ROS2 Node Structure and Run Length Encoding

Pranav Mintri

26th October, 2025

1 Python Code - Run Length Encoding

```
i=0
word=input("Enter the word: ")
letters = []
numbers = []
j=1
while i<len(word)-1:
    if word[i]!=word[i+1]:
        letters.append(word[i])
        numbers.append(j)
        j=0
    j+=1
    i+=1
letters.append(word[len(word)-1])
numbers.append(j)
i=0
final_word=""
while i<len(letters):
    final_word+=(letters[i])+str(numbers[i])
    i+=1
print(final_word)
```

The above code is an example of how run length encoding can be done in Python.

2 ROS2 Node Structure

2.1 Creating `turtlesim_node`:

First, ROS2 must be sourced in the terminal in which we will be using it. This has been done by default.
Next, in order to create a node `turtlesim_node`, the following is to be run on the terminal:

```
ros2 run turtlesim turtlesim_node
```

```
pranavmintri@pranavmintri-750QHA:~$ ros2 run turtlesim turtlesim_node
Warning: Ignoring XDG_SESSION_TYPE=wayland on Gnome. Use QT_QPA_PLATFORM=wayland
 to run on Wayland anyway.
[INFO] [1761457775.388097288] [turtlesim]: Starting turtlesim with node name /tu
rtlesim
[INFO] [1761457775.391105157] [turtlesim]: Spawning turtle [turtle1] at x=[5.544
445], y=[5.544445], theta=[0.000000]
```

Figure 1: Creating `turtlesim_node`

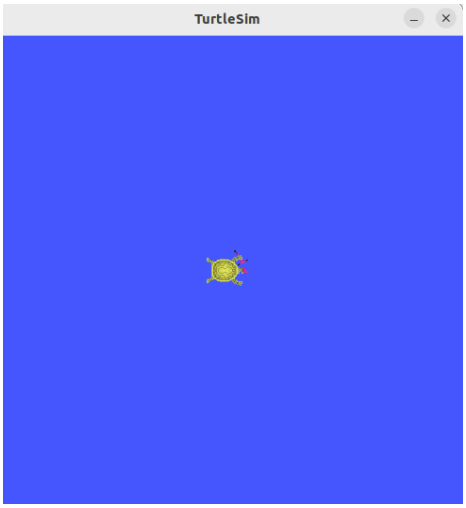

Figure 2: TurtleSim

This will open TurtleSim.

2.2 Creating `turtlesim_controller`:

This node will be subscribed to the topic `keystroke`, and will publish data to the topic `cmd_vel`. This is meant to communicate with the `turtlesim_node`, but its functioning is fundamentally different from the usual `teleop_turtle` node, in that it will not read data directly from the terminal window.
Hence, we need to create a new program (node) that integrates Run Length Encoding (the aforementioned Python code). It is supposed to subscribe to `keystroke`, read the string, and produce a "run-length-encoded" string. This can be published to `cmd_vel`.

First, we must create a package. To create a package in the `ros2_humble` workspace, we must first navigate to:

`~/ros2_humble/src`

Now, the following command can be used:

```
ros2 pkg create --build-type ament_python my_teleop_package rclpy std_msgs
```

Now, we must further navigate and open `my_teleop_package`.
Within that folder, another folder with the same name as the package (`my_teleop_package`) must have been created. It is here that we shall store the Python file for `turtlesim_controller`. The code itself, as instructed, will not be mentioned in this report, but the logic of RLE (Run Length Encoding) has already been described in the Python program above. Changes will also have to be made to the `setup.py` file inside the package, but that shall be mentioned in due course.

2.3 Creating `keyboard_publisher`:

This node is only meant to read input from the user (terminal window), and publish it to `keystroke`. The Python code for this node also has to be stored in the same folder as the Python code for `turtlesim_controller`.

2.4 Creating `telemetry_logger`:

This node will be subscribed to `keystroke` and `/turtle1/pose`. The latter topic is for the communication of the position of the turtle, and the former is for the keystrokes entered in the terminal window by the user.
Essentially, the position and the user commands will be displayed on this node. The Python code for this node also has to be stored in the same folder as the Python code for `turtlesim_controller` and `keyboard_publisher`.

