ROS 2 exam

2h, documents / internet allowed (closed chats please)

1 Description

In this exam you will have to write a C++ node and run it two times through a launch file. As usual with ROS exams at Centrale Nantes, some StarWars robots are featured and should follow each other in this order:

$$\text{d0} \rightarrow \text{bb8} \rightarrow \text{r2d2}$$

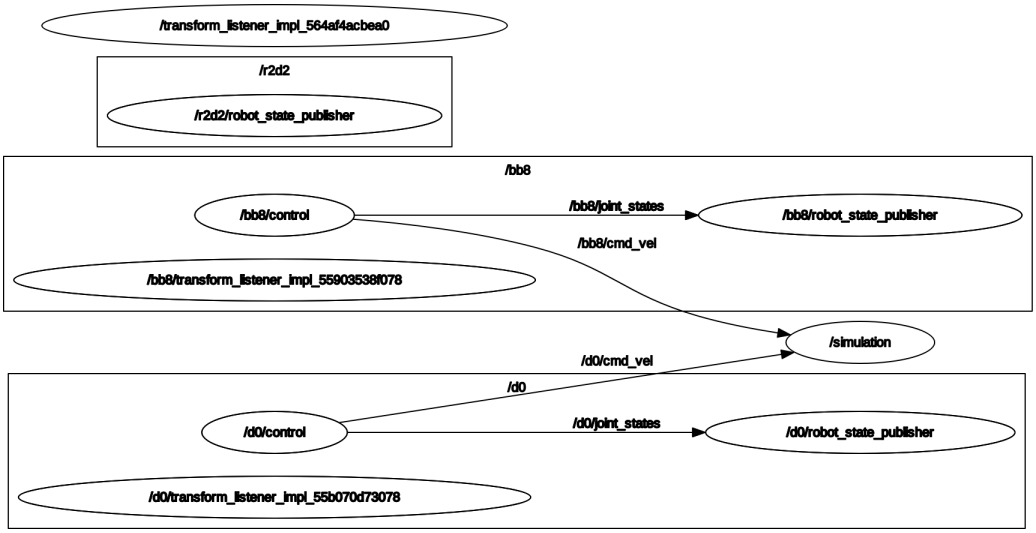
The package should first be compiled, then the simulation can be run once and for all with:

```
ros2 launch ros2_2020 simulation_launch.py
```

R2D2 is already moving independently. The simulation publishes the pose of all robots through TF, using the following frames:

- `r2d2/base_link`
- `bb8/base_link`
- `d0/base_link`

For the two controlled robots (BB8 and D0), the simulation also subscribes to the command velocity on `bb8/cmd_vel` and `d0/cmd_vel`, as a `geometry_msgs/Twist` message.
The final graph should look like this:



2 Writing the node

The node is already more or less setup as `control_node.cpp`. For now it does exactly nothing. BB8 and D0 should get moving, but also have joints that depend on their current velocity. The sampling time can be set to $dt = 0.05$ s.

2.1 Motion control

You have to use the (already setup) TF listener to get the position of the target with regards to the robot frame. The code from Lab 3 (puppet arm) can be of good use.

If the transform is available, we denote (x, y) the linear position error. The control law is then:

$$\begin{cases} v &= k_v(x-1) \\ \omega &= k_\omega \text{arctan2}(y, x) \end{cases}$$

The gains can be set to around 2 for a good behavior.

As usual with 2-0 robots, this command velocity has to be written as a `geometry_msgs/Twist` message, with the fields `linear.x` for v and `angular.z` for ω.

2.2 Joint control

Publishing this command should make the frames move in the simulation, but the 3D models need additional information to be displayed.
Indeed, the controlled robots have 3 joints, the position of which depend on the current velocity as follow:

Joint name	Position
`wheel`	$+3.7.v.dt$ (incremental)
`torso`	$v.\pi/12$
`neck`	$\omega.\pi/8$

Such joint position should be published on the `joint_states` topic of the robot namespace. You can initialize the joint states message in the node constructor, with the names.

Then, when the robot moves, update the `position` field and the `header.stamp` from the current time (`now()`).

You can start by publishing only 0 values, that should let the 3D models display correctly.

3 The launch file

The launch file should run the node twice, so that BB8 follows R2D2 and D0 follows BB8. Each node should be run in the corresponding robot namespace. The robot name and target should be passed as node parameters, that will be used to know which frames to use in the control.

4 Tips

Compile the code once using `colbuild --packages-select ros2_2020`. Then, use `gqt` in the package folder to configure QtCreator. Do not forget to use it with ROS 2 setup.

In order to efficiently debug your code, it is strongly advised to write the node in a specific way to have BB8 follow R2D2. This node should do all necessary things with hard-coded parameters and topics.
You can then make it generic and run it through the launch file.

Feel free to have a look at the `basic_node.cpp` from the lab templates.

Declaring node parameters is detailed in online tutorials and in my slides.

You can use the `simple_launch` package to write the launch files.