Lecture 06.03 Running and launching ROS nodes

Let's fire up some ROS nodes! Technically, we could `cd` around our filesystem, find packages, and start nodes with[4]

```
python <filename>.py
```

However, this is highly inconvenient. The `rosbash` package includes several utilities to improve this experience. Install it with the following.

```
sudo apt install rosbash
```

Reload your shell with `exec $SHELL`.

First, we might want to list files in an installed ROS package by simply executing, in any directory, *`rosls`* as follows. **rosls**

```
rosls <package_name>
```

Second, we might want to change to the directory of an installed ROS package with, in any directory, *`roscd`* as follows. **roscd**

```
roscd <package_name>
```

Third, there's tab completion. Terminal itself has *tab completion*: in any directory with a subdirectory named `foo`, type `cd fo<tab>`. It's a sort of autocompletion. ROS itself has this for its commands like `roscd`. Try starting to to type `roscd rospy_tutorials` and hit tab. If there's more than one matching package, double-tap tab to get a list. **tab completion**

There are a couple others that we'll explore in the following sections: `rosrun` and `roslaunch`.

06.03.1 Running ROS nodes

In this section, we'll start a few nodes, mostly from the `rospy_tutorials` package, installed with the following command.

```
sudo apt install ros-melodic-ros-tutorials
```

[4]For the curious, some nodes we'll be starting in a second could be started by navigating to `/opt/ros/melodic/share/rospy_tutorials/001_talker_listener` and executing, say, `python talker.py`.

As usual, after installation, `exec $SHELL` Before we start any nodes, we need a `roscore` service started.

```
roscore
```

rosrun

Now open a fresh terminal. We'll start our first "real" node with the `rosrun` command.

```
rosrun rospy_tutorials talker
```

In general, the syntax is as follows.

```
rosrun <package_name> <program_filename> [args]
```

So `talker.py` is run and should start printing something like the following every ten milliseconds.

```
[INFO] [1585538656.490473]: hello world 1585538656.49
[INFO] [1585538656.591393]: hello world 1585538656.59
[INFO] [1585538656.691669]: hello world 1585538656.69
```

This `talker` node is publishing `hello world <time>` on topic `chatter`. In a new terminal window, let's start a node to listen to the topic `chatter`: the `listener` node.

```
rosrun rospy_tutorials listener
```

This should give us something like the following.

```
[INFO] [1585542073.580711]: /listener_6552_1585542070720I heard
↪  hello world 1585542073.58
[INFO] [1585542073.682800]: /listener_6552_1585542070720I heard
↪  hello world 1585542073.68
[INFO] [1585542073.780337]: /listener_6552_1585542070720I heard
↪  hello world 1585542073.78
```

The ROS graph we just built is considered the "hello world" of ROS and is depicted in Figure 06.4.

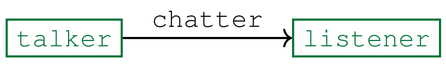

Figure 06.4: the `talker-listener` ROS graph with topic `chatter`.

You can generate similar ROS graph representations with the following, in a new Terminal.

```
rqt_graph
```

When you're satisfied, *stop* each node with ctrl+C.

stop a node

06.03.2 Launching ROS nodes

It is inconvenient to manually `rosrun` every node for larger (i.e. typical) ROS graphs. *Launch files* have extension `.launch` and are collections of node information that the command *`roslaunch`* operates on. The example `talker-listener` graph from above has a launch file `talker_listener.launch`.

launch files

`roslaunch`

Let's first find the launch file.

```
roscd rospy_tutorials/001_talker_listener
ls
```

```
listener  listener.py  README  talker  talker_listener.launch
↪  talker.py
```

Now let's print its contents.

```
cat talker_listener.launch
```

```
<launch>
  <node name="listener" pkg="rospy_tutorials" type="listener.py"
  ↪  output="screen"/>
  <node name="talker" pkg="rospy_tutorials" type="talker.py"
  ↪  output="screen"/>
</launch>
```

The `pkg` parameter for each `node` tag specifies the package from which the node comes; the `type` tag, the Python file; the `output` tag is often `"screen"` so that the node outputs to the console (instead of just a log file). The `name` tag may at first seem superfluous. However, it is very important: distinct `names` can be given to the same node `type`. For instance, two `listener.py` nodes can be launched with distinct `names`. This is one way of separating what is called the *namespace* of a ROS graph.

namespace

From any directory, the `talker-listener` graph can be launched with the following call to the launch file.

```
roslaunch rospy_tutorials talker_listener.launch
```

We should get the same results as our manual (`rosrun`) method above.