My ROS project

Hanlin LIU
Master SAR
Student number 3971558

Valentin COINTREL
Master SAR
Student number 21107824

***Abstract*—This is our LaTeX report on our ROS project. In this document, we will present the ROS architecture of our project and we will explain the choices we made that led to this architecture. We will also analyze the performance of our architecture and see what part of it can be improved.**

I. Introduction

The goal of this project was to create a ROS architecture that would be use by a turtlebot to navigate in different areas (challenge) where each area has its own difficulties. The robot has a camera and a LiDAR that gives informations to help him navigate.
The first area consists of a "road" represented by two lines of different colors. The goal of the robot is to follow these lines while remaining in the middle. The 3 difficulties are that: there is a roundabout; sometimes the road has tight turn and the last is that the road may have obstacle.
The 2nd area is a corridor in "U" shape. The robot has to enter the corridor and get to the exit without touching the walls on both of his sides. He can only use the LiDAR. The difficulty is to turn without touching the walls.
The 3rd area is a closed environment filled with obstacle. The robot has to enter this "box" and find the exit. The difficulty is that the robot cannot rely on his camera to find the exit since there is no light, so he only has to use the LiDAR to avoid obstacle and find the exit.

The report will be structured as follows:
In a first time, we will do a global presentation of our ROS architecture.
In a second time, we are going to get into the details of the ROS architecture. We will show the "thinking process" that the robot uses to navigate and complete the different tasks/challenges.
After that we will analyze the performance of the robot based on different tests and explain why we choose these tests.
We will then conclude and show the elements in our ROS architecture that can be improved.

II. Presentation of the ROS architecture

In this part, we will present the ROS architecture we have built to solve the objectives we mentioned in the introduction.

A. A short overview

The architecture is based around 2 main nodes:

We have the teleop node which is used to control the robot with the keyboard (mostly used for debugging the perception of the lines in the challenge 1).

We have the node challenge_ensemble which is the central piece of our architecture. The node gathers data from different topics and processes it inside different functions. These functions then determine, based on the data, the velocity commands (angular and linear) to be sent.

The node challenge_ensemble is subscribed to the */camera/image* topic so that he can receive the data published by the camera. The data is a message type *Image* from the *sensor_msgs.msg* package. It contains information relative to the image.

The node challenge_ensemble is also subscribed to the */scan* topic so that he can receive the data published by the LiDAR. The data is a message type *LaserScan* from the *sensor_msgs.msg* package. It contains information relative to the LiDAR. The information we use from this message is the *LaserScan.ranges* which is an array of floats containing the value of the distance between the robot and its environment (360°). All orientations where distance is measured are separated by 1 degree. The array therefore has 360 values.

The functions inside the node challenge_ensemble that analyze the data and send commands based on it are:

- *callback_follow*. This function converts the image into an usable format, then it creates 2 masks (one for each line to be detected) to only show/see the lines the robot has to follow. He then determines a point between the 2 lines that he has to follow. According to the difference (error) of position between this point, and the robot, the function calls another function. Either *tourne()* if the robot is in the part of challenge 1 where there are obstacles or *follow(error)* if the robot is in another part of challenge 1.
- *tourne()* and *follow(error)* publish a speed command through the topic */cmd_vel*. This topic transports *Twist* message type (this message type comes from the *geometry_msgs.msg* package). These messages have 3 linear components (x,y,z) and 3 angular components (x,y,z) so that we can control the robot by assigning values to one or more linear components if we want to move in translation and the same goes with the angular components if we want to rotate.
- *callback_obsta*. This function calculates the average distance in some orientation ranges (The ranges differ ac-

cording to the challenge). Below a certain treshold we considere that there is an obstacle (the treshold differs depending on the challenge). If we are in challenge 2 or 3 depending on the situation, the threshold and the existence of an obstacle, a function *publisher_obsta(linear, angular)* is called.
- *publisher_obsta(linear, angular)* publishes a speed command through the topic */cmd_vel* for turning or moving forward depending on the position of the obstacle. If we are in challenge 1, the *callback_obsta* simply stock the following informations inside global variable: if an obstacle is near and which side is it on. These informations will then be used by the *callback_follow* function when this function is called (as seen previously).

