ROS and experimental robotics.
Part 3: project.

1 Context, goal and evaluation of the project

In this project, you 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 some lines,
- 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

Figure 1: Illustration of the gazebo world you will work on.

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 `projet`. You can find on Moodle the corresponding package, including the simulated world we provide for the project (see Figure 2), together with the corresponding launchfile spawning a turtlebot 3 burger endowed with a LDS and camera sensors. Of course, these files only serve as a basis for your requirements, and you are free to modify them accordingly[1]. A `rviz` config file allowing you do display the robot, the LDS data and the acquired image is also provided.

#1.1.1. Unzip the package `projet` in your `catkin` workspace, launch `catkin_make` and source the `devel/setup.bash` file. If needed, use the `rospack profile` command to refresh your workspace.

#1.1.2. Use the provided launch file to launch `gazebo` and `rviz`. Test your teleoperation node written during part 1 so as to make the turtlebot move inside the provided environment, and check the consistency of the simulated sensors. **Please launch `rviz` and/or `gazebo` only when needed as it may require additional processing power. Also, do not hesitate to exploit `rosbag` to record and replay data from simulations (Gazebo) and/or the real robot.**

#1.1.3. On the basis on the provided launchfile, write a new launchfile dedicated to the real robot and allowing you tu visualize in `rviz` the sensors data.

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

[1] But please **do not modify** in any way the files in the different `turtlebot3` packages or worlds files.

1.2 Evaluation

At the end of the project, you will have to:

- send a compressed version of your `projet` package via Moodle;
- write a report on your work (more details below);
- make a demonstration of your work on the real robot during a competitive "race" against each other.

1.2.1 Organization

> 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 one/multiple launch file(s) allowing to launch everything required in simulation so as to assess your work.

For the evaluation, your teachers will use this(ese) launchfile(s) to check if the tasks are completed. 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. But note that mastering all the tasks 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.).
Practically, we propose the following organization:

- you can work by two, regardless of your initial group. No more than 1 group of 3 students per specialty (ISI, SMR, SAR) is allowed;
- you have now entire days of practicals (on Friday for ISI studients, on Monday for SAR/SMR students);
- for your organization, and because the real robot might not be alwayse available, use extensively `rosbag` to record and replay real data so as to design your algorithms.

Only one ROS package must then be uploaded to Moodle per student group. Do not forget to write down your two names inside some kind of `README` file at the root of your package! It is also important to understand that the "simulation" part of the project must be seen as a way to test, evaluate, assess strategies that should then be implemented on the real robot. This transition from simulation to reality is one of the key point in ROS architectures, and its versality should help you in the process, providing you develop everything in a generic way (with parameters for instance). Again, you can also benefit from recordings, **rosbag**, etc. to develop offline algorithms that should work directly on the real robot, with real-time sensor data.

1.2.2 Report

In addition to a zip/gz archive of your code, you also have to write a LaTeX report, with the following requirements:

- IEEE format, written in English;
- up to 3 pages, doubled-sided, 2 columns;
- with a mandatory organization given in the template available on Moodle.

Your document shall include detailed technical information regarding the ROS architecture of your solution, including the list of the used nodes, the topics/messages they are using to communicate, etc. In particular, you have to include a mandatory figure representing your architecture which must be clearly explained in the text. The exact form/representation of this figure is up to you but must be self-contained. The technical information should not only basically list the nodes you are using, but also focus on the reason of your choices (which node, to do what, etc.) and justify the role of each element in your architecture.

In addition to the previous explainations, it is also expected you provide a performance analysis of your architecture in terms of e.g. speed, servoing errors statistics, precision, etc (non-exhaustive list). To that end, you have to define, compute and analyze some features of your choice and discuss the results you obtained.

A LaTeX template is available on Moodle. The resulting document is shown on Page 8.

