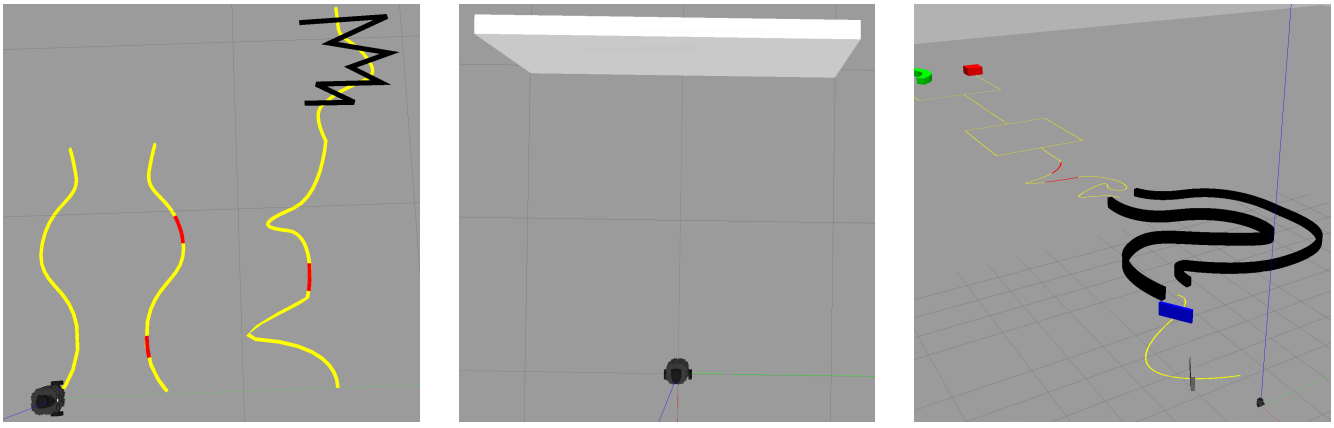
ROS and experimental robotics.
Part 3: project.

1 Context, goal and evaluation of the project

In this project, you will control a simulated Turtlebot 3 burger[1] mobile robot in various environments, so as to make it navigate from one starting position to a goal position depending on the task to solve. The navigation will successively exploit:

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

The project will take the form of 3 different challenges associated to the 3 previous points, each of them being separated in 3 or 4 tasks of increasing complexity, as show in figure 1.



(a) Challenge 1 (b) Challenge 2 (c) Challenge 3

Figure 1: Illustration of some of the gazebo worlds you will work on for each challenge.

1.1 Description of the organization of your work

All your developments (code, launchfiles, scripts, gazebo worlds, configuration files, etc.) must be placed inside one and unique ROS package named `challenge_project`. You can find on Moodle two different launchfiles `rviz.launch` and `gazebo.launch` which are already written so as to display on `rviz` the Turtlebot 3 burger and its sensor data, and to spawn a Turtlebot 3 burger inside an empty world in `gazebo`, respectively. Of course, these two files only serve as a basis for your requirements, and you are free to modify them accordingly.

#1.1.1. Create a package `challenge_project` in your `catkin` workspace, with all the dependencies needed for writing code in Python, `std_msgs`, `turtlebot3_msgs` and `cv_bridge`.

#1.1.2. Create all the usual folders inside your package to store all the files available for download from Moodle in the "Part 3" section (launchfiles, `rviz` configuration file, urdf robot description, etc).

#1.1.3. Use the provided launch files to launch `gazebo`. Adapt your teleoperation node written during part 1 so as to make the robot move inside the empty environment. You can also add an obstacle in the environment to check if the data sent by the LDS are consistent.

#1.1.4. Use the provided launch files to launch `rviz`. Test if everything is working fine: you should see the image sent by the camera and the laser data on `rviz`. **Please launch `rviz` only when needed as it may require additional processing power. It is a tool used here for visualization and debug only.**

Congratulations, you are now ready to work on the project!

[1] If it seems complicated for your personal computer to exploit the simulated Turtlebot 3 burger (as it may require more processing power), you can use the `robot_with_sensors`, endowed with a camera and a LDS sensor, from part 2 instead.

