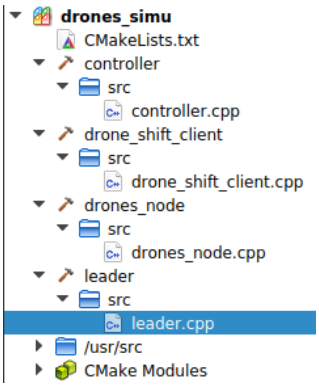
SOFAR lab exam 2021

The topic of the exam is a very crude simulation of a formation of drones. The exam has two parts : part I uses publish/subscribe and services. Part II uses the tf2 package.

Scrupulously follow the preparation instructions.

Preparation for part I

- Create a new workspace: `~/lab_exam` and create the subfolder `~/lab_exam/src`
- Copy the file `drones_v1_students.zip` to `~/lab_exam/src` and decompress it. You should obtain the following hierarchy: `~/lab_exam/src/drones_gr1_v1_students` with three subfolders named `drones_msg`, `drones_serv` and `drones_simu`. Each is a package. Packages `drones_msgs` and `drones_serv` only define messages and contain no nodes. Folder `drones_simu` contains the provided nodes (which you do not need to look at) and the skeleton of the node you need to program in part I: `controller.cpp`. Ignore the other nodes for the moment: they compile but do nothing.
- Open a terminal. Go to `~/lab_exam`. Compile (`catkin build`), then execute « `source devel/setup.bash` ». Remember you will need to execute this command again if you recompile from scratch after suppressing subfolders `build` and `devel` from `lab_exam`.
- Open another tab. Go to `~/lab_exam/src/drones_gr1_v1_students/drones_simu`. Execute the script « `gqt` » and start qtcreater (`qtcreator Cmakelists.txt`).
- Developping the project tree should give you this :



- Launch the nodes to check that everything is OK by typing :
  `roslaunch drones_simu all.launch drones:=2`
  In another tab or terminal type « `rosnode list` ». The following nodes should be listed :

```
/drone0/controller
/drone0/drone
/drone1/controller
/drone1/drone
/leader
```

Fundamental:

- In this part, you do not need and **you must not** modify any file other than « `controller.cpp` ». Actually you do not even need to look at the code of other nodes. It will not help.
- You **must not** modify the launch files. They use techniques we have not studied and can be confusing. You can look at them later if you are interested, but not during the exam.
- The only file you need to launch is « `all.launch` ». In the command
  `roslaunch drones_simu all.launch drones:=2`
  « `drones:=2` » gives the number of drones that fly in formation around the leader. No spaces are allowed around the « `:=` » operator.

Provided files/nodes for part I.

- The « leader » type of node is used for the leader drone. The leader drone follows a fixed (helicoidal) trajectory. This node is aware of the number of drones in the formation and publishes to a topic « `/goals` » the positions at which the drones must be. Check the corresponding message format and make sure you understand what it means.
- The type of node « drones_node » represents a drone of the formation. It publishes the position of the drone and subscribes to a velocity command. This type of node is run in multiple instances by the launch file.
- The type « controller » is node that controls a drone to follow the leader. Each controller node controls a single drone. The controller drones receives the goal positions for all drones, but uses only the information pertinent to the drone it is controlling, ignoring the rest.

Tasks of part I.A.

- Using commands like « `rosnode info` », « `rostopic echo` », « `rosmsg show` », make sure you understand the format and role of each topic, and the logic used to name the topics.
- Assuming one leader and three drones in the formation (in addition to the leader), draw the graph of the application. The graph shall include :
  - All node names.
  - All connections via topics.
  - All topic names as they will be in the running application.
- **Call the teacher for validation of the graph.**
- Program the controller node. The control will be a simple proportional controller with a gain of 10. Each component of the desired velocity vector will be KP*(goal_coord-drone_coord). The controller **needs** the ID of the drone it is going to control. The ID **must be read** as a local node parameter, with default value 1. Make sure you correctly read (print it to check) the drone ID before you go any further (self-validation).

- **Call the teacher for validation of the application.**

Suggestions :

- Keep three terminal tabs with `roscore`, `rviz` and `rqt_graph` running at all times.
- Use an additional tab to compile and run the application.
- Use an additional terminal (not tab) to run commands.
- In rviz, add markers so you can see the leader (red disk) and the drones (blue disks).

Tasks of part I.B.

The positions of the drones are generated randomly at startup. But the leader node provides a service to relocate a drone by sending in the request part of the service message its ID, the desired distance to the leader and the desired polar angle.

Write a program that reads the information at the keyboard and calls the service to relocate the drone (file « `drone_shift_client.cpp` »). Note that this program does not require a ros::Rate object. **Do not add your node to the launch file**. Run it in a separate terminal. **Call the teacher for validation.**

When part I is complete, zip the whole « drones_simu » folder and submit to Hippocampus.

Preparation for part II.

In part II, we are going to solve the same problem as in part I, this time using the package tf2. But for that, **you must remove the packages of part I**. Move the folder `drones_gr1_v1_students` out of your workspace. Normally, you should even be in a position to delete it, as **you must have zipped and submitted your work of part I before proceeding to part II**. Just in case, do not delete it, but put it outside your workspace.

In this part, you are allowed to **add** code to `leader.cpp` and `drones_node.cpp` if you feel it is necessary, but the existing parts of the code **must remain unchanged**.

Setting up: Decompress the file `drones_gr1_v2_students.zip` in folder `~/lab_exam/src`, then compile, source file `setup.bash`, go to folder `~/lab_exam/src/drones_gr1_v2_students`, configure and start qtcreator.

Before programming, list on paper which frames you plan to broadcast (give their names and the frame they will be attached to), and which node is going to broadcast them. Do this work assuming one leader and three additional drones in the formation. **Call the teacher for validation**.

Program the application using tf2.

Tip: To create frame names, you may need to use string concatenation, which is performed using the overloaded operator « + ». Also, you may need the function std::to_string(int) which transforms and integer into a string.