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 several times through a launch file.

The files are split into two packages that you should compile first:

- `ecn_usv`: simulates an unmanned surface vehicle. You do not have to modify anything in this package
- `ecn_2023`: simulates a world and some drones, this is where you do the work.

The overall simulation can be run with two commands (to be run in two terminals):

```
ros2 launch ecn_2023 world_launch.py
ros2 launch ecn_usv usv_launch.py
```

once run there should be no reason to stop it. The drones will go back to their home position if no control is received.

Objectives

The goal of the exam is to have 4 drones track a suspicious USV moving around an island. To do so, each drone should have a desired position (defined with regards to the USV) and a controller to track this setpoint. The steps are thus:

- Write the controller node
- Write a launch file that:
  - Runs the controller for all drones
  - Runs static transform broadcaster to define where the drones should be with regards to the USV

The nodes should be run in the namespaces `uav1` ... `uav4`.

2 Static transform broadcaster

The target position of a given UAV is defined with regards to the USV base frame. Such a transformation should be published on `/tf` through a Static transform broadcaster:

```
ros2 run tf2_ros static_transform_publisher
```

This node takes the following arguments, assuming we set the desired pose for UAV 1:

- `--frame-id`: name of the parent frame (`usv/base_link`)
- `--child-frame-id`: name of the child frame (`uav1/target`)
- `--x`: X-offset of the pose
- `--y`: Y-offset of the pose
- `--z`: Z-offset of the pose (15)

The X and Y offsets should be set depending on the UAV number:

Drone	X-offset	Y-offset
1	10	0
2	0	10
3	-10	0
4	0	-10

You can run first this static transform publisher in a console for drone 1, to help coding the control node.

3 Control node

The control node is written in `uav.cpp`. Parts of the node class are already here. As usual, it is a good idea to code the control explicitely from a given UAV before making the code generic and use it from a launch file.

3.1 Inputs / outputs

The node should:

- Declare a parameter to know which UAV is considered (1 to 4)
- Declare a publisher to send the controls (use `ros2 topic` to identify it)
- Use a service client to retrieve the current error to its target position
- Run a controller at a rate of 50 ms

Some gains are already declared.

3.2 Service client

A service called `/target` is advertized by the `/target_computer` node. The service is of type `ecn_2023/srv/Target`:

```
string uav
---
float64 x
float64 y
float64 z
float64 theta
```

The request should contain the name of the UAV (`uav1` ... `uav4`). The response will be the current error (x, y, z, θ) from the UAV to its target pose.
Calling this service in a synchronous way can be done through the `client` member variable:

- Needs to be initialized in the constructor, to be given the name of the service
- Can be called with various approaches (see `client_spinner.h`):

```
// returns an actual ResponseT if the call succeded
std::optional<ResponseT> call(const RequestT &req)

// returns True is the call succeded, in this case res is the response
bool call(const RequestT &req, ResponseT &res)
```

3.3 Control

The control is a basic proportional-integral that takes the error from the service call. This function should be run at 50 ms.

Function `updateControl()`
Call service to get current error, named `error` here
if *Service call failed* **then**
  return;
end
`// Integral error for x and y`
$ei_x \leftarrow ei_x + \texttt{error.x}$
$ei_y \leftarrow ei_y + \texttt{error.y}$
`// Compute desired velocity`
$v_x \leftarrow K_p * \texttt{error.x} + K_i * ei_x$
$v_y \leftarrow K_p * \texttt{error.y} + K_i * ei_y$
$v_z \leftarrow K_p * \texttt{error.z}$
$\omega_z \leftarrow K_w * \texttt{error.theta}$
Publish velocity command $(v_x, v_y, v_z, \omega_z)$

Algorithm 1: UAV Control

4 The launch file

Once your node runs for a given drone, write a launch files that runs the node and the static transform publisher for the 4 drones.

Tip You can use a loop for the drone number and get the corresponding namespace:

```
for i in (1,2,3,4):
    ns = f'uav{i}'
    # do stuff in this namespace
    # use i to run static_transform_publisher with correct arguments
```

5 Bonus: super launch launch file

Once everything works, you may also write a super-launch file that runs everything in one command:

- include `world_launch.py`
- include `usv_launch.py`
- include your own launch file for the UAV's

In order to reduce oscillations for the USV, the `usv_launch.py` file can be passed better gains when included:

- `Kv` argument can be set to 4.
- `Kw` argument can be set to 1.2