1.2 Evaluation

At the end of the project, you will have to send us a compressed version of your `challenge_project` package via Moodle.

> You are free to write as many nodes as you want, and organize your package at your will. The only mandatory requirement is that you write a launch file for each challenge and each task, with a naming scheme like `challengeX_taskY.launch`.

For the evaluation, your teachers will use these launchfiles to check if the task `Y` of the challenge `X` is completed. **If the launchfiles do not fill the above name requirements, your evaluation will not be possible!** Each launch file must launch all the required nodes (including `gazebo`), set all the required arguments and parameters, so that no other commands have to be used to assess your work.

You must also write at the root of your package a `README` file (pdf, raw text, etc.), with at least the following content:

- your names, student number and group,
- the list of the folders inside your package, and the purpose of each file inside. A very short description is sufficient,
- for each of your Python script, please detail the main concepts used in the script, i.e. explain in a few lines the core concepts used in your algorithm, **but with no code at all!**

Mastering all the tasks for all challenges is not sufficient for you to reach the best grade. You must also carefully comment all your code, whatever the language. In the end, the quality of your code will be also evaluated, together with the fulfillment of traditional ROS *good practices* (like, not calling a service everytime a topic is changed, etc.). **You are allowed to work with one college of yours.** Only one ROS package must then be uploaded to Moodle. Do not forget to write down your two names inside the `README` file!

Evaluation

The detailed evaluation grid is provided on page 9, indicating the grade corresponding to each achievement.

2 Challenge 1: line following

For this first challenge, you are supposed to code one or multiple nodes working together so that the Turtlebot 3 follows a line in the environment. We provide one `gazebo` world for this first challenge; you can download the corresponding `challenge1.world` on Moodle, see Figure 2. Place it in the `worlds` folder of your `challenge_project` ROS package.

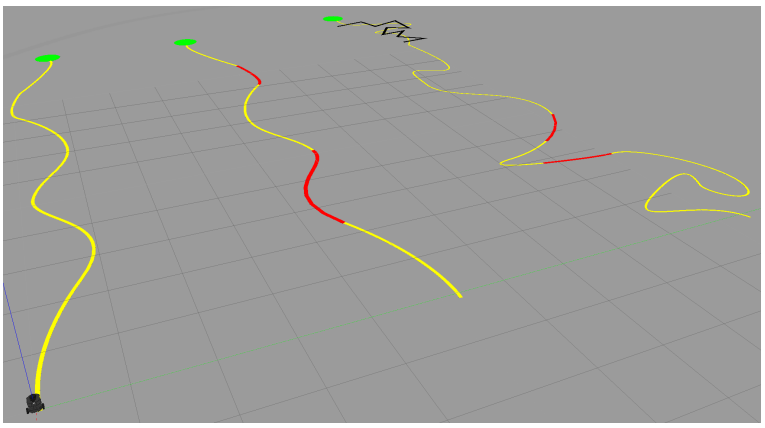


Figure 2: `gazebo` world for the first challenge and its different tracks corresponding to the tasks to be solved.

When loading the `challenge1.world` on `gazebo`, you will see three disconnected lines, each of them corresponding to different tracks of increasing complexity. Each track is linked to a task described in the next subsections.

2.1 Task 1: a simple line following algorithm

This task in mainly concerned with the following of a line printed on the floor of the world. This line is continuous and smooth, and is only made of one homogeneous yellow color. In all this project, you must use the `OpenCv` library to process the image made available by `gazebo` on the corresponding ROS topic. You can find at the following address some tutorials showing how to write Python code with `OpenCv`: https://docs.opencv.org/master/d6/d00/tutorial_py_root.html.

For this task, you will mainly have to:

- build a new node which subscribes to the image topic published by `gazebo` ;
- use basic `OpenCv` functions to process the image. Since you are trying to detect a yellow line at the bottom of the image, a simple binarization of the image, followed by an image moment computation (see https://en.wikipedia.org/wiki/Image_moment) should be sufficient to extract some simple information you can use to drive the robot from the image ;
- finally, use this information to actually send a velocity command to the robot.