2.5 Editing the setup and XML files:

We have created three custom nodes. Now, the setup file needs to be edited, to incorporate all three nodes. The part that needs to be edited is as follows:

```
entry_points={
    'console_scripts': [
        # The publisher node
        'keyboard_publisher = my_teleop_package.keyboard_publisher:main',
        # The controller node
        'turtlesim_controller = my_teleop_package.command_translator_node:main',
        # The logger node
        'telemetry_logger=my_teleop_package.telemetry_logger_node:main',
    ],
},
```

As is visible, the general format is:
`'<node_name>=<package_name>.<file_name>:main'`.
A few additions need to be made to the package XML file, that was created by default while creating the package. The following dependencies need to be added:

```
<depend>geometry_msgs</depend>
<depend>turtlesim</depend>
```

These are required for the telemetry logger.

2.6 Running the nodes:

`turtlesim_node` is already running. Now, we shall open a new terminal window and first of all, build/rebuild our package using symlinks. This may be redundant at times, but notwithstanding, we shall execute the following command.

```
colcon build --packages-select my_teleop_package --symlink-install
```

Now, this window must be closed, and a new window must be opened so that the new environment is automatically sourced.
Now, in this new window, to run `turtlesim_controller`, we shall type:

```
ros2 run my_teleop_package turtlesim_controller
```

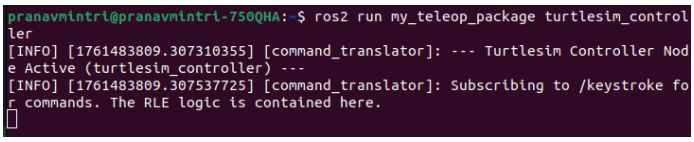

Figure 3: `turtlesim_controller`

Similarly, on a new window, type:

```
ros2 run my_teleop_package keyboard_publisher
```

Here, we can type the commands, like 'w' and 'd', multiple times if required.
The general format is quite clear.

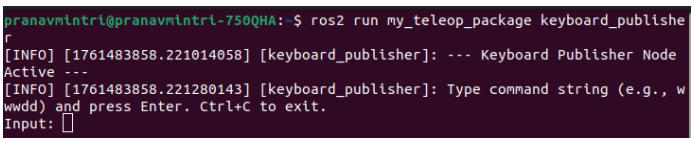

Figure 4: `keyboard_publisher`

```
ros2 run <package_name> <node_name>
```

Thus, in another window, we can type:

```
ros2 run my_teleop_package telemetry_logger
```

```
pranavmintri@pranavmintri-750QHA:~$ ros2 run my_teleop_package telemetry_logger
[INFO] [1761485976.685527547] [telemetry_logger]: --- Telemetry Logger Node Acti
ve ---
[INFO] [1761485976.686757356] [telemetry_logger]: Listening for raw commands on
/keystroke.
[INFO] [1761485976.687886043] [telemetry_logger]: Listening for turtle pose on /
turtle1/pose.
[INFO] [1761485976.689930140] [telemetry_logger]: [TELEMETRY LOG] Turtle Positio
n: X=5.54, Y=5.54, Theta=0.0
[INFO] [1761485976.705708423] [telemetry_logger]: [TELEMETRY LOG] Turtle Positio
n: X=5.54, Y=5.54, Theta=0.0
[INFO] [1761485976.722681676] [telemetry_logger]: [TELEMETRY LOG] Turtle Positio
n: X=5.54, Y=5.54, Theta=0.0
```

Figure 5: `telemetry_logger`

2.7 Visualizing the node/topic structure:

All the nodes and their corresponding subscribing/publishing capabilities to various topics can be visualized from the following diagram. This diagram can be accessed by simply typing `rqt` in a new terminal window.
We navigate to **Plugins**, then **Introspection**, and then **Node Graph**, following which, the following node graph can be observed:

