Robot Operating System

Lab 2: the "mirror arm" node

1 Goals

Program a node that can move the left arm of the Baxter robot in a symmetric way with respect to the motion of the right arm (symmetry with respect to the sagittal plane of the robot). The node should work in position mode.

1.1 Bring up Baxter

Even if you have a real Baxter robot it can be a good idea to test the lab in simulation first. In both cases, we want to have a RViz display, which is mandatory in simulation and quite handy on the real robot. RViz is run automatically in both cases.

1.1.1 On the real robot

Baxter is a ROS1-based robot. To work with ROS 2 we thus have to run a bridge that transforms all or some topics between ROS 1 and ROS 2.
A launch file is available in the `baxter_bridge` package to run both the bridge and RViz.
You have to connect to Baxter's ROSMASTER in the terminal where you run the bridge:

```
ros2ws && ros_baxter # so that your ROSMASTER is Baxter
ros2 launch baxter_bridge baxter_bridge_launch.py
```

1.1.2 In simulation (including virtual machine users)

The Baxter simulator behaves as the actual Baxter from the ROS 2 side, only with a very limited part of the same topics and services.
The `baxter_bridge` node should be run from a ROS 2 terminal:

```
ros2ws
ros2 launch baxter_simple_sim sim_launch.py lab:=mirror
```

The last argument makes the right arm move with a pre-computer motion, as we cannot move it manually.

1.2 Initial state

The right arm of the robot can be manually moved by grabbing the wrist, or has a pre-computed motion (in simulation). It is waiting for any command for the left arm.

On the real robot of course, only one group can run their code at a time. This is ensured by the `baxter_bridge` that only republished command topics if no other computer has just published on the same topic. Thus, it may be interesting to test your code in the simulation first.
A few in/out topics exist and can be listed through:

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

2 ROS concepts

To create the control loop you must get the current *state* of the right arm and send *commands* to the left arm. You thus need to subcribe and publish to some topics. To not hesitate to use RViz to compare the frames, some of the joints need to get the **negative** value of the other arm.

3 Tasks

- Identify the topics that the node should subscribe and publish to: names, message type
- Check the online documentation "ROS 2 C++ topics" to get the overall syntax. This work requires at least a publisher and a subscriber. A timer can also be used to control at which rate you control the arm.
- Program the node in C++ (and then in Python3 if you feel like it)

The package is already created for this lab (`lab2_mirror`), you just have to update the C++ file and compile it.
Feel free to keep this package as a template for future packages / nodes that you will create.