Robot Operating System

Lab 1: Packages, launch files, parameters and topic remapping

1 Goals

In this lab we will see the main tools to analyze nodes and topics. A ROS 2 package will be used to move Baxter in a simulation environment.

1.1 Using the terminals

In ROS, a lot of commands have to be run from the terminal. Each terminal should be configured either to run ROS 1 things, or ROS 2 things. A simple shortcut allows changing this:

```
ros1ws # type this to configure the terminal for ROS 1 (default)
ros2ws # type this to configure the terminal for ROS 2 (has to be done manually)
```

Each time a package is compiled, the corresponding command (ros1ws / ros2ws) should be run in order to refresh the package list.

1.2 Running the simulation (already installed)

The Baxter simulator behaves as the actual Baxter, meaning that it works on ROS 1. A ROS 1 / ROS 2 bridge is used to forward messages between the two protocols. The simulation should be run from a ROS 1 terminal with:

```
roslaunch baxter_simple_sim simulation.launch
```

It will display RViz with a few in/out topics that can be listed through:

```
rostopic list (ROS 1)
ros2 topic list (ROS 2)
```

1.3 Compiling the package

The folder should be put in your ROS 2 workspace (`~/ros2/src`). Compilation is done by calling colcon from the root of the workspace:

```
ros2ws
cd ~/ros2
colbuild
```

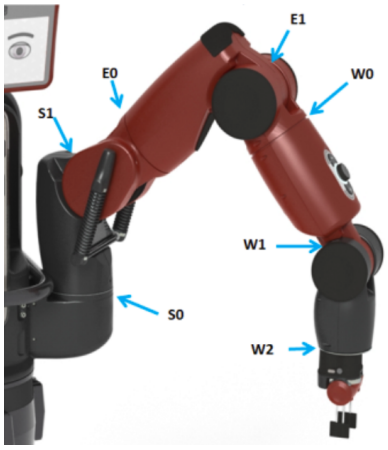
2 Running the control nodes

A basic control GUI can be run with:

```
ros2 launch move_joint slider_launch.py
```

It runs a node that sends the slider value on a topic (here `setpoint`).

The Baxter robot has 2×7 joints as shown in Fig. 1, the names of which are listed in the table.



joints (`left_` & `right_`)	`s0`	`s1`	`e0`	`e1`	`w0`	`w1`	`w2`

Figure 1: Baxter joints

A special node called `move_joint` should be used to change the value published by the slider GUI, to the setpoint of a particular joint:

```
ros2 run move_joint move_joint
```

2.1 Initial state of the control nodes

Use `ros2 node` and `ros2 topic` to get the information on the nodes. There are currently two limitations:

- The `move_joint` node needs a parameter to tell which joint it should be controlling
- The slider publishes on `setpoint` while the `move_joint` node listens to `joint_setpoint`

Additionally, it would be nice to run the two nodes in the same lauch file.

2.2 Regroup all nodes in the same launch file

Open the `slider_launch.py` and add a line equivalent to calling `ros2 run move_joint move_joint`. Then, add a parameter to tell which joint should be controlled:
`parameters = {'joint_name': 'right_e0'}`

To run this launch file, you can go to its folder and type `ros2 launch ./slider_launch.py`. Display the graph (`rqt_graph`) to detect that the slider node and `move_joint` do not communicate, because they do not use the same topics.

2.3 Remapping

In this section we will remap the topic name of `move_joint` such that is uses `setpoint` instead of `joint_setpoint`.

Modify the launch file by adding this argument to the `move_joint` node:

```
remappings = {'joint_setpoint': 'setpoint'}.items()
```

Run the launch file again and you should be able to control the chosen joint.

3 Playing with launch files

3.1 Argument for joint name

Now that a launch file exists to control 1 joint from a slider, we will change the hard-coded joint name to an argument:

```
sl.declare_arg('name', 'right_e0')
```

This syntax tells the launch file that is now has a `name` argument, with default value `'right_e0'`.

Change the hard-coded value to the argument one `sl.arg('right_e0')` and check that the behavior is the same

Also, you can rename the slider GUI by giving an additional argument:

```
sl.node('slider_publisher', 'slider_publisher', name=sl.arg('name'), ...)
```

Then, run the launch file with another name, for instance:
`ros2 launch ./slider_launch.py name:=left_e0`
On another terminal, try to run the same launch file for another joint. What happens?

3.2 Including launch files in other launch files

In this last section we will write a new launch file that will include the previous one, for various joint names. In order to avoid node / topic duplicates, each nodes relative to a specific joint will be in their own namespace.
The syntax to include a launch file in another one is as follow:

```
sl.include('move_joint', 'slider_launch.py', launch_arguments = [('name', 'right_e0')])
```

It should be called from a namespace block, with this syntax:

```
with sl.group(ns = 'right_e0'):
  sl.include('move_joint', 'slider_launch.py', launch_arguments = [('name', 'right_e0')])
```

Check that the behavior is similar to the previous one. Then, write a for loop to run this code for many joint names:

```
joints = ('right_e0', 'right_e1', ...)
for joint in joints:
    with sl.group(ns = joint):
        sl.include(...)
```