One code example is available on the part 2 subject of this teaching unit (see #2.3.15) showing how to use `OpenCv` functions in Python. You can also use visualization capabilities of `OpenCV` to display the current image, the processed image, and the feature you are extracting from it (by displaying a text, a point, etc.). Importantly, **you must command the robot with the maximal usable robot velocity**, i.e. the maximal velocity for which your line following algorithm is working. Importantly, this maximal speed can depend on your computer processing capabilities.

Evaluation

For this task, you must write a launch file entitled `challenge1_task1.launch` (i) spawning the robot at the beginning of the first (entirely yellow) track with a correct orientation, and (ii) making the robot start its movement along the line. **The robot should stop as soon as the line is not visible anymore in the image.**

2.2 Task 2: robot speed as a function of the line color

This task is almost the same as the first one. The main difference is that there is still one line to follow, but now with two different colors:

- when the line is yellow, the robot must follow the line at maximal usable velocity;
- as soon as a red line is perceived, the robot must slow down significantly.

Thus, you have to detect two different colors in the image and to modify the robot behavior accordingly. Basically, it means you have to define two different binarization strategies corresponding to the two lines color so as to adapt the velocity as a function of the detected color.

Evaluation

For this task, you must write a launch file entitled `challenge1_task2.launch` (i) spawning the robot at the beginning of the second (yellow and red) track with a correct orientation, and (ii) making the robot start its movement along the line, and adapting its speed to the current line color. **As before, the robot should stop as soon as the line is not visible anymore in the image.**

2.3 Task 3: a more challenging environment

To end this first challenge, a third track is proposed in `challenge1.world`. The line is still with two colors (yellow and red), and the robot has still to adapt its speed as a function of the line color. Now, the line to follow is not smooth anymore and exhibits sharp peaks in the yellow zone corresponding to the high velocity command. It is clear that if the robot does not adapt its speed to the current condition, the line will not be visible in the image anymore. Two strategies might be combined to solve this issue: (i) even if the order is to move as quickly as possible when perceiving a yellow line, you have to adapt the robot velocity command to the "sharpness" of the line to follow, and (ii) if the line is lost in the image, you have to code a "line search" strategy to come back to the traditional "following line" scenario.

You will see at the end of the third track that the yellow line to follow might be blurred and masked by some other black line. This the final bonus challenge you have to deal with: how to drive the robot when the line is sometimes not visible? You are here free to propose your own solution to this issue. The robot must stop its movement at the end of the track. The goal position is indicated by a green area you can detect in the image to trigger the end of the movement.

Evaluation

For this task, you must write a launch file entitled `challenge1_task3.launch` (i) spawning the robot at the beginning of the third (yellow and red) sharp track with a correct orientation, and (ii) making the robot start its movement along the line, and adapting its speed to the current line color and the sharpness of the line. **The robot must stop its movement as soon as the target position is reached (green area).**

3 Challenge 2: obstacle avoidance

For this second challenge, you have to exploit only the laser data to code some obstacle avoidance behavior. Again, we provide one `gazebo` world named `challenge2.world` in which all the following tasks must take place. As shown in figure 3, this world is made of a simple rectangular wall the robot must avoid or reach depending on the task to solve.

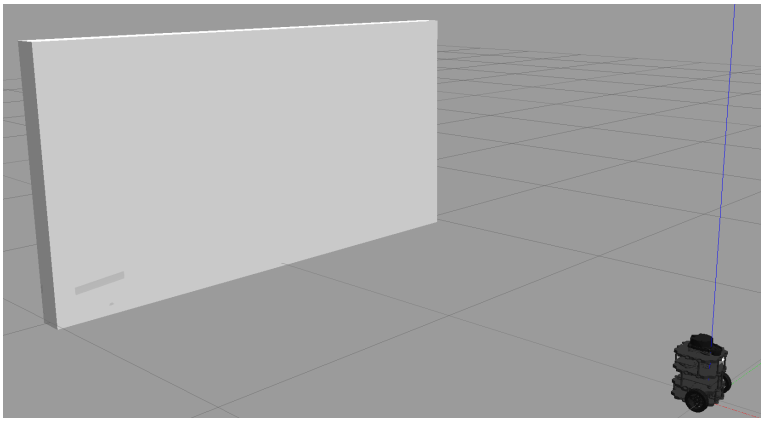


Figure 3: `gazebo` world for the second challenge. Depending of the task, the wall will move forward and/or backward.

