Final Project

ROS + Gazebo: Maze navigation using sensor data (Max score: 8)

AUT CE - Introduction to Robotics (Spring 2021)

OUT: July 3, 2021 (Tir 12, 1400)

DUE: Jult 22, 2021 (Tir 31, 1400) at 11:59pm - ***There will be no extension!***

Instructions

Policy

Final project is due on *courses.aut.ac.ir* by the posted deadline.

You can discuss the exercises with your classmates, but you should write up your own solutions inside your group. If you find a solution in any source other than the material provided in the course, you must mention the source.

The questions must be answered using Python + ROS.

Submission

Create a tar archive of the folder with your ROS package and submit it to **Final Project** on courses. You should also have one PDF file in your archive, with an explanation of your results regarding the experiments with the robot, and the instructions for running the nodes, if any.

1 Introduction

The goal of this project is to let you keep practicing with ROS and Gazebo while starting using sensor data for robot navigation. In particular, the task will consists in designing a *reactive controller* that can let your robot navigating *safely and effectively* in a *maze-like* environment using basic sensor data (i.e., in a closed-loop modality). Safe means without collisions with obstacles, effectively means able to move fast and smooth in the "right" direction (as explained below).

Since a number of new concepts and ROS components are needed to tackle the task, we have two new sections for document `start-with-ros.pdf`:

- *Section 4: On-board sensors: depth camera, laser scan, bumpers, cliff sensors*
- *Section 5: Setting the robot pose for simulation experiments*

Section 4 is fundamental for the project, therefore you need to go through it before starting with the project, while Section 5 is not strictly necessary, but it can be useful during the experiments.

The maze-like environment is described in the attached file `funky-maze.world` in `sdf` format. The world file contains the description and locations of a number of blocks that altogether define a maze-like scenario, which is shown in the figure. The "origin" is set in the upper-right corner, where the robot is (together with a can of coke!).

Make a folder `worlds` in the root of your catkin workspace, `catkin_ws`, and add the `funky-maze.world` file in there. You can start a Gazebo simulation with the given world scenario by executing:

```
$ export TURTLEBOT_GAZEBO_WORLD_FILE=~/catkin_ws/worlds/funky-maze.world;
> roslaunch turtlebot_gazebo turtlebot_world.launch
```

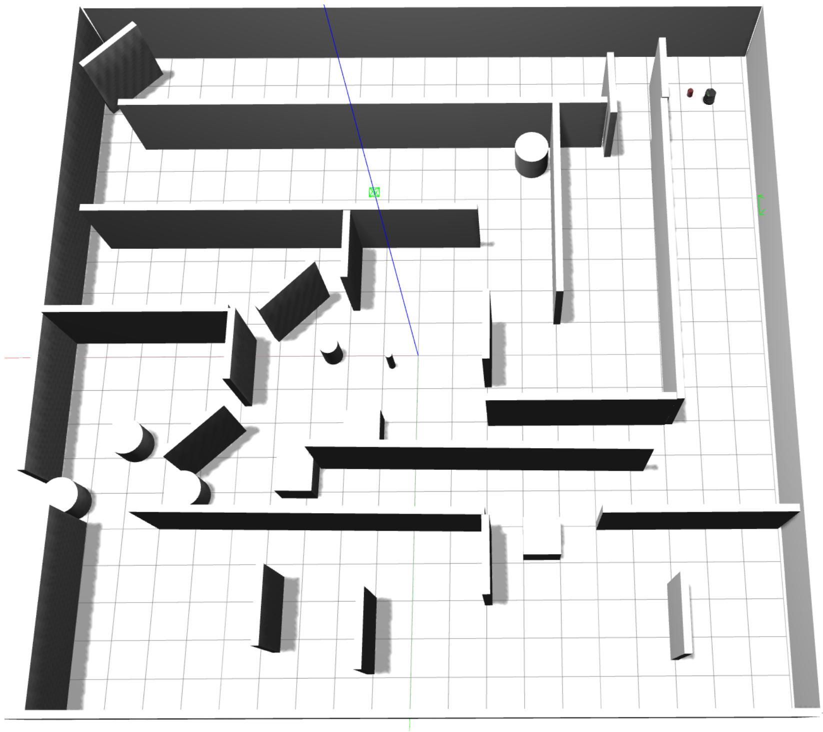

Figure 1: The maze-like scenario.

In addition, you can set the initial position of the robot as it is shown, with the orientation straight towards the open corridor (see the target path in the next figure). For this, you can use any of the methods described in the mentioned Section 5 above.

2 Task: Maze navigation (6 points)

Design a *reactive robot controller* (also defined as a *reflex agent*) to navigate through the maze. The robot has no knowledge about the structure of the maze (i.e., no "hardwired" ad-hoc behaviors are possible/admitted!). Robot's "reflex-based behavior" must consists of two basic sub-behaviors:

- *go straight* (i.e., keep current heading) when there are no obstacles in front,
- *make turns* to avoid to collide with obstacles that are on the way.

