Sorbonne Université – Département de Sciences De l'Ingénieur
ROS and experimental robotics
March 2022, 1h30.

You are in charge of the development of a mobile robot called `mybot`, endowed with 2 directional wheels. This robot is also equipped with 12 ultrasound sensors (US), see Figure 1. Each sensor of the ultrasound array aims at estimating the distance to the obstacles around the robot through a time-of-flight computation: a short high frequency burst is sent by the US emitter, which comes back to an US receiver when an obstacle is present. Each sensor actually measures the time taken by the wave to go back and forth, which is a function of the speed of sound ($c \approx 340$ m.s^{-1}) and the distance d to the obstacle. The US sensor documentation indicates:

> *Each US sensor is not able to measure any distance greater than* $5m$*, or smaller than* $5cm$*.*

In practice, the ROS interface to the sensors array relies on a node `mybot_usarray` which communicates directly with the array of US sensors, and publishes (among other things) an array containing the 12 time-of-flight values (in s) for each US sensor on the topic `/usarray`. In this topic, the data are arranged in the sensor IDs order, see again Figure 1.

Unfortunately, it appears that one or multiple US sensor(s) in the whole array might sometimes be broken. The symptoms may vary: sometimes the faulty sensor(s) send(s) the same value everytime regardless of the actual obstacle distance, sometimes the sensor exposes a negative time-of-flight or very small or large, unrealistic, value. Then, the `mybot_usarray` node exposes a service `/usarray_mngt` which allows to deactivate the faulty sensor(s) in the array. The documentation of the `mybot_usarray` node for the service `usarray_mngt` explains :

> *The node exposes a service named* `usarray_mngt`*. The message type used in the service is declared in the* `srv` *folder of its ROS package. As in input, you have to provide the ID of the sensor you want to activate or deactivate. You also have to provide a text containing either "ACTIVATE", "DEACTIVATE" or "STATUS" to use the service. When using "STATUS", the provided ID is not used by the service. As a response, you will get a text validating your request.*

Your objective is to verify if you have a faulty unit in your robot US sensors array, localize it, deactivate it, and then to write an emergency stop ROS node that should send a speed of 0 to the `/cmd_vel` topic used by the `mybot_control` node to drive the robot. Your node must also expose a service `is_running` which should specify if you are in a state where an emergency stop has been ordered.

Evaluation

At the end of the exam, you have to upload two files to Moodle:

- A PDF report, written with `openoffice` for example (in french or english), where you answer to the questions below and where you can copy and past the outputs of the different commands you use to answer the questions. For example, for the question "3. List all the running nodes", you must put in your report :
  - the exact command line you use to answer the question,
  - and the corresponding output you get.

  Obviously, you have to comment all your responses. A simple copy and past without a sentence explaining the outputs is **not** considered as a full response to the questions.
- A `zip` file of your node that will be used to access your code, but also to actually launch your own package and nodes.

Preparation

1. To begin with, download the file `mybot.zip` from Moodle. Extract it in the `src` folder of your catkin workspace. Launch `catkin_make` to build your ROS workspace and source you `.bashrc`.

2. Go to the `src` folder of the package `mybot` you just downloaded, and make executable all the python scripts in there.
3. Quickly test that all is OK by running the node `mybot_usarray`. Contact your teacher if it is not the case.

1 Exploration of ROS and of the nodes

4. Run the nodes `mybot_usarray` and `mybot_control` from the package `mybot`. Then list all the running nodes.
5. List the information about the `mybot_usarray` node. Explain each line of the output you get.
6. List all the available topics. Explain the role of the `/rosout` topic.
7. What kind of message is available on the `/usarray` topic? Detail each field of the message.
8. Display a few messages available on the topic `/usarray`. At which rate are they published?

2 Emergency stop node

You have to write a new ROS node `emergency_stop` which should trigger an emergency stop when the robot detects an obstacle **in front or behind it**. This emergency stop must be activated by publishing a null speed to the `/cmd_vel` topic of the `mybot_control` node as soon as the distance d to the obstacle detected **by any sensor in the front or back of the robot** is smaller than a threshold (expressed in meters) than can be tuned manually. If no obstacle is detected, then the robot is expected to go straight at a speed of your choice.

9. Create a package `exam` with dependencies to `rospy`, `std_msgs` and `mybot` ;
10. Create the node `emergency_stop` which should subscribe and publish to the adequate topics by using `rospy`:
    (a) Define a variable `threshold` in your code that can be used to tune the obstacle detection. By default, an obstacle closer to $0.5m$ must trigger an emergency stop ;
    (b) Print a message `"GO"` or `"STOP"` in the terminal and publish a null speed with the **first detection only** of presence of an obstacle.
    (c) Create a launch file where the three nodes `mybot_control`, `mybot_usarray` and `emergency_stop` are launched together, and where the distance threshold is set ;
    (d) Create a ROS parameter `threshold` in your launch file that you can use to define the above `threshold` variable in your code.

Evaluation

It is important you comment extensively your code, even for elementary lines of code. Your node will be assessed by running your node (it must obviously run as expected and provide the functionalities listed above), but the quality of the code and its comments will also be taken into account.

