Lab03 - 09/11/2023

Understanding ROS 2: Gazebo, Rviz 2, practical exercise

Objectives

Get familiar with common simulation and visualization software used in robotics.
Get familiar with Gazebo, a commonly used simulation environment in robotics.
Develop a simple Bump&Go controller to drive your robot through a path delimited by walls.

Tutorial

Gazebo and TurtleBot3 simulation

Gazebo is an open-source robot simulation software that provides a platform for simulating the dynamics of robots and their interactions with the environment, including sensors. It is commonly used in the field of robotics for various purposes, including robot design, development, testing, and validation. Gazebo offers a 3D simulation environment that allows users to create and simulate robots and environments in a highly realistic and customizable manner.

TurtleBot3 is a popular open-source, low-cost, and compact mobile robot platform designed for education, research, and prototyping in the field of robotics. This is the robot model you will use throughout this course for laboratory activities and for your final project. To get familiar with it, we will start by running it in simulation.

Download all the TurtleBot3 packages from GitHub into your src folder. Run this command (it is a single line) from the top folder of your workspace:

```
$ curl
https://raw.githubusercontent.com/SESASR-Course/turtlebot3/humble-devel/turtlebo
t3_sim.repos | vcs import src
```

You will find all the downloaded files inside `src/turtlebot3`, including `turtlebot3_sim.repos` where all the downloaded repositories are listed.
Now, install the needed dependencies and build the workspace:

```
$ sudo apt update
$ rosdep update
$ rosdep install --from-paths src --ignore-src -y
$ colcon build --symlink-install
```

Then, try running the simulation with the following command. **The first time you start the simulation it may take a long time because it has to download the models.** Be patient, and if you do not see the robot in the simulation, press CTRL + C in your terminal, then start again the simulation with the same command.

```
$ export TURTLEBOT3_MODEL=burger
$ ros2 launch turtlebot3_bringup gazebo.launch.py
```

A window similar to the one reported in Figure 1 should open. If you do not see it, try running the previous command adding `verbose:=true` at the end to debug your problem.

NB: the **TURTLEBOT3_MODEL variable is essential** for almost every turtlebot3 node. You must export it in every new terminal or in alternative you can put the statement inside your `~/.bashrc` file to automatically export the variable in every new terminal.

In a new terminal open the turtlebot3 teleop node to check if the simulation is working correctly.

```
$ export TURTLEBOT3_MODEL=burger
$ xterm -e ros2 run turtlebot3_teleop teleop_keyboard
```

If xterm is not installed run

```
$ sudo apt update && sudo apt install -y xterm
```

Close the previous Gazebo instance and try to open the world you will use for this laboratory activity:

```
$ ros2 launch turtlebot3_bringup lab03.launch.py
```

Hint: try to inspect the topics available and the nodes running!

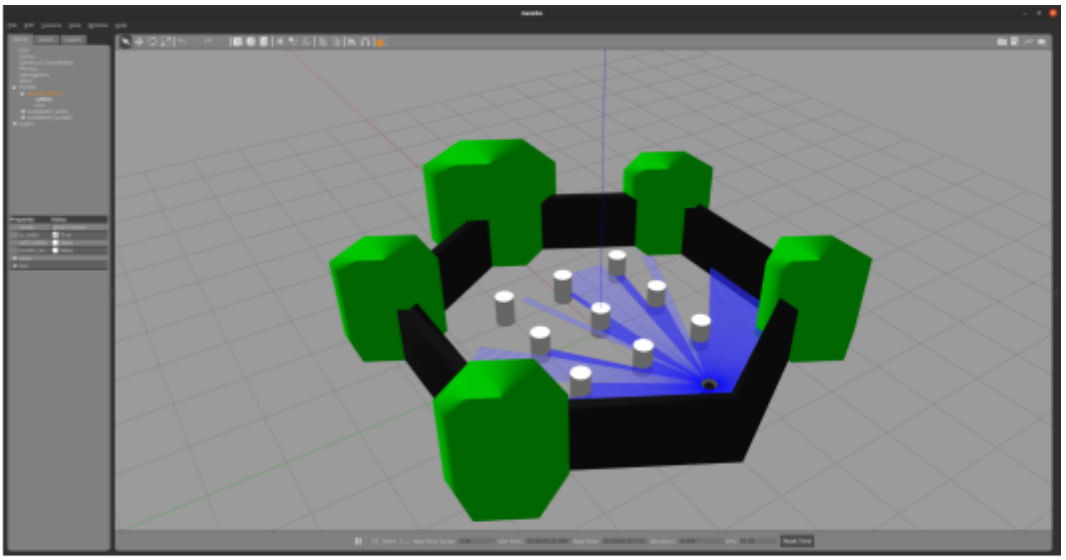

Figure 1: Gazebo simulation of TurtleBot3 in its default world

Rviz2

Rviz2 is a 3D visualization tool that can help you understand what is happening to your robot. It can visualize the tf tree and all the data types that are exchanged in ROS 2 like Odometry, LaserScan, PointCloud, Images and so on. To run it, type `rviz2` in a terminal.
To start Rviz2 together with the Gazebo simulation use the following command (the same option is also available in `lab03.launch.py`)

```
$ ros2 launch turtlebot3_bringup gazebo.launch.py start_rviz:=true
```

Now try to visualize the LiDAR topic. From *Add* > *By topic,* double click on LaserScan under */scan* topic. You should see a lot of small red dots surrounding the robot as shown in Figure 2.

