Getting Started With ROS2

Madhav Agrawal

January 16, 2024

1 Introduction

ROS (Robot Operating System) is an open-source middleware framework for developing and controlling robots. It has very helpful tools for development, debugging, and simulation, fostering a collaborative and extensive developer community.

2 How ROS Works?

2.1 What is a Package?

ROS uses packages to organize its programs. Every ROS program that you want to create or execute is organized in a package. You can think of a package as all the files that a specific ROS program contains; all its CPP files, python files, configuration files, compilation files, launch files, and parameter files. A package contains launch files, scripts folder and package.xml (dependencies).

2.2 What are Nodes?

One of the primary purposes of ROS is to facilitate communication between the ROS nodes. Every program in ROS is called a node. Every independent task can be separated into nodes which communicate with each other through channels. These channels are also known as topics.

For example, one node can capture the images from a camera and send the images to another node for processing. After processing the image, the second node can send a control signal to a third node for controlling a robotic manipulator in response to the camera view. Nodes may publish messages on a particular topic or subscribe to a topic to receive information.

2.3 What are Topics?

A topic is simply a medium of exchange of data (like a channel). Some nodes called Publishers can publish data on the topic, some nodes called Subscribers can subscribe to the data on the topic. A topic has a message type (similar to the data type of a variable). All publishers and subscribers on this topic must publish/subscribe data of the associated message type.

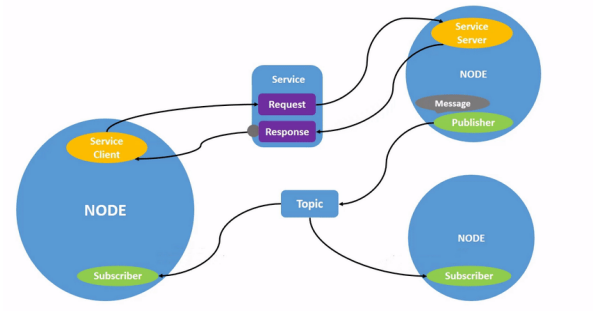

Figure 1: Communication between nodes

When a node wants to publish something, it will inform the ROS master. When another node wants to subscribe to a topic, it will ask the ROS master form where it can get the data.

```
rqt_graph: Reveals communication between nodes through topics
ros2 topic echo: Displays real-time data published on a specific topic.
ros2 topic pub: Enables manual publication on desired topics.
```

3 Working with ROS

3.1 Creating Workspace

A workspace is a directory containing ROS 2 packages. Before using ROS 2, it's necessary to source your ROS 2 installation workspace in the terminal you plan to work in. This makes ROS 2's packages available for you to use in that terminal.

Best practice is to create a new directory for every new workspace. To create workspace create a new folder with src folder in it (for storing pacakges). Change directory and set it to current directory.

```
mkdir -p ~/mrt_ws/src
cd ~/erc_ws/src
```

3.2 Colcon Build

Colcon is a build tool which performs the task of building a set of packages with a single invocation. Using it we can build a package and create the additional necessary files, using just one command. For installing and configuring Colcon run the command:

```
sudo apt install python3-colcon-common-extensions
```

From the root of your workspace (mrt_ws), you can now build your packages using the command:

```
cd ..
colcon build
```

Do colcon build every time when you make change in the package. It edits all the other files accordingly.

3.3 Create A New Package

Package creation in ROS 2 uses ament as its build system and colcon as its build tool. A package can be created using either CMake or Python. Packages should be created in the src directory, not the root of the workspace. So, navigate into mrt_ws/src, and run the package creation command:

```
cd mrt_ws/src
ros2 pkg create --build-type ament_python package1
```

3.4 Create Python Executable Files

Navigate into scripts folder of package and create a new python file, make it executable and push the codes to it.

```
cd ~
cd mrt_ws/src/package1/package1
touch talker.py
chmod +x talker.py #Making the python file executable
```