This time, no specific library like `OpenCV` is required since the data sent by the simulated LDS is nothing more than a vector of (double) values.

3.1 Preparation of the environment

#3.1.1. Download from Moodle the file `challenge2.world` and place it in the `worlds` folder of your ROS package `challenge_project`.

#3.1.2. Download from Moodle the file `challenge2_scripts.zip` and unzip it in the `scripts` folder of your ROS package `challenge_project`. Do not forget to make the files executable with the following command:

```
$ roscd challenge_project
$ cd scripts
$ chmod +x challenge2_task2_world_control.py
$ chmod +x challenge2_task3_world_control.py
```

3.2 Task 1: emergency stop

For this first task, you have to implement an emergency stop behavior based only on the data sent by the laser sensor.

Evaluation

For this task, you must write a launch file entitled `challenge2_task1.launch` (i) spawning the robot in the world center and in the direction of the wall, (ii) making the robot move towards the wall, and (iii) making the robot stop when reaching a safety distance that can bet through a ROS parameter.

3.3 Task 2: laser-based distance servoing

In this task, the wall is not fixed anymore. It will move forward and backward in the direction of the robot and in front of it. The robot must maintain a constant distance to the moving wall, and thus follow its movement, with the same dynamic as the wall movement (i.e. the robot must be reactive enough to quickly follow the change of direction in the wall movement).

For this task, you have to add the following code to your launch file:

Listing 1: code to insert in your launch file

```
<!— Challenge 2, task 2 —>
<node name="obstacle_control" pkg="challenge_project" type="challenge2_task2_world_control.py"/>
```

This additional code will run a node in charge of making the wall move in the environment. Without it, the wall will remain still like for task 1.

Evaluation

For this task, you must write a launch file entitled `challenge2_task2.launch` (i) spawning the robot in the world center and in the direction of the wall, (ii) making the robot maintaining a constant distance d to the moving wall. This distance d must be set through a ROS parameter with default value at 20cm.

3.4 Task 3: laser-based moving obstacle avoidance

In this task, the wall still moves in the environment. Differently from task 2, the wall will now move towards and backwards from any direction around the robot. The robot still has to maintain a fixed distance d perpendicularly to the moving wall, even if the wall is moving from the right or left of the robot.

For this task, you have to add the following code to your launch file:

Listing 2: code to insert in your launch file

```
<!— Challenge 2, task 3 —>
<node name="obstacle_control" pkg="challenge_project" type="challenge2_task3_world_control.py"/>
```

This additional code will run a node in charge of making the wall move in the environment from a random position. Without it, the wall will remain still like for task 1.

Evaluation

For this task, you must write a launch file entitled `challenge2_task3.launch` (i) spawning the robot in the world center, (ii) making the robot maintaining a constant distance d to the moving wall whatever the position of if. This distance d must still be set through a ROS parameter with default value at 20cm.

4 Challenge 3: visual and laser navigation

For this last challenge, you will combine all your previous developments so as to allow a robot to navigate in a complex environment on the basis on visual **and** laser information. Again, a specific `gazebo` environment is provided as a `challenge3.world` file together with other folders and files that must be placed correctly inside your ROS package.

4.1 Preparation of the environment

#4.1.1. Download from Moodle the file `challenge3.world` and place it in the `world` folder of your ROS package `challenge_project`.

#4.1.2. Download from Moodle the file `challenge3_script.zip` and unzip it in the `scripts` folder of your ROS package `challenge_project`. Do not forget to make the files executable with the following command:

```
$ roscd challenge_project
$ cd scripts
$ chmod +x challenge3a_world_control.py
$ chmod +x challenge3b_world_control.py
```

#4.1.3. Finally, write a launch file to use `challenge3.world` in gazebo. You should then see a word like in Figure 4.

