Control a simulated and a real mobile robot

Lina Insaf BEKDOUCHE
Master ISI
Student number 28620402

Lisa TOUZARI
Master ISI
Student number 28617271

Xinyi CUI
Master ISI
Student number 3873971

I. INTRODUCTION

In this project, we will control a simulated and a real Turtlebot 3 burger mobile robot in a realistic challenging environment, so as to make it navigate from one starting position to a goal position with different tasks to solve on the path. The navigation should successively exploit:

- images obtained from a simulated/real camera to detect and follow a line,
- a laser scan obtained from a simulated/real LDS to detected and avoid some obstacles,
- and finally both of them to navigate in a challenging environment where both sensors are required together.

The project will take the form of 3 successive challenges relying on the sensorimotor capabilities of the turtlebot, as illustrated in Figure 1.

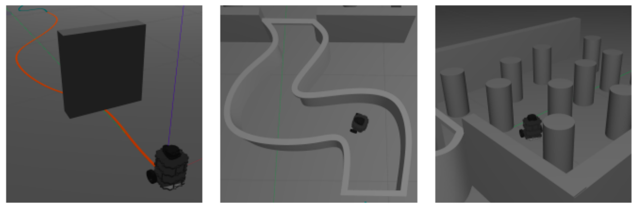
(a) Challenge 1: line following (b) Challenge 2: corridor (c) Challenge 3: cluttered environment

Fig. 1. Our challlenges.

II. PRESENTATION OF THE ROS ARCHITECTURE

Our architecture is organized as follow:

- Under our workspace (in src), we created our package challenge_2021. A package refers to a typical structure of files and folders.
- Our package contains main folders : - Scripts: contains our nodes, a node really is an executable file. ROS nodes use a ROS client library to communicate with other nodes. Nodes can publish or subscribe to a Topic. Nodes can also provide or use a Service. - Launch: contains launching files that allow us to run our nodes.

A. A short overview

Challenge 1 :

- First, we are going to create a new node named $challenge1$.
- In order for our robot to follow a colored line and avoid obstacles at the same time, we need the camera and lidar data, which we need to subscribe to two topics: $/camera/image$"and $/scan$"
- To make our robot move, we need to update the linear and angular velocity of our robot which we publish in the topic : $/cmdvel$.
- The types of messages published among the topics $/camera/image$ and $/scan$ are respectively are: $Image$ and $LaserScan$.
- Apart from this,the type of messages published in the topic $/cmdvel$ is $Twist$.

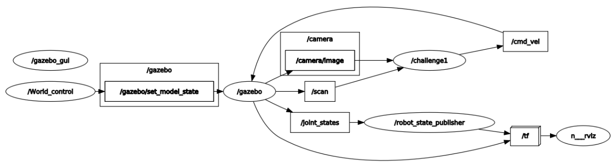

Fig. 2. Architecture for challenge 1.

This graph gives us the architechture of our nodes for challenge 1.

- Like we can see, we have the node /challenge 1 that we created that subscribes to /camera/image and /scan topics to get the images of our camera and the position of our obstacles.
- Our node is also publishing on the topic /cmd_vel informations on how the robot should move.
- We can also see the nodes rviz and gazebo. gazebo gives us simulated version of our mobile robot with all the topics that we neeed. The robot model is published in Rviz.
- In the case of a real robot we have pretty much the same architecture, except for the gazebo node which would be replaced by the robot himself.

Challenge 2 :

- First, we are going to create a new node named $challenge2$.
- In order for our robot to navigate autonomously inside a corridor, we need the lidar data , which we subscribe to the $/scan$ topic and then published the velocity commands in the $/cmdvel$ topic.
- To make our robot stop at the end of the corridor, we need to get its position, thus we subscribe to the $/odom$ topic.

- The types of messages published in the topics $/camera/image$ and $/scan$ are respectively are: $Image$ and $LaserScan$.
- Besides,the types of messages published in the three topics listed previously are respectively: $LaserScan$, $Twist$ and $Odometry$.

