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, a few mobile robots are placed in a map and should follow each other.

Your code is to be uploaded on this dropbox on the ROS section.

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

```
ros2 launch ecn_exam_2021 simulation_launch.py
```

If you need to reset the simulation at some point because the robots have gone who knows where, you can let the simulation run and just launch (in another terminal):

```
ros2 launch ecn_exam_2021 reset_launch.py
```

In this simulation, three turtlebot robots are spawned with a laser scanner. The goal is to control robots 2 and 3 (namespaced as `/turtlebot2_test` and `/turtlebot3`) so that they follow, at a given distance, the nearest object they detect with their laser scanner.

The nav stack is run for turtlebot1, you can thus have it move by clicking the "2D Goal Pose" button. Ideally, turtlebot1 should approach the others so that they then move together.

2 Writing the node

The node is already more or less setup as `control_node.cpp`. For now it does exactly nothing. The constructor should:

1. Declare a `distance` parameter, that should be set (later through a launch file) to 0.5 (for turtlebot2) and 0.8 (for turtlebot3)
2. Declare a `robot_name` parameter that should be either `turtlebot2` or `turtlebot3`
3. Declare a publisher on the topic `cmd_vel`
4. Declare a subscriber on the topic `scan`
5. Declare a timer that calls the `move` method every 100 ms.

When the simulation is running, use command line tools to identify the types of messages that should go on each topic.

2.1 Motion control

The `move` method is already declared. This method should:

1. find the closest point in the latest laser scan message through the `findClosest` function. This function returns a `Target` object that is described in the next section (already done)
2. have `target_pub` publish this target. This allows checking in RViz if you have identified the correct closest point (already done)
3. build a command message from the target control law:

$$\begin{cases} v_x &= \texttt{target.computeVx(distance)} \\ \omega_z &= \texttt{target.computeOmegaZ()} \end{cases}$$

4. publish this command message on the `cmd_vel` topic

2.2 Finding the closest point

The laser scan message is structured as follows:

```
float32 angle_min # start angle of the scan [rad]
float32 angle_max # end angle of the scan [rad]
float32 angle_increment # angular distance between measurements [rad]
float32 range_min # minimum range value [m]
float32 range_max # maximum range value [m]
float32[] ranges # range data [m]
```

Assuming the scan message is called `scan`, ranges are thus stored in `scan.ranges` as a `std::vector`.
The range at index i corresponds to the angle `scan.angle_min + i*scan.angle_increment`.

In order to find the closest detected point, you have to loop through the indices and keep the one corresponding to the smallest range, ignoring ranges that are not between `range_min` and `range_max`.
Just update `target.range` and `target.angle` inside this loop and return `target`.

3 The launch file

Once the node runs correctly for turtlebot2, make it generic and modify `both_launch.py` so that it runs the node twice:

- The node controlling turtlebot2 should be run inside the `/turtlebot2` namespace without any remappings, and use a distance of 0.5 m
- The node controlling turtlebot3 should be run inside the global namespace with remappings, and use a distance of 0.8 m

Do not forget to also set the `robot_name` parameter when running the nodes.

4 Tips

Compile the code once using `colbuild --packages-select ecn_exam_2021`. 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 only for turtlebot2 first. 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.