The goal of the robot is to keep making *tours*, that is, circulating "forward" in the maze, possibly forever. This is indeed possible given the structure of the maze. In Figure 2, an example of a feasible tour path (obtained with a reactive controller) is shown. If your robot more or less would move following a path similar to the one shown in the figure, then the robot can keep circulating in the maze forever. However, strong deviations from the indicated path could easily take the robot to move inside areas from where it can't (easily) get out.

Why are we calling the target robot controller a "reflex" or a "reactive controller"? Because decision-making must be based exclusively on the current sensory input + the current state of the robot (i.e., current velocity and/or behavior, such as "the robot was avoiding an obstacle to the left"). In a sense, robot's behavior must be an immediate response (reaction, reflex) to a sensory input, neither reasoning forward nor backward before taking a decision for the new `Twist` vector (linear and angular velocity).

3 Your work

- Create a new ROS package with two nodes (at least) to implement the reactive controller. One node, `velocity-controller` moves the turtlebot in the maze-like environment. In other words, it sets the velocity controls. The other node, `obstacle-detector`, subscribes to the laser scan and bumper sensors topics and uses this information to build an instantaneous obstacle map (i.e., if and where obstacles are in the FOV of the robot based on the currently received information from the sensors). Information

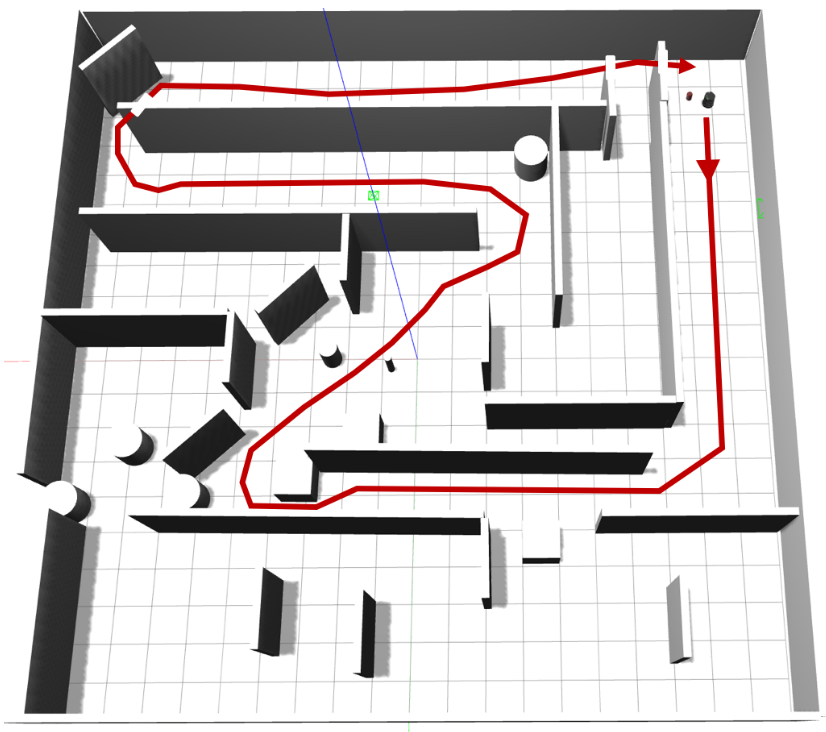

Figure 2: One feasible maze *touring* path.

about the presence of obstacles is published by `obstacle-detector` on topic `obstacle_detected`. The `velocity-controller` node is a subscriber of this topic, and it makes use of the received information to adapt velocity controls.

- Robot's navigation should be:
  - *safe*: no collisions with obstacles,
  - *effective*: the robot should be able to keep touring in the maze, similarly to what shown in Figure 2 (or even better!). Moreover, the faster and the smoother the motion, the better.
- How do you quantify that you have achieved the above two goals? Once you're ready with your controller, starting from the indicated pose, let your robot navigating the maze for 15 minutes (real time). Assuming that your robot is capable of "touring":
  - count how many times the robot is passing near the can of coke (i.e., passing by the origin);
  - count how many times the bumpers get triggered;
  - compute an estimate of your average speed;
  - Optional: can you compute a measure of smoothness of your motion?

  Report these numbers and discuss them:
  - Was the performance satisfactory? Justify and discuss the answer.
  - What are the issues (if any) of your controller?
  - Identify precise ways to proceed in order to improve the navigation performance of the reactive controller.

4 Designing a deliberative controllers (2 extra points)

Design a deliberative robot controller of your choice to navigate through the maze and discover every location on the map. In this case, your architecture should have sensing, localization, map generation, planning, and action. There will be no limitation about methodology of each part. Let your robot navigating the maze and report its performance.