EPD 3: Path tracking & collision avoidance

Objectives

- Modifying our robot controller by following a path instead of a single goal → path tracking
- Extending the controller to avoid collisions while following the path → collision avoidance

Assignments

1 Introduction

In the previous EPD we developed a simple goal follower. We also introduced the robot sensing through a 2D laser range finder to check possible collisions with obstacles.

In this EPD we wil improve our controller. We will transform our goal follower into a local controller capable of following a given path and avoiding collisions with obstacles in its way (see Figure 1).

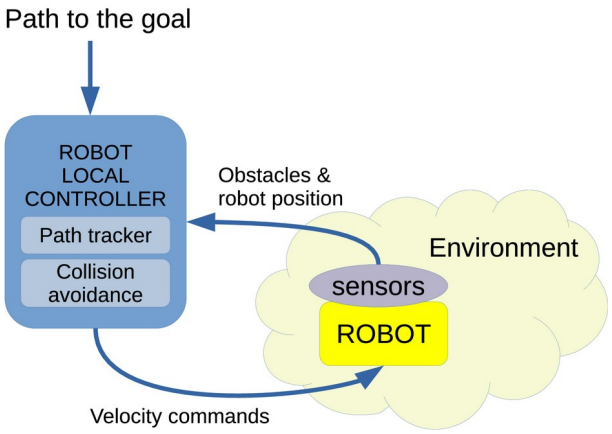

Figure 1. Diagram of robot navigation in EPD3

2 Changing the single goal by a (pre-defined) path to the goal.

In the EPD 2, we provided a single final goal to our Turtlebot robot. However, in most situations, as we will see from this EPD on, a robot local controller receives a continous path to the final goal. This path is approximated as a set of discrete points (equidistant usually) as can been in Figure 2.

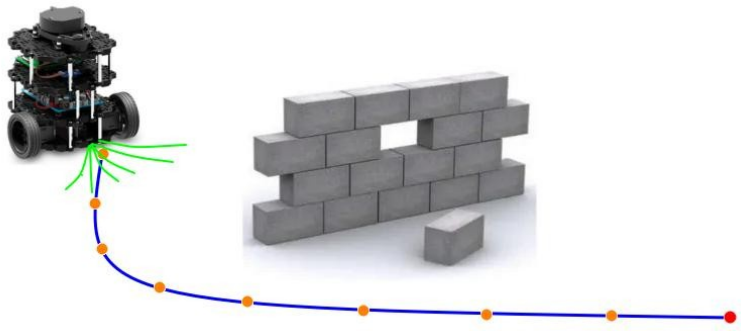

Figure 2. Representation of a path to a goal (red point).

We will modify the ROS Node developed in EPD2 to receive a Path message instead of single goal position. We will add a new topic subscription will receive messages of type nav_msgs/Path:

nav_msgs/Path Message

File: nav_msgs/Path.msg

Raw Message Definition

```
#An array of poses that represents a Path for a robot to follow
Header header
geometry_msgs/PoseStamped[] poses
```

Compact Message Definition

```
std_msgs/Header header
geometry_msgs/PoseStamped[] poses
```

autogenerated on Wed, 13 Jan 2021 03:28:12

geometry_msgs/PoseStamped Message

File: geometry_msgs/PoseStamped.msg

Raw Message Definition

```
# A Pose with reference coordinate frame and timestamp
Header header
Pose pose
```

Compact Message Definition

```
std_msgs/Header header
geometry_msgs/Pose pose
```

Figure 3. Path message interface

Path tracking

There are plenty of methods that can be used for path following. As a first approximation, the students are encouraged to calculate the closest point from the robot to the path (see point C in Figure 4). Then, the robot should be commanded to reach not that point but another point farther away (G). The distance between C and G is usually called lookahead (L). At each new iteration in our control module, the point to be reached is to be calculated again. This way, the direction of $v_{command}$ can be obtained.

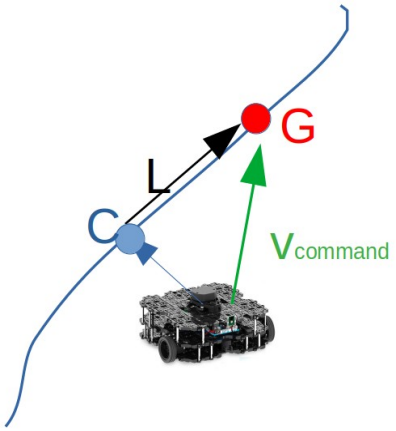

Figure 4. A simple strategy for path tracking

Provided code

Download the zip file “robot_controller.zip” from the EPD3 in Aula Virtual. This is a ROS package so you can unzip it in your ROS workspace. Then, from your container, compile the workspace:

```
$ cd /home/rva_container/rva_exchange/rva_ws
$ catkin_make
```

The remarkable files that the package contains are:

robot_controller:

- scripts:
  - path_publisher.py → node to publish in ROS a given path read from path.yaml.
  - robot_controller.py → node to control the robot. The student must fill and complete the code.
  - robot_utils.py → auxiliary functions to help in the computation of the robot kinematics.
- launch:
  - path_publisher.launch → launch file to run the node for path publishing.
  - turtlebot3_sim.launch → launch file to run the simulation of the Turtlebot3 for this EPD.
  - epd3.launch → general launch that includes the above launch files and run the robot_controller node.
  - Challenge0.launch → general launch to run the robot controller in the environment for exercise 2.
- config:
  - path.yaml → file with the coordinates of the path points.
  - challenge0_path.yaml → file with the coordinates of the path to be followed in exercise 2.
  - epd3.rviz → Rviz configuration file that will be used to run Rviz with the preconfigured visualization.