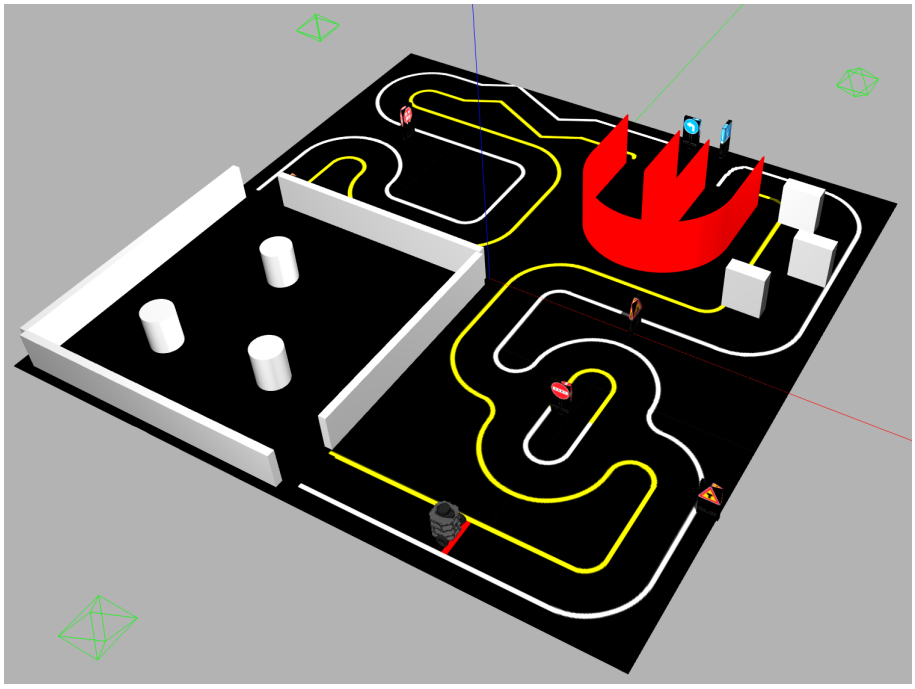


Figure 2: The entire `gazebo` world, with the 3 three challenges to solve: (i) line following and obstacle avoidance, (ii) movement in a corridor, and (iii) navigation in a cluttered environment.

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 some lines in the environment. But this challenge is not only about image processing, since there will be also obstacles on the way to the second challenge. You can also notice that two different line colors are used on the left and on the right sides of the track, which sometimes split up when facing an intersection. This track following can also be tested with the real robot: the robot must follow these lines made of a coloured tape stuck on the floor (the exact color might be different from the one used in simulation), and detect if an obstacle is present in front of it and navigate accordingly (this can involve an emergency stop behavior).

In all this project, you must use the `OpenCv` library to process the image made available by `gazebo` or the real robot on the corresponding ROS topic. You can find at the following addresses some tutorials showing how to write Python code with `OpenCv` `https://docs.opencv.org/master/d6/d00/tutorial_py_root.html`, see also the documentation in `https://docs.opencv.org/4.2.0/pages.html`.

For this task, you will mainly have to:

- build a new node which subscribes to the image topic published by `gazebo` or the real robot;
- interface ROS and OpenCV together; it requires a bridge to convert the image from the ROS `sensor_msgs/Image` message to the correct OpenCV class. `CvBridge` provides the required functions to perform this conversion, see `https://wiki.ros.org/cv_bridge/Tutorials/ConvertingBetweenROSImagesAndOpenCVImagesPython`
- use basic `OpenCV` functions to process the image. Since you are trying to detect colored lines in the image, a simple binarization of the image, followed by an image moment computation (see `https://en.wikipedia.org/wiki/Image_moment`) might be an interesting step to extract some simple information you can use to drive the robot from the image, see Listing 1 on page 7 ;
- finally, use this information to actually send a velocity command to the robot.

Some indications are provided in the following, guiding you on 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.).

Evaluation

Simulation: for this task, you must write a launch file entitled `challenge1.launch` (i) spawning the robot at the beginning of the track with a correct orientation, and (ii) making the robot start its movement along the line, navigate among obstacles and eventyally stop accordingly if required (emergency stop).

3 Challenge 2: movement in a corridor

For this second task, you have to code one or multiple nodes allowing the robot to navigate inside a corridor, made of two adjacent walls guiding the robot from the first challenge (lines following) to the third. It is clear here that the robot movement must be inferred mainly from the LDS data, with a robot moving at the center of the corridor and following the left and right walls to the exit. While this task can be solved in simulation, we also plan to build a small corridor for the real robot. The objectives remain the same, and the robot must navigate safely between the two walls to the exit.