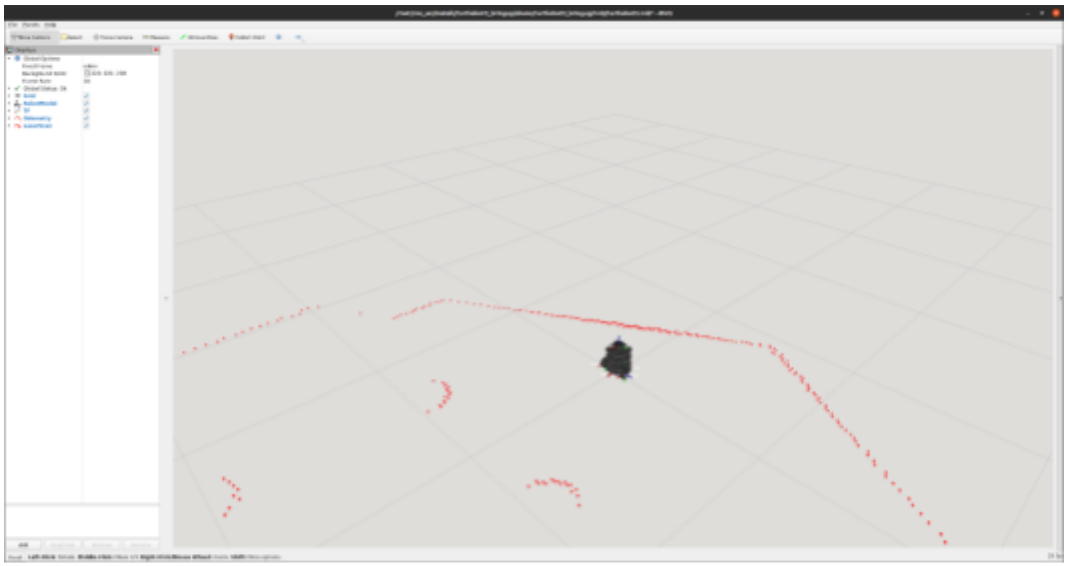

Figure 2: Rviz2 displaying the robot model, the TF frames and LiDAR points

For a detailed description of all the functionalities, you can refer to the official documentation. Please, keep in mind that the graphical aspect of some windows may have changed over the years.

LiDAR

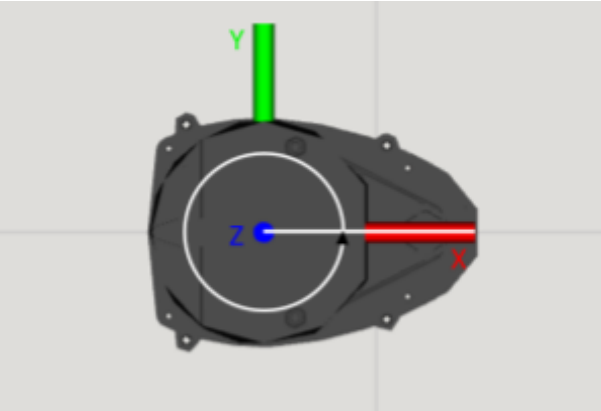

Figure 3. LiDAR reference frame

The LiDAR uses a laser beam to measure the distance to objects around it. Rotating continuously, it measures all the points surrounding the robot at a fixed angular step, depending on the rotational speed and on the measurement speed of the sensor.

The LiDAR equipped on TurtleBot3 has an angular resolution of 1° and does 5 complete rotations per second. The angle 0° is aligned with the X axis of the sensor and the angle increments with a positive rotation around the Z axis (right-hand rule). The maximum distance measured by this LiDAR is equal to 3.5 m.

The robot package publishes LiDAR range measures on a topic, typically called `/scan`. The message published is a `sensor_msgs/msg/LaserScan` and it is reported below. Ranges is an array of fixed length and contains, in order, the measured distances starting from *angle_min*, at fixed steps of *angle_increment*, until *angle_max*.

```
---
header:
  stamp:
    sec: 15
    nanosec: 578000000
  frame_id: base_footprint
angle_min: 0.0
angle_max: 6.28318977355957
angle_increment: 0.017501922324299812
time_increment: 0.0
scan_time: 0.0
range_min: 0.11999999731779099
range_max: 3.5
ranges: '<sequence type: float, length: 360>'
intensities: '<sequence type: float, length: 360>'
---
```

You can find a detailed description of each message field in the [ROS 2 official documentation].

TurtleBot3 simulation launch file

As you may have understood, the launch file we used until now is located inside the package `turtlebot3_bringup`, inside the `launch` folder, and it is called `gazebo.launch.py`. Open it inside VSCode and let us have a closer look at it.