3.5 Simple Publisher

3.5.1 To open VS Code

```
code . - For current directory
code .. - For parent directory
```

3.5.2 Code for a simple publisher node

```
import rclpy
from std_msgs.msg import String

def timer_callback(timer, i):
    # Create a String message
    msg = String()
    msg.data = 'Hello World'

    # Publish the message using the global publisher
    publisher.publish(msg)

    # Print a message indicating what is being published
    print('Publishing: "%s"' % msg.data)

def main(args=None):
    # Initialize the ROS 2 system
    rclpy.init(args=args)

    # Create a ROS 2 node named 'minimal_publisher'
    node = rclpy.create_node('minimal_publisher')

    # Create a global publisher for the 'topic' with a message type of String
    global publisher
    publisher = node.create_publisher(String, 'topic', 10)

    # Set the timer period to 0.5 seconds
    timer_period = 0.5

    # Initialize a counter variable
    i = 0

    # Create a timer that calls the timer_callback function every timer_period seconds
    timer = node.create_timer(timer_period, lambda: timer_callback(timer, i))

    # Increment the counter
    i += 1

    try:
        # Start spinning the ROS 2 node
        rclpy.spin(node)
    finally:
        # Destroy the node explicitly when done spinning
        # (optional - otherwise it will be done automatically
        # when the garbage collector destroys the node object)
        node.destroy_node()

        # Shutdown the ROS 2 system
        rclpy.shutdown()

# Entry point to the script
if __name__ == '__main__':
    # Call the main function if this script is the main module
    main()
```

3.6 Simple Subscriber

3.6.1 Make a new file

```
cd ~
cd erc_ws/src/week0_tutorials/week0_tutorials
touch listener.py
chmod +x listener.py
```

3.6.2 Code for subscriber

```
import rclpy
from std_msgs.msg import String

def listener_callback(msg):
    print('I heard: "%s"' % msg.data)

def main(args=None):
    # Initialize the ROS 2 system
    rclpy.init(args=args)

    # Create a ROS 2 node named 'minimal_subscriber'
    node = rclpy.create_node('minimal_subscriber')

    # Create a subscription to the 'topic' with a message type of String
    subscription = node.create_subscription(String, 'topic', listener_callback, 10)

    # Prevent unused variable warning
    subscription

    try:
        # Start spinning the ROS 2 node
        rclpy.spin(node)
    finally:
        # Destroy the node explicitly when done spinning
        # (optional - otherwise it will be done automatically
        # when the garbage collector destroys the node object)
        node.destroy_node()

        # Shutdown the ROS 2 system
        rclpy.shutdown()

# Entry point to the script
if __name__ == '__main__':
    # Call the main function if this script is the main module
    main()
```

3.7 Additional Changes To Do

3.7.1 Add Dependencies

Open package.xml contained in mrt_ws/src/package1 with your text editor. Add the following dependencies corresponding to your node's import statements:

```
<exec_depend>rclpy</exec_depend>
<exec_depend>std_msgs</exec_depend>
```

3.7.2 Add Entry Point

In setup.py file add the following lines to add entry point:

```
entry_points={
        'console_scripts': [
                'publisher = package1.talker:main',
                'subscriber = package1.listener:main',
        ],
},
```

3.7.3 Build And Run

```
cd ~
cd erc_ws

colcon build
source install/setup.bash

ros2 run week0_tutorials publisher
ros2 run week0_tutorials subscriber
```

3.8 Launch File

ROS 2 Launch files allow you to start up and configure a number of executables containing ROS 2 nodes simultaneously. Create a new directory in mrt_ws/src/package1 to store your launch files:

3.8.1 Creating A New Python File

```
cd src/package1
mkdir launch

cd launch
touch pubsub.launch.py
code .
```

Add the following code.

```
from launch import LaunchDescription
from launch_ros.actions import Node

def generate_launch_description():
    return LaunchDescription([
        Node(
            package='package1',
            executable='publisher',
        ),
        Node(
            package='package1',
            executable='subscriber',
        ),
    ])
```

3.8.2 Modifying Setup file

Now add the following line in setup.py in the data_files

```
(os.path.join('share', package_name, 'launch'), glob(os.path.join('launch', 'pubsub.launch.py'))),
```

and add the following on the top of setup.py

```
import os
from glob import glob
```

On executing ros2 launch package1 pubsub.launch.py, we will be able to see Publisher and Subscriber in the list of Nodes.

Thankyou!! That's Enough For Now!!