Evaluation

Simulation: for this task, you must write a launch file entitled `challenge2.launch` (i) spawning the robot at the beginning of the corridor with a correct orientation, and (ii) making the robot start its movement between the walls and stop at the exit.

Note that if the simulation is oriented towards the validation of the navigation in the corridor only, you will have to think about the succession of the two first challenges: first navigate on the basis on visual data mostly, and then use the LDS to guide the robot between two walls. The way the simulated and real robot will handled the succession of these two tasks is of particular importance and must be dealt with correctly.

4 Challenge 3: navigation in a cluttered environment

In this third task, you have to code one or multiple nodes allowing the robot to navigate among multiple obstacles placed randomly in the environment. Obviously, the robot must not touch any obstacles during its movement and it must be driven towards the exit. The exit is materialized by two green posts placed randomly around the blue square delimiting the zone of this last challenge.

In this task, both sensors (LDS and camera) might be used to make the robot reach this target: the goal here is then to make the robot reach the exit.

The same kind of objectives must be reach with the real Turtlebot 3 burger. A cluttered environment will be used to assess the robot ability to navigate and reach the exit.

Again, the ability of your architecture to switch smoothly from the navigation in a corridor to a navigation in a cluttered environment must be carefully addressed !

Evaluation

Simulation: for this task, you must write a launch file entitled `challenge3.launch` (i) spawning the robot at the beginning of the cluttered environment with a correct orientation, and (ii) making the robot start its movement between the obstacles and stop when reaching the exit.

5 Conclusion: towards a competition between students on the real robot

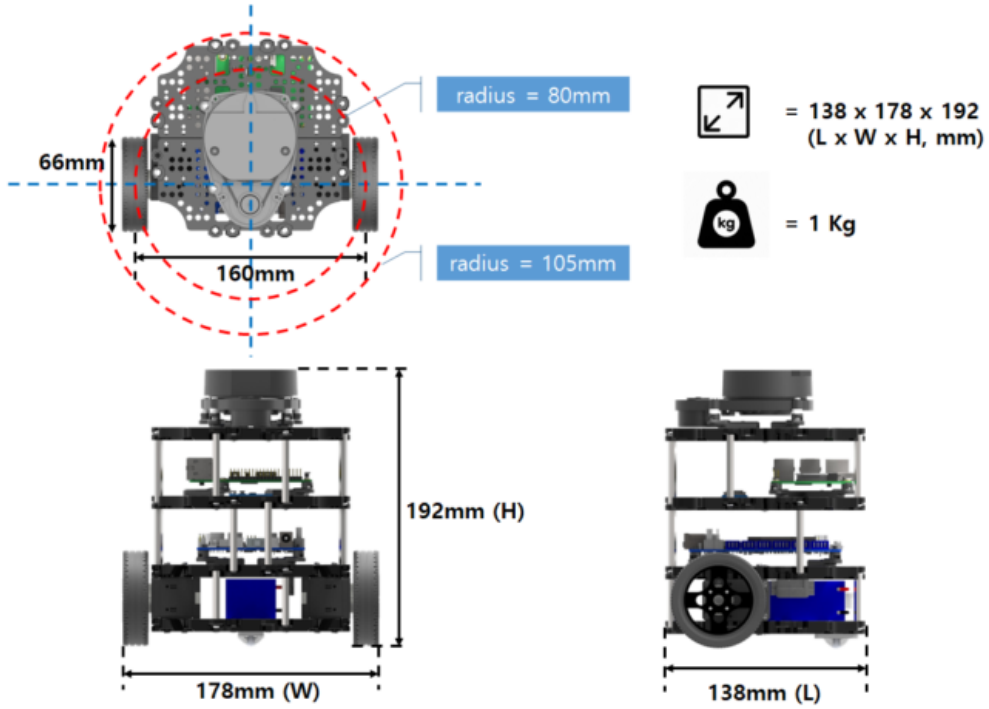
During the last week of the semester, we will organize a contest between students teams with the real robot. Your robot will have to realize all the previous tasks, at once, as quick as possible. Two tracks should be made available so as to face two teams with the same goal: reaching the finish line. Basically, the winner will be declared on the basis on:

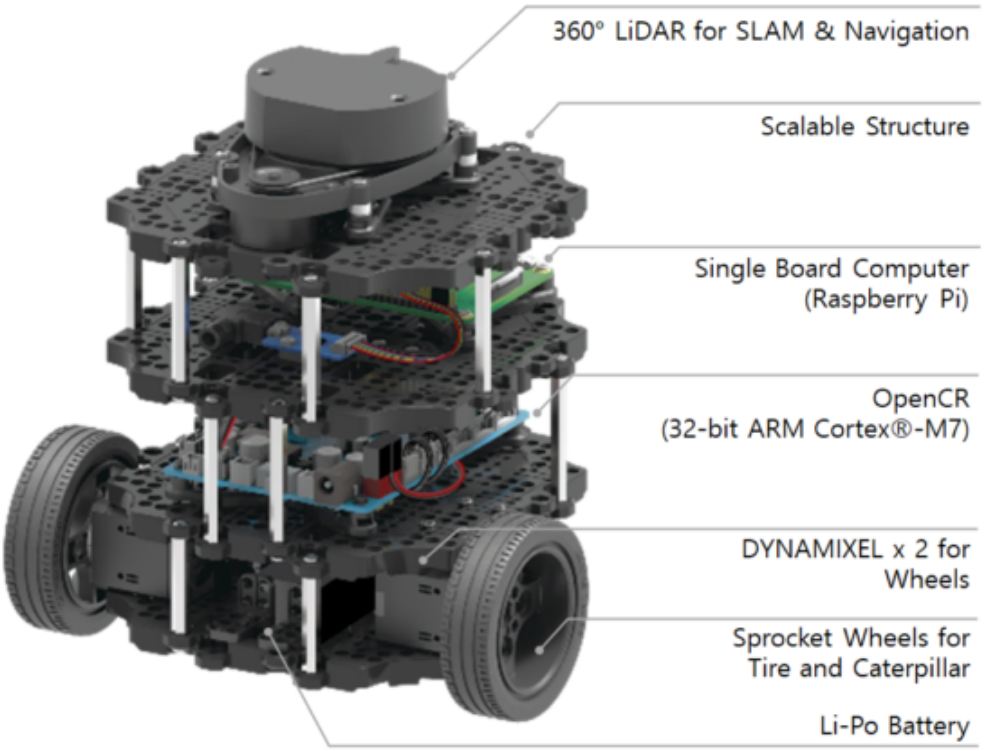
- the time taken to reach the target, which should be as short as possible;

- the number of collision on the path to the target (obstacles, walls, etc.). In practice, each collision will produce some penalty to the team;

The exact conditions of the context will be communicated as soon as possible and will depend on the conditions we will have in May.

6 Appendix 1: technical datasheet of the Turtlebot 3 burger





Items	Burger
Maximum translational velocity	0.22 m/s
Maximum rotational velocity	2.84 rad/s (162.72 deg/s)
Maximum payload	15kg
Size (L x W x H)	138mm x 178mm x 192mm
Weight (+ SBC + Battery + Sensors)	1kg
Threshold of climbing	10 mm or lower
Expected operating time	2h 30m
Expected charging time	2h 30m
SBC (Single Board Computers)	Raspberry Pi 3 Model B and B+
MCU	32-bit ARM Cortex®-M7 with FPU (216 MHz, 462 DMIPS)
Remote Controller	-
Actuator	XL430-W250
LDS(Laser Distance Sensor)	360 Laser Distance Sensor LDS-01
Camera	-
IMU	Gyroscope 3 Axis Accelerometer 3 Axis Magnetometer 3 Axis
Power connectors	3.3V / 800mA 5V / 4A 12V / 1A
Expansion pins	GPIO 18 pins Arduino 32 pin
Peripheral	UART x3, CAN x1, SPI x1, I2C x1, ADC x5, 5pin OLLO x4
DYNAMIXEL ports	RS485 x 3, TTL x 3
Audio	Several programmable beep sequences
Programmable LEDs	User LED x 4
Status LEDs	Board status LED x 1 Arduino LED x 1 Power LED x 1
Buttons and Switches	Push buttons x 2, Reset button x 1, Dip switch x 2
Battery	Lithium polymer 11.1V 1800mAh / 19.98Wh 5C
PC connection	USB
Firmware upgrade	via USB / via JTAG
Power adapter (SMPS)	Input : 100-240V, AC 50/60Hz, 1.5A @max Output : 12V DC, 5A

