ROS 2 exam

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

1 Description

In this exam you will have to write a C++ or Python node and run it several times through a launch file.

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

```
ros2 launch ecn_2024 simulation_launch.py
```

This simulation involves several robots (5 in the initial simulation). You can add more by modifying `simulation_launch.py`. Each robot is controlled through the `cmd_vel` topic of its own namespace.

Another node, namely `letter2path`, is run with the simulation and advertizes the service `/plan`. This service is of type `ecn_2024/srv/Letter` and returns the contour corresponding to a given letter.
The resulting package should then be zipped and uploaded to this folder.

2 Control node

The control node should be run for each robot and should:

- Call the `/plan` service to get the contour to draw a given letter
- Move the robot from a timer callback, in order to track the path
- Optionally, publish the path that is tracked by the robot

The control node has a `tracker` member variable that will take care of the actual control law. It needs to be given:

- the frame that should be controlled, typically obtained through a parameter.
  The frame has the shape `robot_name/base_link`
- the path that is to be tracked, obtained by calling the service

2.1 Calling the service

The service definition is given by `ros2 interface show ecn_2024/srv/Letter`:

Request	Response

`string letter`
`float64 x`
`float64 y`
	`geometry_msgs/Point[] contour`

The request should be filled with node parameters `x, y` and `letter` so that they can be modified when launching the node.

2.1.1 C++

The method to call a service, assuming `client` is the name of the client, is:

```
client->wait_for_service();
client->async_send_request(request, <callback function>);
```

The callback function is already created as `client_cb`. This callback should:

- initialize the `tracker` from the response (`.initFrom` function)
- (optional) convert the contour into an actual `nav_msgs/Path` in order to visualize it in RViz

In such a callback, the argument is a `SharedFuture` and as such:

- `result = future.get()` is a pointer to the actual result
- the contour is obtained through `result->contour`

2.1.2 Python

The way to call the service, asuming `client` is the name of the client, is:

```
client.wait_for_service();
future = client.call_async(req)
future.add_done_callback(<callback function>)
```

In the callback function you should consider the response and use `self.tracker.initFrom`

2.2 Moving the robot

The robot should move with a timer running at 10 Hz. The `tracker` member variable has a `track()` method that will return two values (v, ω) that let the robot track its contour.

This values should be converted to the appropriate message and published on the `cmd_vel` topic under the corresponding namespace.

If the tracker was not correctly initialized (wrong frame / no contour to track) then the velocity will be null but the logic still holds.

(optional) If the actual path was saved in the callback, you just need to update its timestamp so that RViz continues to display it.

3 The launch file

As always, write the main node for one robot only, then make it generic. A launch file named `fleet_launch.py` should then run your node for several robots, inside each robot's namespace.

You can pick the letters you want and in order to draw a word, each letter should have an offset of +10 along `x` compared to the previous one. You can also add a vertical offset if you want.