Practical Work 3: ROS 2 Intermediate Level

IA712: Mobile Robotics

Zhi Yan

1 Objective

The goal of this session is to learn how to manage a proper ROS 2 development workflow. By the end, you will be able to:

- Create a custom development workspace.
- Create your own ROS 2 packages.
- Define and use custom message interfaces.
- Write a launch file to start a multi-node system.

2 Creating a Workspace and a Package

First, we'll create a workspace and a Python package to hold our nodes.

1. Open a new terminal. Create a workspace directory, including the `src` subdirectory.

```
mkdir -p ~/ros2_ws/src
cd ~/ros2_ws/src
```

2. Inside `src`, create a new Python package named `robot_pkg`.

```
ros2 pkg create --build-type ament_python robot_pkg
```

This creates a new folder `robot_pkg` with the basic structure for a Python package.

3 Creating a Custom Message Interface

Let's create a custom message to report a robot's hardware status.

1. We need a separate package just for our custom interfaces. This is a common practice.

```
cd ~/ros2_ws/src
ros2 pkg create --build-type ament_cmake interface_pkg
```

Note: Interface packages must be of build type `ament_cmake`.

2. Inside this new package, create a `msg` directory and a file named `HardwareStatus.msg`.

```
cd interface_pkg
mkdir msg
touch msg/HardwareStatus.msg
```

3. Open `msg/HardwareStatus.msg` in a text editor and add the following content. This defines the structure of our message.

```
int64 temperature_celsius
bool are_motors_ready
string debug_message
```

4. Now, we must edit `package.xml` and `CMakeLists.txt` in `interface_pkg` to build these messages.

   Add these lines to `interface_pkg/package.xml` inside the `<package>` tags:

```
<build_depend>rosidl_default_generators</build_depend>
<exec_depend>rosidl_default_runtime</exec_depend>
<member_of_group>rosidl_interface_packages</member_of_group>
```

   Add these lines to `interface_pkg/CMakeLists.txt`:

```
find_package(rosidl_default_generators REQUIRED)

rosidl_generate_interfaces(${PROJECT_NAME}
"msg/HardwareStatus.msg"
)
```

   Make sure to place these lines before the `ament_package()` call at the end of the file.

4 Creating the Publisher and Subscriber Nodes

Now let's create two Python nodes in our `robot_pkg` to use this new message.

1. Create a file named `/ros2_ws/src/robot_pkg/robot_pkg/hardware_status_publisher.py` with the following publisher code:

```
import rclpy
from rclpy.node import Node
from interface_pkg.msg import HardwareStatus # Import custom
message

class HardwareStatusPublisher(Node):
def __init__(self):
super().__init__('hardware_status_publisher')
self.publisher_ = self.create_publisher(HardwareStatus, '
hardware_status', 10)
self.timer = self.create_timer(1.0, self.timer_callback)
self.temp_ = 65

def timer_callback(self):
msg = HardwareStatus()
msg.temperature_celsius = self.temp_
msg.are_motors_ready = True
msg.debug_message = f"Temperature is {self.temp_} C"
self.publisher_.publish(msg)
self.get_logger().info(f'Publishing: "{msg.debug_message}"')
self.temp_ += 1

def main(args=None):
rclpy.init(args=args)
node = HardwareStatusPublisher()
```

```
    rclpy.spin(node)
    rclpy.shutdown()

    if __name__ == '__main__':
    main()
```

2. Create a subscriber node in `/ros2_ws/src/robot_pkg/robot_pkg/hardware_status_subscriber.py`:

```
    import rclpy
    from rclpy.node import Node
    from interface_pkg.msg import HardwareStatus

    class HardwareStatusSubscriber(Node):
    def __init__(self):
    super().__init__('hardware_status_subscriber')
    self.subscription = self.create_subscription(
    HardwareStatus,
    'hardware_status',
    self.listener_callback,
    10)

    def listener_callback(self, msg):
    self.get_logger().info(f'I heard: temp={msg.temperature_celsius},
   motors_ready={msg.are_motors_ready}')

    def main(args=None):
    rclpy.init(args=args)
    node = HardwareStatusSubscriber()
    rclpy.spin(node)
    rclpy.shutdown()

    if __name__ == '__main__':
    main()
```

3. Edit `/ros2_ws/src/robot_pkg/setup.py` to add the entry points for these nodes. The `console_scripts` section should look like this:

```
    'console_scripts': [
      'status_publisher = robot_pkg.hardware_status_publisher:main',
      'status_subscriber = robot_pkg.hardware_status_subscriber:main',
    ],
```

4. Add the dependency on our interface package to `/ros2_ws/src/robot_pkg/package.xml`:

```
    <exec_depend>interface_pkg</exec_depend>
```

5 Building and Running the System

1. Navigate to the root of your workspace and build all packages.

```
    cd ~/ros2_ws
    colcon build
```

The build should succeed. It will build `interface_pkg` first, then `robot_pkg`.

2. Source the local setup file to make the new packages available.

```
source install/setup.bash
```

3. In **Terminal 1**, run the publisher:

```
ros2 run robot_pkg status_publisher
```

4. In **Terminal 2** (remember to source the workspace again!), run the subscriber:

```
# Don't forget to source first!
source ~/ros2_ws/install/setup.bash
ros2 run robot_pkg status_subscriber
```

You should see the subscriber printing the messages from the publisher.

6 Creating a Launch File

Finally, let's create a single launch file to run both nodes.

1. Create a launch directory and the file itself.

```
cd ~/ros2_ws/src/robot_pkg
mkdir launch
touch launch/robot_status.launch.py
```

2. Add the following content to `robot_status.launch.py`:

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

def generate_launch_description():
return LaunchDescription([
  Node(
  package='robot_pkg',
  executable='status_publisher',
  name='my_status_publisher'
  ),
  Node(
  package='robot_pkg',
  executable='status_subscriber',
  name='my_status_subscriber'
  ),
])
```

3. We need to tell the build system to install this launch file. Open `setup.py` and ensure the entire file looks like the following, which correctly finds and installs launch files.

```
import os
from glob import glob
from setuptools import setup

package_name = 'robot_pkg'

setup(
name=package_name,
version='0.0.0',
packages=[package_name],
```

```
    data_files=[
      ('share/ament_index/resource_index/packages',
      ['resource/' + package_name]),
      ('share/' + package_name, ['package.xml']),
      (os.path.join('share', package_name, 'launch'), glob('launch/*.
    launch.py'))
    ],
    install_requires=['setuptools'],
    zip_safe=True,
    maintainer='user',
    maintainer_email='user@todo.todo',
    description='TODO: Package description',
    license='TODO: License declaration',
    tests_require=['pytest'],
    entry_points={
      'console_scripts': [
        'status_publisher = robot_pkg.hardware_status_publisher:main',
        'status_subscriber = robot_pkg.hardware_status_subscriber:main
    ',
      ],
    },
    )
```

4. Build and run!

```
cd ~/ros2_ws
colcon build
source install/setup.bash
ros2 launch robot_pkg robot_status.launch.py
```

You should now see the output from both nodes in a single terminal.

This concludes the practical session. You have successfully created, built, and launched a custom multi-node ROS 2 system!