Lab02

Simulation in Gazebo and visualization with Rviz2

Objectives

- Understand **launch** and **parameters** file in ROS 2
- Get familiar with common simulation software used in robotics: **Gazebo**
- Get familiar with common visualization software used in robotics: **Rviz 2**
- Develop a simple **Bump&Go controller** to drive your robot through a path delimited by walls

Requirements

Parameters and Launch files

Parameters

In ROS 2, **parameters** are **key-value pairs** used to **configure** nodes at **runtime** without the need to modify the code. Parameters allow nodes to be **more flexible** and adaptable to different operating conditions or scenarios by controlling various settings, such as thresholds, file paths, sensor configurations, or algorithm settings.

Parameters must be declared inside the code and their value can be retrieved depending on its value type. Allowed parameters types are `integer`, `double`, `bool`, `string` and `arrays` of the previous types. An example on how to declare a couple of parameters is reported below.

```python
import rclpy
import rclpy.node

class MinimalParam(rclpy.node.Node):
  def __init__(self):
    super().__init__('minimal_param_node')

    self.declare_parameter('param1', 'world')
    param1 = self.get_parameter('param1').get_parameter_value().string_value

    self.declare_parameter('param2', '2.0')
    param1 = self.get_parameter('param2').get_parameter_value().double_value
```

Parameters can be set when running a node with the following syntax:

```
ros2 run my_package my_node --ros-args -p param1:=world -p param2:=3.5
```

Launch files

A ROS 2 **launch file** is a configuration file used to start **multiple nodes** and **configure** their **parameters** in a single, organized way. It simplifies the process of running a complete robotic system, which often involves launching various nodes, setting parameters, and configuring other aspects like remapping topics or setting environment variables.

ROS 2 launch files can be written in Python, XML or YAML. Actually, the preferred language is Python since one can take advantage of Python's scripting capabilities, which offer greater flexibility and readability compared to the XML or YAML.

Configuration files

Many times, especially when using launch files, you will find nodes' parameters specified in configuration files. **Configuration files** are written in **YAML** format and they are a dictionary of key-value pairs. They are extremely useful to summarize the value of many parameters of multiple nodes. For example:

```yaml
# All characters after '#' are comments
# Pay attention to the file indentation. It is important as in Python
scripts.
robot_driver_node: # node name
  ros__parameters: # mandatory after the node name
    # dictionary of key-value pairs
    speed: 2.0
    max_turn_rate: 1.5
    robot_name: "Turtlebot"
    sensor_ids: [1, 2, 3]
    is_active: true
```

Gazebo

NOTE: if you use a Mac check Appendix B: Simulation with Gazebo Sim.

Gazebo is a powerful, open-source **simulation tool** used to model, simulate, and test robots in complex, realistic environments. It is widely used in robotics research, development, and education to simulate both the physical interactions of robots and their sensors before deploying them in real-world environments. Gazebo integrates well with ROS (Robot Operating System), particularly through plugins and interfaces, making it a popular choice for simulating robots within the ROS ecosystem.

ACTION: To **install** Gazebo along with its ROS 2 integrations run the following command in a terminal:

```
sudo apt update
sudo apt install ros-humble-gazebo-ros
```

Then, **open** the file **`~/.bashrc`** and **paste** the following line **before** the command to source ROS 2.

```
source /usr/share/gazebo/setup.bash
```

To start a Gazebo simulation in an empty world run the **`gazebo`** command in a terminal or start it from the Ubuntu Apps menu.

Rviz

RViz 2 is a 3D **visualization tool** for ROS 2 that allows users to visualize data from a robot in real-time. It is used to display sensor data (e.g., laser scans, camera feeds), robot models, transforms (TF), navigation data, and other information to help developers and operators better understand how the robot is interacting with its environment.

ACTION: To **install** Rviz 2 run the following command in a terminal:

```
sudo apt update
sudo apt install ros-humble-rviz2
```

Then to start the program run the command **`rviz2`** in a terminal with ROS 2 sourced.

Turtlebot 3 simulation

NOTE: if you use a Mac check Appendix B: Simulation with Gazebo Sim.

To do the exercise at the end of this laboratory you need a working simulation environment for the Turtlebot3. In the next two sections you will learn how to compile the simulation package and how to run the simulation. Moreover, you will also learn how to visualize the state of your robot with Rviz 2.

Download turtlebot3_simulation package

ACTION: **open** a terminal, **change directory** to be inside your `src` folder of the ROS 2 workspace, run the following commands to **download** the `turtlebot3_simulations` package, **install** the dependencies and **build** the workspace. **Only the command after the $ symbol must be executed in the terminal, the text before the $ symbol represents the current working directory.**

1. Clone the package

```
~/ros_ws/src$ git clone https://github.com/SESASR-Course/turtlebot3_simulations.git
```

2. Move to base folder of the workspace, install dependencies and build the workspace

```
~/ros_ws/src$ cd .. # change directory to the base folder of workspace
~/ros_ws$ rosdep install --from-path src --ignore-src -y -r
~/ros_ws$ colcon build --symlink-install
```

Run the Turtlebot3 simulation and visualize robot and sensors

To open the simulation that you need to complete the final exercise, you have to follow three steps. In the following example it is assumed that the current working directory is your ROS 2 workspace base folder. **Only the command after the $ symbol must be executed in the terminal, the text before the $ symbol represents the current working directory.**

1. Source your local workspace

```
~/ros_ws$ source ./install/setup.bash
```

2. Set the Turtlebot 3 model

```
~/ros_ws$ export TURTLEBOT3_MODEL=burger
```

3. Run the simulation

```
~/ros_ws$ ros2 launch turtlebot3_gazebo lab02.launch.py
```

HINT: To avoid running the **first two commands** every time you need to start the simulation, you can put these commands in your `~/.bashrc` **after** the command to source ROS 2.