The content of these files will be briefly explained in class.

Generating new paths

In the following EPDs we will see how to compute a path to given goal. However, in this epd we will use a predefined path in which the point coordinates have been manually anotated in a yaml file (*robot_controller/config/path.yaml*).

You can use new paths by creating a new yaml files and using the Rviz tool "2D Nav Goal". Every time you set a navigation goal, you can watch the coordinates in your container terminal by echoing the topic "move_base_simple/goal"

```
$ rostopic echo /move_base_simple/goal
```

Exercise 1:

Extend the node robot_controller.py to follow the given path to the final goal.

Once you have modified your code, you can run the different nodes by launching the file epd3.launch:

```
$ roslaunch robot_controller epd3.launch
```

This will execute:

- the Gazebo simulation.
- The path_publisher node.
- Your robot_controller node.

3 Extend the robot controller to handle possible collisions (first part of the Robotics Challenge!)

In the EB classes we are presenting different techniques for reactive local planning (*T4 – control & collision avoidance*). Using these techniques we could endow our path trackers with skills to avoid certain obstacles. Try to extend your robot controller by implementing one of the reactive methods proposed in T4 to avoid collisions:

- Bug algorithm
- Potential fields
- Velocity Obstacles
- Dynamic Window Approach
- Any other idea that you can devise

To ease the implementation of some of them, we provide a set of auxiliary functions (*robot_utils.py*) that can be used by your controllers.

To test your controllers in this part we will use a different environment and a different path (see Figure 5). You can launch your navigation system with the proper environment and path by running:

```
$ roslaunch robot_controller challenge0.launch
```

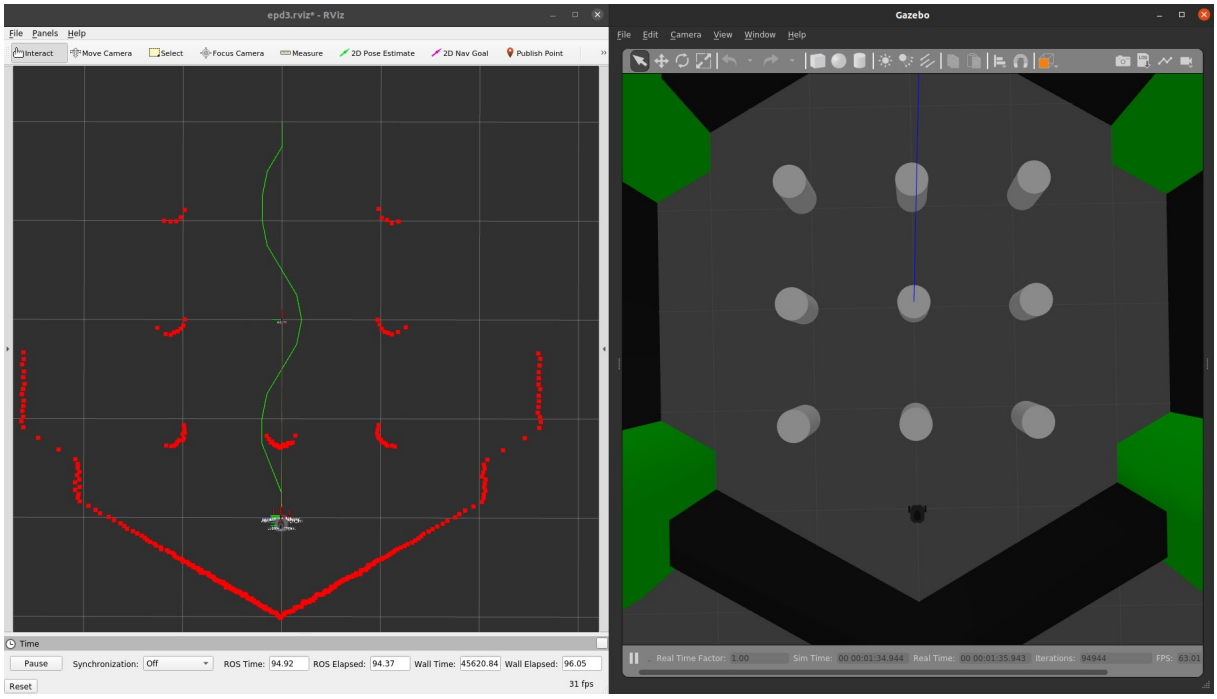

Figure 5. Capture of Rviz (left) and Gazebo (right) with the setup for testing the collision avoidance of the controllers.

Exercise 2:

Fill the function collisionAvoidance of your node robot_controller.py in order to command the robot to follow the given path while avoiding obstacle collisions. Feel free to add/modify the variables and functions that you may need.