11. Using the service `usarray_mngt`:
    (a) Inspect the output of your node. Knowing that your robot is inside an empty rectangular room, does it work as expected?
    (b) Inspect carefully the message sent by the US sensors. What can you see? Propose a solution using the `usarray_mngt` service.
12. Publication of the distance information:

(a) Modify your node so that it now publishes to a new topic `/distance` the four mean[1] **distances** (corresponding to the four directions) to the obstacles around the robot.

(b) On the basis on the output of the `/distance` topic, place the walls around the robot in Figure 1 (do not forget to specify the scale on your plot!)

13. (Bonus) Creation of the service `is_running`:

(a) Create a service named `is_running`, expecting a message of type `std_srvs/Trigger` and returning a string `"GO"` or `"STOP"` depending on the state of the emergency stop.

(b) Test you service with the command `rosservice`.

[1] To that end, you might be interested in the function `nanmean()` from `numpy`.

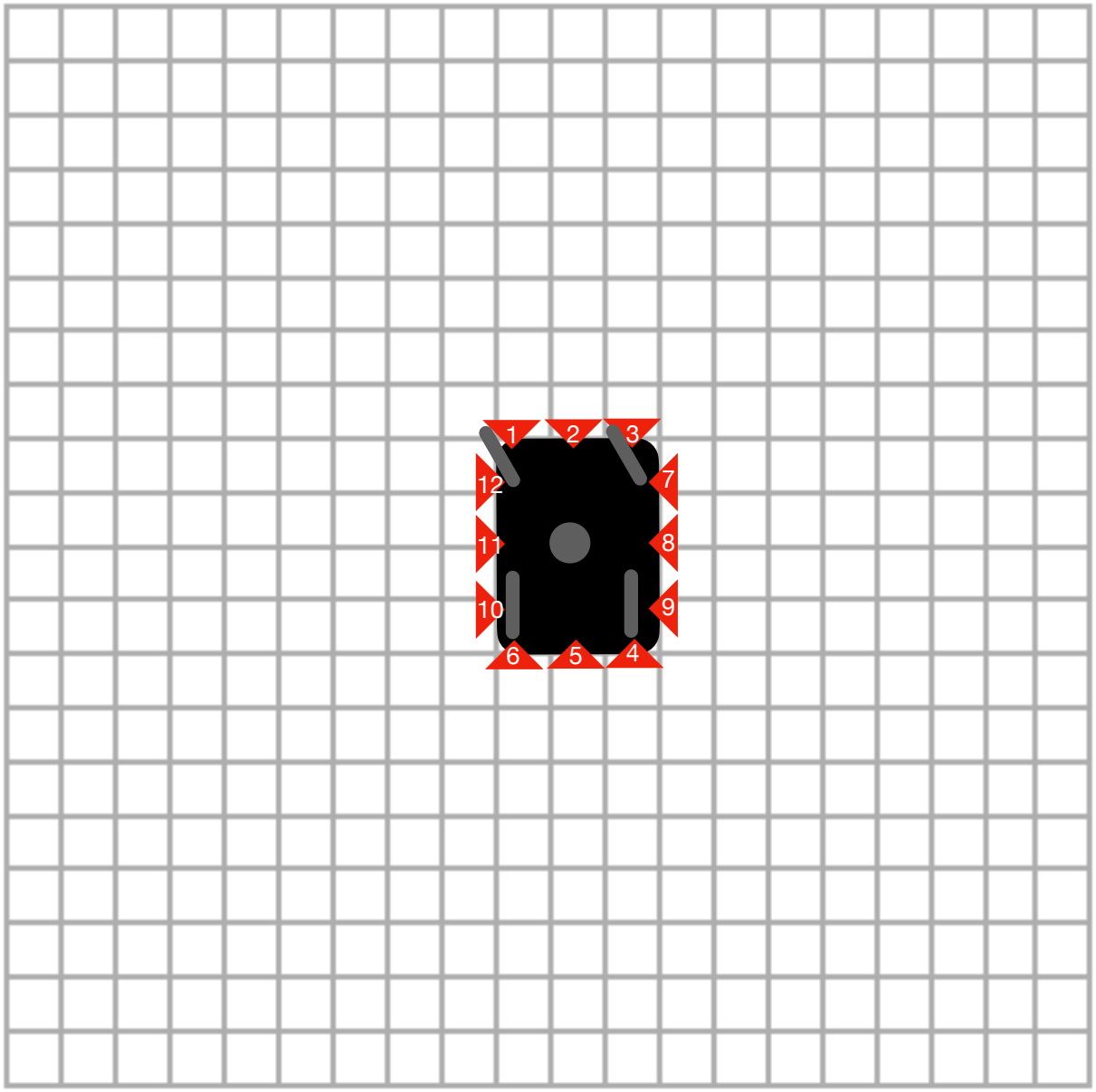

Figure 1: View of the `mybot` robot and of its US sensors (in red) spatial organization. Each sensor is identified with an `ID` number (in white) inside the sensor array.