7 Appendix 2: starter code for OpenCV

Listing 1: Code example for OpenCV

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import cv_bridge
import cv2

cvBridge = cv_bridge.CvBridge()

# Transform the image to openCV format, msg is the original image from ROS
cvImage = cvBridge.imgmsg_to_cv2(msg, desired_encoding='bgr8')

# Change color represtation from BGR to HSV
hsv = cv2.cvtColor(cvImage, cv2.COLOR_BGR2HSV)

# Image binarisation
mask = cv2.inRange(hsv, ...)

# Compute the mask moments
M = cv2.moments(mask)

if M["m00"] > 0:

    # Calculate x,y coordinate of center
    cX = int(M["m10"] / M["m00"])
    cY = int(M["m01"] / M["m00"])

    # Display the image with cv2.imshow, or draw a form with cv2.rectangle or cv2.circle
```

My ROS project

Nicolas DUPONT
Master ISI/SAR/SMR
Student number 12345678

Robert DUPOND
Master ISI/SAR/SMR
Student number 12345678

***Abstract*—This is my wonderful LaTeX report on my ROS project. In this document, we will ...**

I. INTRODUCTION

This doubled-sided, 2 columns document must not exceed 3 pages, and should be written in English. It should be constituted of a limited number of figures and tables, and be oriented towards the analysis of your architecture and of its performance. It is *not* a purely technical report, we expect you to explain in this document your choices, to analyze their consequences, mainly in terms of performance w.r.t. the task to be solved.

The plan of the document (section/subsection organization) must not be changed. Neverthlesse, you can eventually add some subsections if really required.

II. PRESENTATION OF THE ROS ARCHITECTURE

In this section, you shall present the ROS architecture you build to solve the objectives you mentioned in the introduction. This presentation must be technical and go in depth when needed, so as to demonstrate you ability to master ROS concepts and to use them in an actual project.

A. A short overview

In this subsection, you can first briefly introduce your architecture: used nodes, topics, messages, services, etc. Then, you shall precisely describe some elements, and include and comment a mandatory figure/sketch like in Figure 1 representing the architecture topolgy, i.e. your nodes, how they communicate with each other, etc.

Fig. 1. My very simple architecture.

B. Algorithmic structure of the architecture

In this subsection, you must detail how all the nodes listed in your architecture work together for a given scenario. For instance:

- node A detects an obstacle on the basis on node B which compute the mean distance to a laser impact in front of the robot from the LDS sensor ;
- then, node C decides to stop the robot movement, and sends a request to the service exposed by node D ;
- so, node D decides wether the robot should (choice 1) stay at rest or (choice 2) move away from the obstacle:
  - if choice 1: node D sends ...
  - if choice 2: node D requests ...
- etc.

Such an algorithm, here described in the form of if/else/then statments, can also be described through a state machine that you should carefully and precisely formalize. You can obviously use an additional figure to support your discussions, if needed.

III. PERFORMANCE CHARACTERIZATION

In this section, you must analyzed the performance of your architecture. Such an analysis requires first the formal definition of indicators/cues (e.g. mean errors, response time, etc., like the fictitious H^∞ definition in Equation (1)) of your choice, which must be correctly chosen so as to assess the performance you are trying to evaluate (e.g. precision of the line following algorithm, stability of the servoing w.r.t. very curved lines, etc.).

$$H^\infty = \alpha + \Gamma. \tag{1}$$

Once chosen and defined, you have then to plot –in a specific figure or on a table like in Table I– the values of your indicator and comment its evolution and signification regarding the performance you are trying to reach.

TABLE I
MY SUPER PERFORMANCE INDICATOR

Table Head	Table Column Head		
	Table column subhead	*Subhead*	*Subhead*
H^∞	42%[a]	67%	89%

[a]Only when it works.

Feel free here to add subsection if it helps in understanding the different steps in your evaluation of your architecture performance.

IV. CONCLUSION AND PERSPECTIVES

You can use this last section to conclude on your work and explain what you could have improved if you had more time, how and why.