Robot Operating System

Lab 3: the “puppet arm” node

1 Goals

In this lab, again the left arm will be controlled depending on the position of the right arm. This time, a constant 3D transform will be imposed between the left and right grippers.

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:=puppet
```

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

2.1 Publishing to a topic

In order to control the left arm, you need to publish a command message on the suitable topic (as in the lab 2)

2.2 Using TF

In ROS, the `/tf` topic conveys many 3D transforms between frames, forming a tree. A TF listener can be instantiated in a node in order to retrieve any transform between two frames.

For `/tf` to get all the transforms for Baxter, a `robot_state_publisher` must be run. It is a standard node that:

- loads the description of a robot (URDF file)
- subscribes to the joint states topic
- publishes all induced 3D transforms with the direct geometric model

In this lab, Baxter's `robot_state_publisher` is embedded either in the simulator or in the bridge and automatically subscribes to the joint states.

2.3 Adding a new frame

The desired pose of the left gripper will be defined as a new frame, relative to the right gripper and called `'left_gripper_desired'`.
The node `static_transform_publisher` from the `tf2_ros` package is designed to publish this kind of arbitrary fixed transform between frames:

```
ros2 run tf2_ros static_transform_publisher x y z yaw pitch roll frame_id child_frame_id
```

where:

- `frame_id` is the reference frame (here `right_gripper`)
- `child_frame_id` is the target frame (here `left_gripper_desired`)
- `x y z yaw pitch roll` is the 3D transform (here `0 0 0.1 0 3.14 0`)

2.4 Putting all together

A single launch file should be written in order to regroup the previous commands:

- run the node `static_transform_publisher` with the correct arguments
- include the launch file `robot_state_publisher_launch.py` from the `baxter_description` package

With the simulation and your launch file running, check that you can indeed retrieve the current 3D transform between the frames `base` and `left_gripper_desired`:

```
ros2 run tf2_ros tf2_echo base left_gripper_desired
```

3 Tasks

- Identify the topics that the node should publish to: names, message type
- Identify the services that the node need in order to convert 3D transform to joint positions: names, service type
- Check the online documentation "ROS 2 C++ services" to get the overall syntax. This work requires at least a publisher, a timer, tf, and service client
- Program the node in C++ (and then in Python3 if you feel like it)

The package is already created for this lab (`lab3_puppet`), 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.