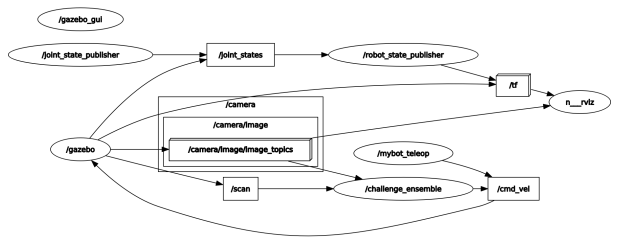

Fig. 1. Our ROS architecture.

In the figure above, the rectangles correspond to the topic and the ellipses (or circle) correspond to the nodes. The arrows indicate the transmission path of the different messages through the topics. /gazebo_gui is used to display the graphic user interface of gazebo. For a robot described with URDF /joint_state_publisher publishes values for all of the movable joints in the URDF to the /joint_states topic. /robot_state_publisher is used to calculate the forward kinematics of the robot and publish the results via tf. n__rviz is used for the representation of the robot in Rviz and for the camera visualization. /gazebo is the node corresponding to the Gazebo simulation where the robot and the environment are located. The other nodes and topics have been explained previously.

B. Algorithmic structure of the architecture

In this subsection, we will detail how all the nodes listed in our architecture work together for a given scenario. For instance:

- The node challenge_ensemble receives data from the camera. It calls the function *callback_follow*.
- Then the function *callback_follow* applies masks to the image, so the focus is only on the lines we want to detect and only on the part near the robot.
- It then determines the midpoint between the 2 lines by doing the average of the x coordinate of the center of each line. We know in which situation we are by counting the number of red lines we crossed on the track. Depending on the function that was called by *callback_follow*, a specific linear and angular speed command are sent to control the robot. (the values are not the same).
  - If we are in the part of challenge 1 where there are obstacles, then the *callback_follow* function calls the function *tourne()* with the error of position between the robot and the point in the middle of the 2 lines as a parameter.
  - Else, the function *follow(error)* is called.
- The node challenge_ensemble receives the data published by the LiDAR.
- It processes the data of the LiDAR inside a function called *callback_obsta*. This function calculates the average distance in some orientation ranges. (The ranges differs depending on the challenge)
- Below a certain treshold we consider that there is an obstacle. (the treshold differs depending on the challenge)
  - If we are in challenge 2 or 3 depending on the situation, the threshold and the existence of an obstacle, the function *publisher_obsta(linear, angular)* is called. This function publishes a speed command for turning or moving forward depending on the position of the obstacle.
  - If we are in challenge 1, *callback_obsta* simply stores the following informations inside global variable: if an obstacle is near and on which side it is. These informations will then be used by *callback_follow* when it is called (as seen previously).

After that the ”thinking process” restart

III. Performance characterization

In this section, we are going to analyze the performance of our architecture.

We decided to plot the evolution of angular velocity sent to the robot during each challenge in order to identify the moment when our architecture could be poorly optimized.

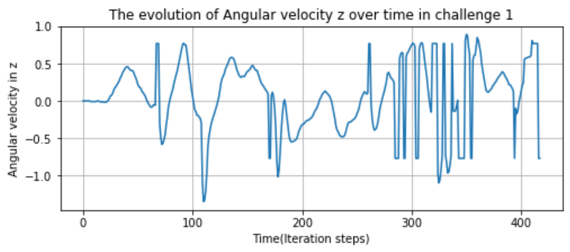

Fig. 2. Angular speed in challenge 1.

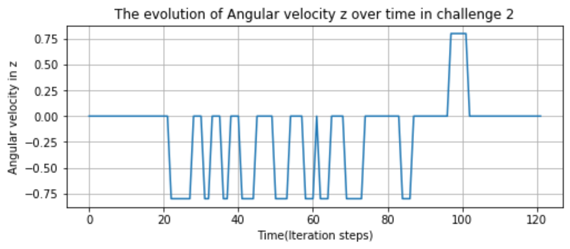

Fig. 3. Angular speed in challenge 2.

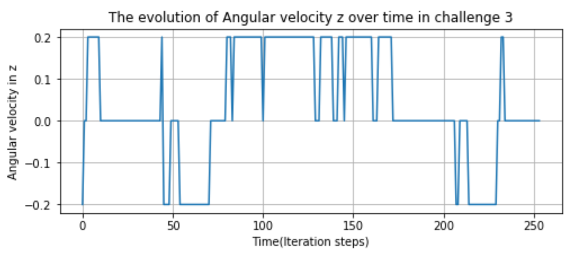

Fig. 4. Angular speed in challenge 3.