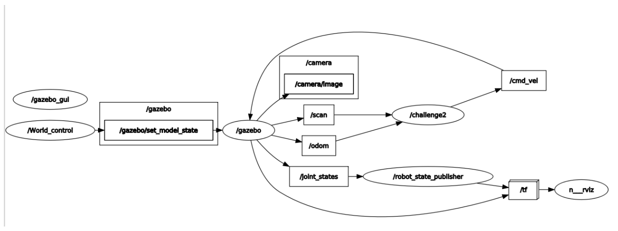

Fig. 3. Architecture for challenge 2.

This graph gives us the architechture of our nodes for challenge 2.

- just like in the challenge 1, we have the node /challenge 2 that we created that subscribes to /camera/image and /scan topics to get the images of our camera and the position of our obstacles.
- Our node is also publishing on the topic /cmd_vel informations on how the robot should move.
- the only change is the that our node /challenge2 is also subscribed to the topic /odom that allow us to read the position of our robot.

Challenge 3 :

- First, we are going to create a new node named $challenge3$.
- In order for our robot to navigate in a more challenging environment where it avoids obstacles and reaches a final goal, we need both the camera and laser data.
- So we subscribe to both topics $/camera/image$ and $/scan$.
- and obviously we need to publish the velocity commands in the topic $/cmdvel$ which makes our robot avoid the diffrent obstacles and get to the finish point.

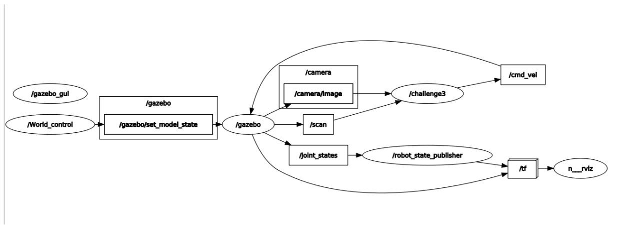

Fig. 4. Architecture for challenge 3.

B. Algorithmic structure of the architecture

Challenge 1: In this first challenge, our robot has to follow a colored line (2 different lines : a yellow line and a blue line). At the same time,it stops in case of detecting a moving obstacle, and avoids a static obstacle.Frankly speaking, there are4 differents tasks for our robot to achieve :

- Follow a yellow line.
- Follow a blue line.
- Stop in front of an obstacle.
- Avoid (contour) an obstacle.

Since we need both the laser and camera data to do the previous tasks, we subscribe to both the $/camera/image$ and $/scan$ topics, and create callback functions (read image callback, read pose callback) so as to get the published information continuoulsy.

- In the core of the read pose callback, we create four different variables :
  - obstacle: to get the mean distance between our robot and an obstacle in front of it, if it is in the ranges between [0, 9] and [350, 359].
  - obstacle right: to get the mean distance between our robot and an obstacle to its right if it is in the ranges [265,274].
  - obstacle back: to get the mean distance between our robot and an obstacle behind it if it is in the ranges [255,264].
  - a boolean variable to detecte obstacle: if obstacle is less or equals to a certain value, which means we have an obstacle ahead, otherwise, the way is free of obstacles.
- In the core of read image callback :
  For each iteration, we get the images published in the $/camera/image$ topic and process them. First, we transform those images to opencv images format and change the color representation from BGR to HSV. Then, we perform a binarisation of the images and proceed with the calculation of the differents masks moments (yellow mask, blus mask, and the total mask).
- We also create two more variables:
  - $tourner$: is a boolean variable, if it is equal to 1, that means we are in front of a static obstacle.
  - $numobstacle$: it detectes wether we are in front of a static obstacle or moving obstacle.
- At this point we have all the data and necessary variables for our robot to perform different tasks of the first challenge:
  - if $tourner$ equal to 1:
    - if we do not detect anything to the right side of the robot (obstacle right), we will turn our robot to the left until we detect the next obstacle.
    - then, we give our robot a set of instructions to go around the obstacle (get away from it) until it is detected in obstacle back, the obstacle is behind us once then we just continue following the line.
  - if $tourner$ equal to 0 :
    - in this part of the programm, we need to know wether we are in the yellow line or the blue line, so if the moments of the yellow_mask $<= 0$, that means we are in the blue line, otherwise, we are in the yellow one. With this information, we can update the num_obstacle variable, if we are in the yellow

line that means the obstacle detected will be the first one, otherwise,it's the second one.
    - We calculate the coordinates of the center of the image, and compute the error between the center and the detected point.
    - if we dont detect any obstacle: detcte_obstacle==False, we will follow the line.
    - if we detecte an obstacle, and the num_obstacle==1, we will stop.
    - if we detecte an obstacle, and the num_obstacle==2, then tourner=1, and we go to the first condition and apply the set of instructions in robot.
  - if the moments of the total_mask<= 0, that means we are at the end of the line, so we stop our robot.