If you search for **DeclareLaunchArgument** inside the file, you will find all the configurable arguments that can be set when calling the launch file from the command line. You have already used some of them in the previous steps, for example, *world* and *start_rviz*. They can be set after the launch file name using the syntax `<argument_name>:=<value>.`
Additionally, you can use the `--show-args` option when launching the file to display all the available arguments.
After all the declared arguments, there are a couple of **IncludeLaunchDescription** statements. This is a way to subdivide the launch files into different parts, each having a precise objective. In this case, *base.launch.py* starts all the nodes related to the robot description and control, while *gazebo.launch.py* starts Gazebo, setting all the environment variables and adding the plugins to communicate with ROS 2.

Report 1 Final Exercise

This exercise concludes the first laboratory module where you acquired the basic knowledge about ROS 2. Only functional requirements are described in this exercise and you must make a choice in the design of a solution.

Main Steps:

1. Use the `lab03.launch.py` (that you can find in the launch folder of the *turtlebot3_bringup* package *turtlebot3/turtlebot3_bringup/launch/*) to start the simulation of the Burger robot in a custom Gazebo world, already provided.
2. Create a package for this exercise named `lab03_pkg`. If you need additional interfaces for eventual custom services or action messages put them in a package named `lab03_interfaces`.
3. Design a simple controller (in a node) capable of guiding your TurtleBot3 through the path delimited by walls. The **starting pose** is **[x, y, yaw] = (0, 0, 0)**, then the robot should reach an **intermediate pose [x, y, yaw] = (6.5, 2.5, 90°)** with a tolerance of 1 meter and **finally,** it should return to its starting position **[x, y, yaw] = (0, 0, 180°)**
4. Register, plot, and evaluate the resulting performance of your controller.

Figure 4. Flowchart of the control algorithm

Logic: The working principle of the controller should be the **Bump&Go**. This is a really simple controller that can choose between two actions: *go forward* and *rotate* on itself. The default action of the robot is to go forward, if it finds an obstacle, then it rotates until it finds a free way to proceed.

Consider in your design that the TurtleBot3 Burger robot has a translational velocity limit of 0.22 m/s and a maximum angular velocity of 2.84 rad/s (162.72 deg/s), although not exceeding 1.5 rad/s is suggested for this task.

Hint (Go): To detect the obstacles you are required to use the LiDAR already implemented in the TurtleBot3 simulation package (topic `/scan`, message `sensor_msgs/msg/LaserScan`). Please keep in mind that the measurements are affected by Gaussian noise and that measures over the maximum allowed distance may assume infinite values.

Hint (Bump): usually the pose of a robot is published on an `/odom` topic and the orientation is expressed using a **quaternion**. It could be convenient to convert it to a **yaw angle** inside your code, when reading the message from the topic. Use the pre-build function from tf_transormations library:

```
Install the package from the terminal
$ sudo apt install ros-humble-tf-transformations
$ sudo pip3 install transforms3d

In your code
import tf_transformations

# where you need to perform the conversion
quat = [quaternion.x, quaternion.y, quaternion.z, quaternion.w]
_, _, yaw = tf_transformations.euler_from_quaternion(quat)
```

Hint: beside the odometry data published on a `/odom` topic by the differential drive controller of the TurtleBot3, you will find an additional `/ground_truth` topic containing an exact pose information of the robot in Gazebo. You can use this data to check the positioning error accumulated by the differential drive odometry system, which influences the performance of your control algorithm! Compare the results of your algorithm using the two poses topics.

Requirements:

- Create a package for this exercise named `lab03_pkg`. If you need additional interfaces put them in a package named `lab03_interfaces`
- In order to have a better configurability of your node, insert parameters for maximum velocity (linear and angular) and for the control loop frequency. If you deem it necessary, add also other parameters.
- Define a suitable method and threshold to determine whether you are close to a wall or you can continue going forward.
- Define a suitable method to choose whether to turn right or left.
- Can you find a suitable condition to stop the robot if it cannot perform any of the possible actions (eg. walls in front, left and right)?

Hint: to prepare the plots after you run the simulation you can record a rosbag during the simulation and then open it with PlotJuggler. An example command to record the needed topics for this lab is `ros2 bag record -e '(odom)|(cmd)|(truth)'`. To load the recorded data with PlotJuggler click the icon near Data in the top left corner, then select the `metadata.yaml` of your recorded bag.

Report

- Describe in a clear way the algorithms you implemented to solve the proposed problem
- Were you able to reach the end of the path? If not, why?
- XY plot of the trajectory reported in the odometry topic vs. the ground truth
- Plots of each coordinate (x, y, yaw) compared to ground truth
- Plot of the velocity commands
- Plot of the absolute error of position (x, y) and orientation

Comment on the obtained results and the problems you faced.

You should deliver the report on the Portale della Didattica with the name:

`sesasr_report01_<group_number>.pdf`

You should also deliver on the Portale della Didattica a zipped folder containing the package(s) that you developed, named:

`sesars_lab<lab_number>_<group_number>.zip`

Deadline December 1st (the report must include relevant required results and comments from the first 3 labs activity).