We can see that in challenge 2 and 3, the commands sent are almost always active for some amount of time and are not immediately offset by an opposite value which is a good thing. Unlike them, challenge 1 has some moments where the commands sent are immediately offset by an opposite value. We can see that from iteration 290 to iteration 350 which corresponds to the part of challenge 1 where there are obstacles. This observation shows that the movement of the robot is not smooth at all and a lot of the commands sent at that time are useless or need to be modified (reduce the value of the angular speed for example) so that the robot can obtain more time and margin to do his movement.

We also decided to plot the detection of the wall based on our processing of the LiDAR data for the challenge 2. We did this to see if an improvement could be made to make the robot movements smoother because each obstacle detection is followed by a rotation movement of the robot.

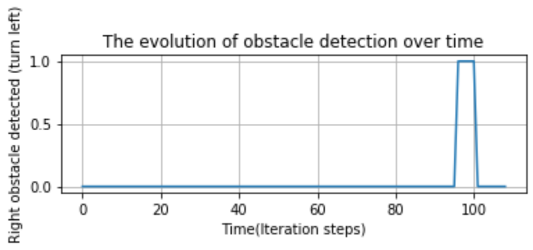

Fig. 5. Detection of obstacle on the right in challenge 2.

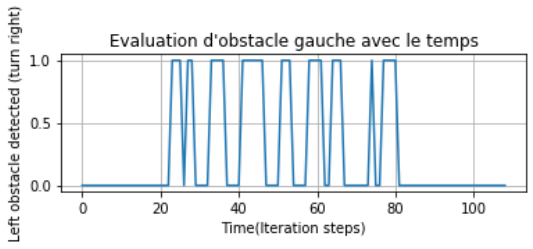

Fig. 6. Detection of obstacle on the left in challenge 2.

We can see that for a single turn on the course (challenge 2 is a U-shaped corridor) the robot has done at least 9 turn because of the detection of the left wall (left wall is the outer wall of the U turn). There is definitely a margin of improvement on the detection. We could reduce the angle of detection to reduce the number of turn needed. On the other side we can see a good behaviour which is that there is almost no immediate offset by an opposite turn which means the robot doesn't turn too much after detecting a wall. But as said before, the robot may not be turning enough.

We also decided to measure the time taken by the robot to do the different challenges in order to note the speed of the robot.

TABLE I
TIME TO COMPLETE EACH CHALLENGE

	Course areas			
Time	***Challenge 1****	***Challenge 2***	***Challenge 3***	***All Challenge***
T_{finish}	231 secondes	38 secondes	39 secondes	308 secondes

*Challenge 1 corresponds to the part that starts from the beginning of the course to the start of Challenge 2 and also to the part between Challenges 2 and 3. (ONLY TRUE FOR THIS TABLE !)

From the result we can see that challenge 2 takes the same amount of time as the challenge 3 even though the challenge 3 is much more complex and longer than the challenge 2. This observation confirms the observation made previously with the obstacle detection where we can clearly see that we are losing some amount of time turning in the challenge 2. The challenge 1 is the longest in terms of distance, so the results we get aren't surprising.

IV. CONCLUSION AND PERSPECTIVES

To conclude, this project has allowed us to learn how to control a robot by processing the data from the sensors and then determining the commands to be sent to the actuators according to the data. We've learned the big difference there is between the simulation and the reality. Most of the values determined in the simulation for the commands (speed,position,...) had to be adapted to reality. The way the robot behaves in real life can be a little different. Also the data we get from the sensors aren't as perfect as in the simulation which can be a big deal to control the robot. We also saw how usefull ROS packages and the general organisation of packages (launch file, scripts) can be when creating the architecture of the robot. It's easy to modify part of the code and then launching the simulation again.
If we had more time, we obviously would have tried to optimize the value of the commands sent to the robot as well as the treatment of the data in order to have a faster robot. Challenge 1 and 3 are the ones where we can improve a lot more for the real robot in particular. We would also have tried to make the code more general and less specific to the challenge that the robot was facing.