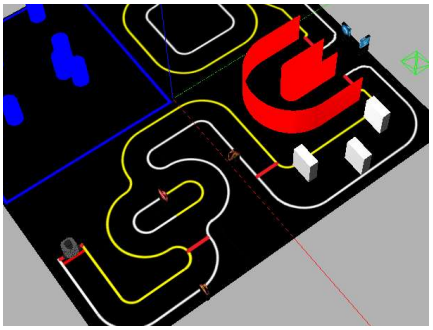
My ROS project

CARREE
Antoine
3803826

OUMBICHE
Loris
21117913

Introduction :

This project's goal is to learn and code in ROS.
To do that we coded a robot that's able to do line following tasks as well as obstacle avoidance such as bottle dodging or crossing a corridor.



The use of LIDAR and camera will be necessary to make the robot interact with his surroundings. Thus, our code is all about, getting the data from captors, processing it and finally feeding the results to the teleoperation formula.

Presentation of the ROS architecture

For practical means , we decided to minimize the number of nodes used, but also the number of topics to which we publish, so most of our nodes that process the data from captors will send an array with "flag", "mean"...

For the architecture we have chosen to make three nodes, Camera node, LIDAR node, Teleop node.

A) node overview / Architecture:

1. Camera node :

This node is used to process the raw image that comes from the /camera topic: The node isolates the colors we ask for (red,green,blue,yellow) and publishes the mean value of all pixels of these colors. The results are then published through the /cam_data topic.

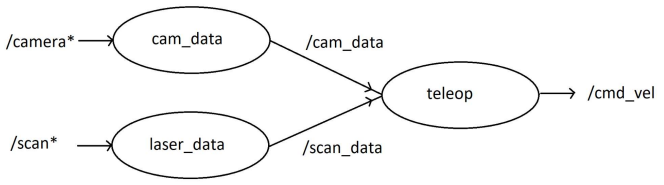
2. LIDAR node :

This node gets the data from the /scan topic that contains a matrix with all the distances of near objects. Each indice of the matrix corresponds to the angle between the robot and the object and we can thus get the object's distances around the robot to define a right and left orientation for the detection (right 20° to 80° and left 280° to 340). Results are then published to the /scan_data topic.

3. Teleop node:

The teleop node has for sole purpose to read the data processed by the camera and lidar nodes and feed it to the right teleoperation formula that will then control the robot. Naturally, the node is thus subscribed to /cam_data and /scan_data.
These data are then sent to the /cmd_vel topic which contains the data of the linear/angular speed of our robot.

connection between node



* we only show the new parts that we have created in the project

B) *Algorithme structure :*

For clarity measures, this part will be divided into three main parts : line following, selection mode and obstacle avoidance

1. *Line following :*

The line following process will use the camera node and the teleop node, all the process related to the image will be processed in the camera node and results will be sent to the teleop node.

START-

At the start when the robot sees a line ,we will use a color mask to extract a specific color (in our case red and green), and then define a mean between each color to get the center of detected lines or objects. This method's major problem is that when the camera detects, for instance, both a red line and object, the mean will be between all the red pixels detected without discriminations and thus lead to a wrongly assumed center. To minimize this problem we choose to only process the lower part of the camera, so that the robot will have less chances to see unwanted objects.

After this first processing, we get two means : a "red mean" and a "green mean". We can see those means as imaginary borders that the robot shouldn't cross as we want the robot to always stay between those means. For this purpose, we will define, on the pixel map of the camera, the position of the robot (center X and lowest in Y). This will allow us to check if the robot is well positioned between our two lines. Note that at this point, /cam_data contains the position of the robot and the position to be reached.

The teleop node defines the error between the wanted position and the actual position and corrects the direction, sending information to the /cmd_vel topic.

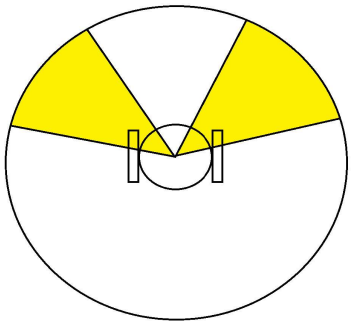
2. *Mode swap :*

The swap mode is calculated with the LIDAR: we define a circle around the robot and detect the number of "object" close to him. If this number appears to be too high, the robot will swap between line following, LIDAR or will use both. Note that we need two thresholds on the number of detected objects because when we encounter thin objects like bottles, the number of points detected is low compared with that of the corridor.

When the corridor is near, we will detect a great number of points and the command will swap into LIDAR only mode.

3. *Obstacle avoidance :*

For explanatory purposes, we will use the corridor exemple. When entering a corridor and swapping the mode into LIDAR the LIDAR node will send to the teleop the mean of the distance detected between right (20° to 80°) and left (280° to 340°) .