PS: the different velocity commands are published to the topic /cmd_vel

Challenge 2: In this second challenge, our robot has to navigate autonomously inside a corridor, and stop at the exit. We get the laser and odometry data with the callback functions: read_odom_callback and read_pose_callback. We define an other function take_action, to specify the command to publish in the velocity topic /cmd_vel.

- read_odom_callback: in the core of this callback function, we get the y coordinate of the robot in the gazebo world.
- read_pose_callback: in the core of this function ,we create a variable region, where we save the minimum distances in the front, right and left of our robot (front, fright, fleft). And inside this function we call the take_action function.
- take_action:in the core of this function, we define a safe_-distance which our robot has to respect if it detects an obstacle in the front, right or left.

Now that we have our necessary functions, our robot can perform different tasks of the seconde challenge as follow:

- if we don't detect any obstacle around us(the obstacle is the wall): then if the wall detected is closer to the right side of the robot we turn left and move forward, else we turn right and advance.
- if we detect the wall in the front, and the wall detected is closer to the right side of the robot we turn left else we turn right.
- the velocity commands are published to the topic '/cmd_-vel' as long as we are still inside the corridor (y coordinate $<$ to a certain value) , else we stop our robot.

Challenge 3: In this final challenge, we need to navigate the robot in a challenging environment, and then reach a final goal. In order to do this, we have to use both the laser and camera data. we define two callback functions, read_pose_callback and read_image_callback.

- For the obstacle avoidance, we use exactly the same algorithm used for the second challenge, all we have to change is the value of a few parameters.
- in the core of the read_image_callback, just like the first challenge, we compute the value of the moments of two masks, yellow and blue, if we detect a blue color, then we move our robot towards the point detected, if we detect the yellow color, we move our robot to the point detected for a certain time then we stop.
- in the core of the read_image_callback, we define a variable $detect$, to know if we detect a color which means we detected the goal, if it is equal to 0, no coor was detected , our robot has to avoid obstacles, else, our robot is outside the maze of obstacles and has to go to the goal and stop.

III. Performance characterization

- in the first challenge, we compute an error value, which is the difference between the center of the camera image and the detected point on the line, we will see that there is a relation between the value of the linear value we publish the topic '/cmd vel' and this error, we can notice that increasing the velocity value induces a bigger error, so we need to find the right equilibrium between the error and the velocity, because if the robot is too slow, the simulation time increases, if it is too fast, it will stay from the right path.

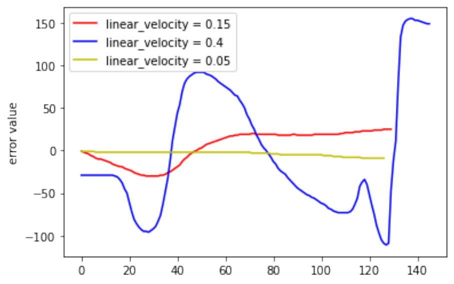

Fig. 5. Error.

IV. Conclusion and perspectives

Through the realization of this project, we were able to complete some robot tasks like line following and obstacle avoidance.The results were acceptable but the robot functions needed to be optimized.

We propose some improvements that could have been done to improve the performans of our robot:

- Self-adjusting the robot's velocity by looping to calculat the error of the detection point and the image center point until it converges to a certain smaller value (0.1 for exemple).
- Changing the velocity accoding to the detection area ,if there is no obstacle in front and a straignt line, we increase the velocity.If there is an obstacle or a turn detected, we will decrease the velocity of the robot.
- We could have used mapping method (mapping the environment that the robot is in through the camera and the Laser scan to map the way) once the mapping is done we use a shortest way algorithm (like dijikstra) to calculate the optimal path of the robot.