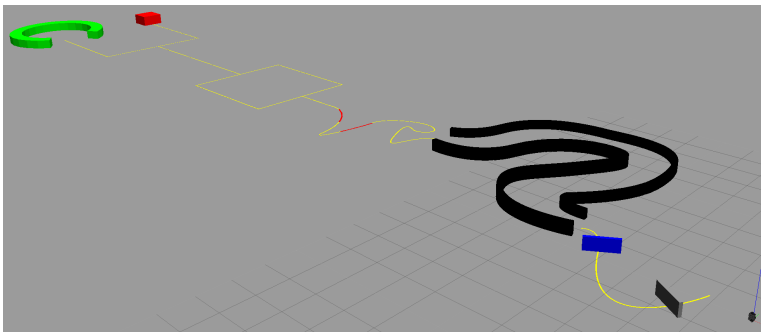


Figure 4: `gazebo` world for the third challenge.

4.2 Task 1: line following and obstacle avoidance

In this first task, you have to integrate your line following algorithm together with your obstacle avoidance technique to make the robot stop when required. Basically, two successive obstacles have been placed along the path of the robot:

- the first one is a moving wall which is automatically "opened" from time to time. The robot has to stop when facing the wall, and must then continue to follow the line on the ground once the obstacle has moved away ;
- the second one behaves like a garage door: the robot has to stop its movement when facing the door, and must then send a message on a specific topic to open it.

In order to make these obstacle active in the environment, you have to add the following code in your launch file:

Listing 3: code to insert in your launch file

```
<!-- Moving obstacle -->
<node name="OBSTACLE" pkg="challenge_project" type="challenge3b_world_control.py"/>
<!-- Garage door -->
<node name="DOOR" pkg="challenge_project" type="challenge3a_world_control.py"/>
```

This additional code will run the two nodes in charge of making the two obstacles active in the environment. Without it, both will remain still and the robot will not be able to go further.

A specific topic is made available to open the "garage door" obstacle. You have to investigate the running ROS architecture to find this topic and understand the structure of the message that must be send to open the door.

Evaluation

For this task, you must write a launch file entitled `challenge3_task1.launch` (i) spawning the robot at the beginning of the yellow line in the environment and with the correct orientation, (ii) making the robot follow the line, and (iii) making the robot wait for the obstacle to go away, either by simply waiting (obstacle 1), or by sending a message on the required topic (obstacle 2).

4.3 Task 2: movement in a corridor

After opening the door, the robot will reach an area where no line is present anymore. Instead, a corridor is available to guide the movement of the robot on the basis on the laser data only. You then have to switch from a visual based navigation (line following) to a laser-based navigation task, and the opposite when reaching the end of the corridor. The robot has to move at the center of the corridor, and to follow the left and right walls to the exit where a line is again available. At the exit, the same scenario as in Challenge 1 is proposed, where two line colors are used to modulate the robot speed.

Evaluation

For this task, you must write a launch file entitled `challenge3_task2.launch` (i) solving first the previous task 1 of Challenge 3, (ii) making the robot switch from visual navigation to LDS navigation and move at the center of the corridor and (iii) switching back to visual-based navigation where the line color is used to modulate the robot speed.

If do not succeed in making the robot navigating inside the corridor, you can directly spawn the robot after it so as to work on the following task 3.

4.4 Task 3: navigation to the target position

To end this 3rd challenge, the robot will have to correctly handle the presence of a bifurcation leading to the target position. The robot will then have to choose one path among two, and then go back to the main track. If the chosen path is a dead end, it then must go back and chose the other one. The wrong path is indicated by a fixed orange obstacle whose position among the two possible paths is random: each time you launch `gazebo`, the obstacle might be on a different path.

The same applies to the final bifurcation leading to the target position, represented as green area. The path to the target area can be blocked by a second, orange, randomly placed, obstacle. But this obstacle can not be detected by the LDS (its collision box is not defined) so that it can only be avoided from visual data.

Evaluation

For this final task, you must write a launch file entitled `challenge3_task3.launch` (i) implementing the previous task 1 and task 2 of Challenge 3, (ii) making the robot choose one path among other ones, and correct itself if this choice does not make the robot reach the final position.