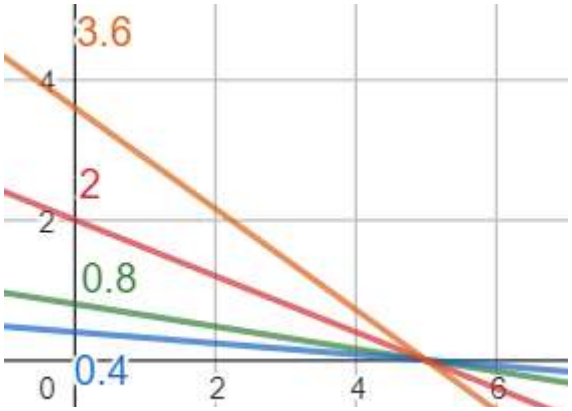


The goal for the robot is to be at the same distance between the two objects/walls. To do so, we calculate the mean of these distances and fetch the difference between them to the /scan_data topic (the sign of the difference gives the spin direction). Finally, the teleop formula will minimize the difference in distance between the two objects.

formula:

$$f(x) = 3.6 \cdot \frac{x - moy}{moy}$$

moy correspond to the reference point where we should be on the camera (the center of the image with a small shift in our case).

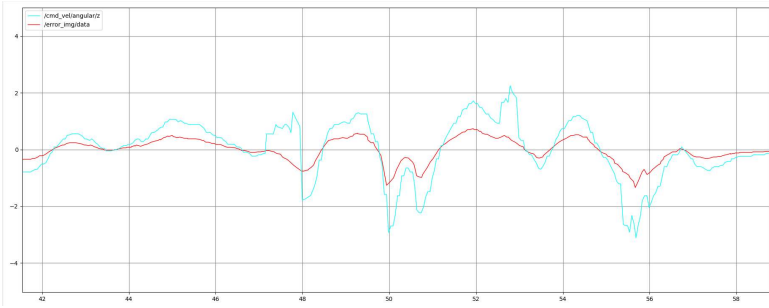


If we modify the coefficient in front of the quotient in the previous expression, the reactivity of our system will change. Indeed, if the coefficient is low the reactivity decreases proportionally, the main difficulty with this kind of formula is that optimal coefficients are found through multiple tests, for both the simulation and the real run.

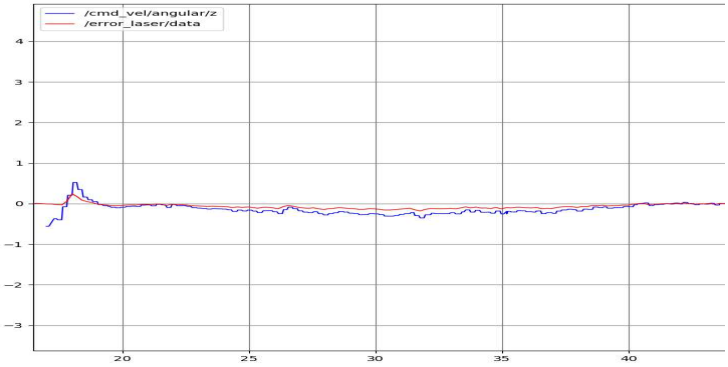
III PERFORMANCE CARACTERISATION :

In this final part, we will deepen the functions we used and what we can expect for both the first function (line following) and the second function(LIDAR detection)

The first figure represents the response to the error of the line following algorithm and its angular reaction (all values are scaled to be readable). The error is in red. As we can see, the angular reaction follows the error (same variation), what's important to observe here is the reaction time and the potential error we can have.
We can't clearly see but the variation are following each other with a time < 0.5 seconds and so the error caused by the processing delay will be low. An advantage of that is that we will be able to increase the linear speed of the robot without getting any error. Another interesting point to see is a that it seems to be an odd variation that's inverted in comparison to the error variation, this point is in reality the entry of the roundabout, the algorithm process red and green color ahead of him and not on his side leading to a direction probleme that can be solve by shifting a bit the mean toward the desired turn direction.



Same as the figure before but in red the error and in blue the angular reaction



Conclusion and perspective :

At the end of this project we found ourselves proficient in creating nodes and making node architecture to control a turtlebot and well aware of the difficulties between the simulation and the real application.
For perspectives we could have made an adaptive speed to make the robot progress smoother or a nonlinear model on the angular process to make a more progressive reaction, our model has the risk to react abruptly to low error, if we used a non linear model the robot would not overreact to really small error that don't need to be corrected. Those errors could in fact come from bad detection of an object, the main idea would be to let the robot "evaluate" the environment without doing anything until he really needs to so he can confirm whether or not he should take action.