Robot Operating System

ROS Launch

Lecture No. 9

Dr. Eng. Essa Alghannam

- `roslaunch` is a powerful command-line tool in the Robot Operating System (ROS) that allows user to easily start and manage multiple ROS nodes simultaneously. Instead of starting each node individually using commands like `rosrun`.
- `roslaunch`: specify all the nodes we need in a single XML file (a launch file).

A simple launch file (e.g., `my_launch.launch`) might look like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<launch>
  <node pkg="my_package" type="node1" name="node1" output="screen"/>
  <node pkg="my_package" type="node2" name="node2" output="screen"/>
  <param name="my_parameter" value="10"/>
</launch>
```

[Start a node named "node1" from the executable `node1` in the `my_package` package, sending the output to the screen.]

[Set a parameter named "my_parameter" to the value "10," which will be available to any nodes that subscribe to it.]

1 Example:

- Inside the package, create a folder named “launch”.
- Inside the folder, create a file named `turtlesim.launch` (we can choose a different name, but the `.launch` extension is important). The suitable location within the ROS workspace should be, for example:

`mycatkin_ws/src/myturtlepackage/launch/turtlesim.launch`.

- Add the following content:

```
<?xml version="1.0" encoding="UTF-8"?>
<launch>

  <!-- Start the turtlesim node -->
  <node pkg="turtlesim" type="turtlesim_node" name="turtlesim"/>

  <!-- Optionally, start the keyboard teleop node -->
  <node pkg="turtlesim" type="turtle_teleop_key" name="teleop_turtle">
    <param name="scale_linear" value="2.0"/> <!-- Adjust speed -->
    <param name="scale_angular" value="2.0"/> <!-- Adjust turning speed -->
  </node>

</launch>
```

1- `<launch>` and `</launch>`: These tags enclose the entire launch file.

2- `<node>`: This tag defines a ROS node to be launched.

a) `pkg`: Specifies the ROS package containing the node (e.g., `"turtlesim"`).
b) `type`: Specifies the executable file within the package (e.g., `"turtlesim_node"`).
c) `name`: Assigns a unique name to the node. This is crucial for identification and avoiding name conflicts.

3- `<param>` (optional): These tags are used to set parameters for the launched nodes. In this case, they adjust the speed of the turtle controlled by the keyboard.

- ✓ scale_linear default value: 1.0
- ✓ scale_angular default value: 1.0
- ✓ linear_speed:Default Value: 1.0 (units/sec)
- ✓ angular_speed: Default Value: 1.0 (radians/sec)

These are scaling factors or multipliers that determine how aggressively the turtle_teleop_key node translates the keyboard input (like pressing arrow keys) into movement commands for the turtlesim turtle.

Finally, run it:

```
roslaunch myturtlepackage turtlesim.launch
```

In case of error, try:

chmod +x ~/mycatkin_ws/src/myturtlepackage/launch/turtlesim.launch

2 Another Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<launch>
  <node pkg="turtlesim" type="turtlesim_node" name="turtlesim"/>

  <node pkg="turtlesim" type="turtle_teleop_key" name="teleop_turtle">
    <!-- Base speeds for the node -->
    <param name="linear_speed" value="3.0"/>    <!-- Default is 1.0 -->
```

```
    <param name="angular_speed" value="1.5"/>   <!-- Default is 1.0 -->

    <!-- Scaling factors for keyboard input -->
    <param name="scale_linear" value="2.0"/>
    <param name="scale_angular" value="2.0"/>
  </node>

</launch>
```

3 Another example:

Create a file named `my_publisher.py` in the ROS package's `src` directory.
~/mycatkin_ws/src/myturtlepackage/src/my_publisher.py

```
#!/usr/bin/env python3
import rospy
from std_msgs.msg import String
def talker():
    rospy.init_node('my_publisher', anonymous=True) # Initialize the node
    pub = rospy.Publisher('my_topic', String, queue_size=10) # Create a publisher
    rate = rospy.Rate(1) # 1hz
    while not rospy.is_shutdown():
        hello_str = "hello world %s" % rospy.get_time()
        # rospy.get_time()returns the current time as a floating-point number
        #representing seconds since the ROS master started.
        rospy.loginfo(hello_str)
        pub.publish(hello_str) # Publish the message
        rate.sleep()

if __name__ == '__main__':
    try:
        talker()
    except rospy.ROSInterruptException:
        pass
```

~/mycatkin_ws/src/myturtlepackage/src/my_publisher.py

Edit CMAKELISTS.txt file

```
catkin_install_python(PROGRAMS src/ my_publisher.py
DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION})
```

Then

```
cd ~/mycatkin_ws/src/myturtlepackage /src
chmod +x my_publisher.py
```

Modify the launch file:

`mycatkin_ws/src/myturtlepackage/launch/turtlesim.launch`.

```
<?xml version="1.0" encoding="UTF-8"?>
<launch>
  <!-- Launch turtlesim for visualization -->
```

```
  <node pkg="turtlesim" type="turtlesim_node" name="sim"/>
  <!-- Launch the Python publisher node -->
  <node pkg="myturtlepackage" type="my_publisher.py" name="my_publisher"
output="screen"/>
</launch>
```

- `output="screen"`: This option controls where the output of the node (standard output and standard error streams) is directed. `screen` means the output (e.g., `rospy.loginfo` messages) will be printed to the terminal where we run the `roslaunch`. Other options include `log` (which sends the output to ROS log files) or `none` (which suppresses output entirely).

Finally:

cd ~/mycatkin_ws

catkin_make

roslaunch myturtlepackage turtlesim.launch

4 Another example:

~/catkin_ws/src/

- **myROSpackage**/
  - srv/
    - ✓ AddTwoInts.srv
  - src/
    - ✓ add_two_ints_server.py
    - ✓ add_two_ints_client.py
  - launch/
    - ✓ add_example.launch
  - include
  - CMakeLists.txt
  - package.xml