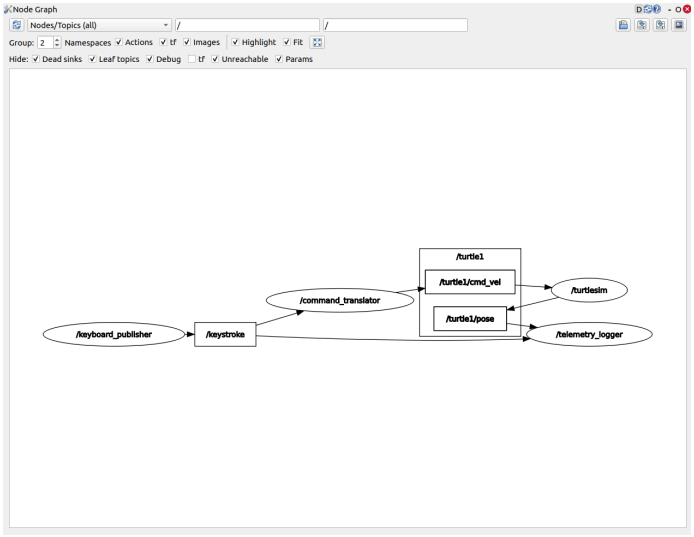

Figure 6: Visualizing the Node/Topic Structure Using `rqt`

2.8 A Demonstration:

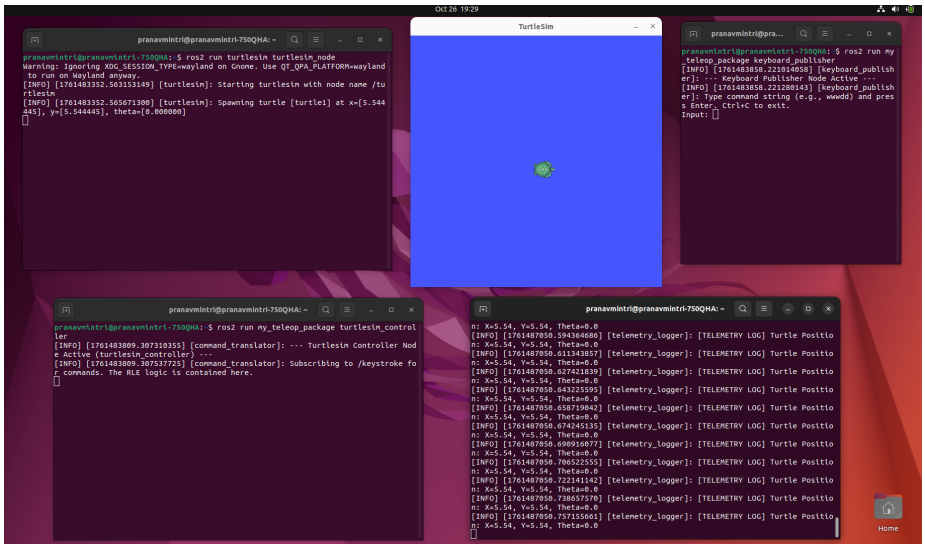

Figure 7: Initial Condition

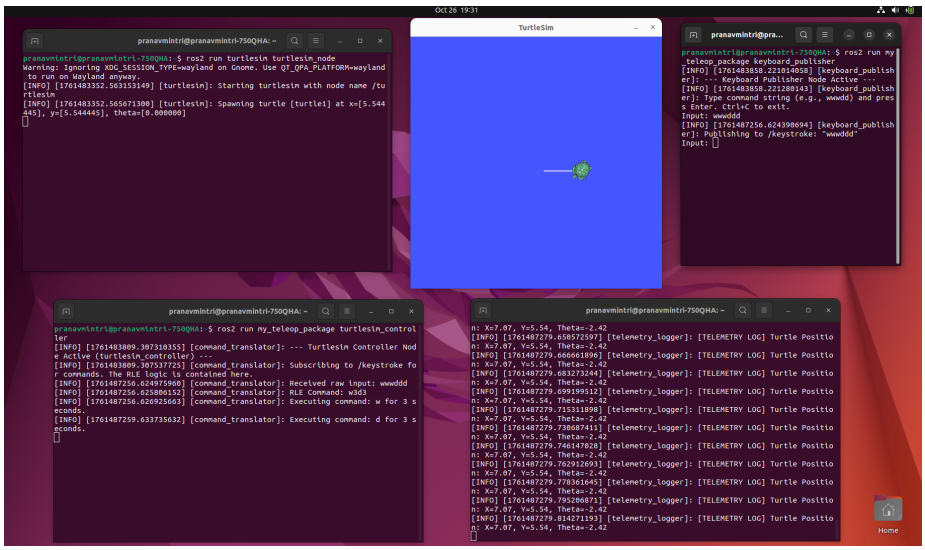

Figure 8: Final Condition (after entering a command wwwddd)