Launch files provided for the beginning of the project

Listing 4: rviz.launch file

```
<?xml version="1.0"?>
<launch>

  <!-- load robot URDF description-->
  <param name="robot_description"
         command="xacro --inorder $(find challenge_project)/urdf/turtlebot3_burger.urdf.xacro"/>

  <!-- send fake joint values -->
  <node name="joint_state_publisher" pkg="joint_state_publisher" type="joint_state_publisher">
    <param name="use_gui" value="False"/>
  </node>

  <!-- combine joint values -->
  <node name="robot_state_publisher" pkg="robot_state_publisher" type="state_publisher">
  </node>

  <!-- show in Rviz, WITH config file -->
  <node name="rviz" pkg="rviz" type="rviz" args="-d $(find challenge_project)/rviz/configuration.rviz"/>

</launch>
```

Listing 5: gazebo.launch file

```
<?xml version="1.0" encoding="UTF-8"?>
<launch>

  <!-- load robot URDF description -->
  <param name="robot_description"
         command="xacro --inorder $(find challenge_project)/urdf/turtlebot3_burger.urdf.xacro"/>

  <!-- robot initial position -->
  <arg name="x" default="0"/>
  <arg name="y" default="0"/>
  <arg name="z" default="0.0175"/>
  <arg name="roll" default="0"/>
  <arg name="pitch" default="0"/>
  <arg name="yaw" default="1.5708"/>

  <!-- simulation parameters -->
  <arg name="world" default="empty"/>
  <arg name="paused" default="false"/>
  <arg name="use_sim_time" default="true"/>
        <arg name="gui" default="true"/>
        <arg name="debug" default="false"/>

  <!-- launch gazebo server and client from empty_world -->
  <include file="$(find gazebo_ros)/launch/empty_world.launch">
    <!-- Note: the world_name is with respect to GAZEBO_RESOURCE_PATH environmental variable -->
    <arg name="world_name" value="$(find challenge_project)/worlds/challenge1.world"/>
    <arg name="paused" value="$(arg paused)"/>
    <arg name="use_sim_time" value="$(arg use_sim_time)"/>
    <arg name="gui" value="$(arg gui)"/>
    <arg name="debug" value="$(arg debug)"/>
  </include>

  <!-- spawn model in world -->
  <node name="mybot_gazebo" pkg="gazebo_ros"
        type="spawn_model" output="screen"
        args="-urdf -param robot_description -model mybot
                        -x $(arg x) -y $(arg y) -z $(arg z)
                                -R $(arg roll) -P $(arg pitch) -Y $(arg yaw)"
  />

</launch>
```

Evaluation grid

ROS & experimental robotics: part 3

	Unacceptable : 0 point	Insufficient : 2.5 points	Objective reached : 7.5 points	Beyond : 10 points	Points
Challenge 1 : line following	The provided launch file does not work and/or does not the naming requirements.	Task 1 is working successfully, and the robot is able to follow a yellow smooth line. The robot stops its movement at the end of the line.	Task 2 is working successfully, and the robot is able to follow a line and to adapt its speed as a function of the line color.	Task 3 is working successfully, and the robot is able to adapt its speed depending of the line color or the line sharpness.	
Challenge 2 : obstacle avoidance	The provided launch file does not work and/or does not the naming requirements.	Task 1 is working successfully, and the robot is able to stop its movement when detecting an obstacle in front of it.	Task 2 is working successfully, and the robot is able to avoid a moving obstacle from a fixed direction.	Task 3 is working successfully, and the robot is able to aboid a moving obstacle, in every direction.	
Challenge 3 : visual and LDS based navigation	The provided launch file does not work and/or does not the naming requirements.	Task 1 is working succesfully, and the robot is able to follow a line, to stop itself in front of a fixed obstacle, and to continue its way to the target position by acting on the obstacle (like a garage door).	Task 2 is working successfully, and the robot is able to switch from visual (line following) to LDS (wall following) to reach the target position.	Task 3 is working successfully, and the robot is able to reach the final target position despite all the obstacles and difficulties in the challenge3 gazebo world.	
Code	The code is not commented at all, and sometimes very hard to understand.	The ROS package is not well organized, the code is sometimes not commented at all,or the comments does not allow to catch the meaning of the code. Some ROS good practices are not respected.	The ROS package is correctly organized, the code is correctly commented and the comments correctly explain all the root concepts used to solve the different challenges.		

Negative modulation as a function of the README file content (up to -10 points)	
TOTAL (/50)	

Noms des étudiants :