.593998]: Node D received: C:Hello, this is a secret message. - Count: 3 B C, sendin
g: D:Hello, this is a secret message. - Count: 3 B C D
```

```
robot@docker-desktop:/robot$ rosrun telephone c_node.py
[INFO] [1727537047.597014]: Node C received: B:Hello, this is a secret message. - Count: 1 B, sending:
 C:Hello, this is a secret message. - Count: 1 B C
[INFO] [1727537049.594397]: Node C received: B:Hello, this is a secret message. - Count: 2 B, sending:
 C:Hello, this is a secret message. - Count: 2 B C
[INFO] [1727537051.591837]: Node C received: B:Hello, this is a secret message. - Count: 3 B, sending:
 C:Hello, this is a secret message. - Count: 3 B C
```

Fig. 2. Results

VI. GitHub

You can refer to the following reference to see the node codes for the project: GitHub/RO11