You can visualize the robot and its sensors using the following launch file:

```
$ ros2 launch turtlebot3_bringup rviz2.launch.py
```

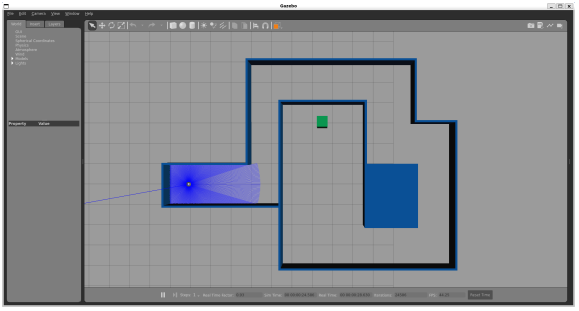

Figure1. Gazebo simulation

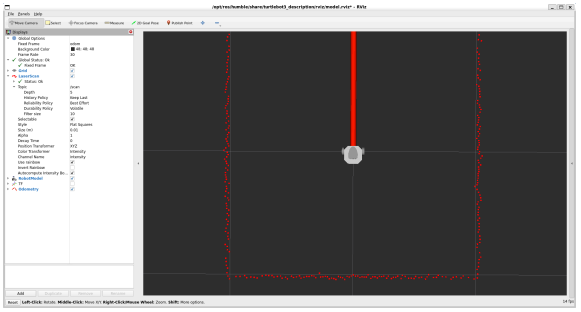

Figure2. Rviz2 with TB3 and LiDAR points

Exercise

Design **a Bump&Go** controller to guide the TurtleBot3 through a path delimited by walls, to reach the green cube at the end. Bump&Go 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.

Main Steps:

1. Use the **`lab02.launch.py`** (that you can find in the launch folder of the **`turtlebot3_gazebo`**) to start the simulation of the TB3 Burger robot in a custom Gazebo world.
2. Create a package for this exercise named **`lab02_pkg`**
3. Design a simple controller (in a node) capable of guiding your TB3 through the path delimited by walls to reach the green cube. The **starting pose** is **[x, y, yaw] = (0, 0, 0)**, the **final goal** is the green cube.
    a. Insert parameters for maximum velocity (linear and angular) and for the control loop frequency.
    b. Define a suitable method and threshold to determine whether you are close to a wall or you can continue going forward.
    c. Define a suitable method to choose whether to turn right or left.
4. Register, plot, and evaluate the resulting performance of your controller.

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 a maximum of 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 of type **`nav_msgs/msg/Odometry`** 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-built function from tf_transormations library:

Figure 4. Flowchart of the control algorithm

Install the package from the terminal:

```
$ sudo apt install ros-humble-tf-transformations
$ pip install --upgrade 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 could influence the performance of your control algorithm! Compare the results of your algorithm using the two poses topics.

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 -a`**. 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.

Appendix A: LiDAR

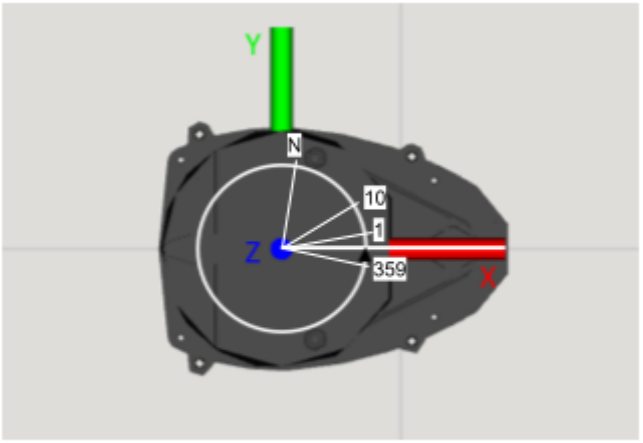

Figure 3. LiDAR reference frame

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.

Appendix B: Simulation with Gazebo Fortress

In newest Apple computers, the M1 and M2 processor architecture (based on ARM64) does not allow the execution of the old Gazebo version, called “classic”. If you are not sure about the processor of your Mac you can find a list of models using M1 here and a list of models using M2 here.

If your Mac is in one of the lists linked before you must use the newer Gazebo Fortress, also called Ignition Fortress, to run the simulations. Follow the steps reported below, then carry out the proposed Exercise.

ACTION: Follow the instructions reported on the official guide to install the simulator.

Download turtlebot3_simulation package

ACTION: **open** a terminal, **change directory** to be inside your **`src`** folder of the ROS 2 workspace, run the following commands to **download** the **`turtlebot3_simulations`** package, **install** the dependencies and **build** the workspace. **Only the command after the $ symbol must be executed in the terminal, the text before the $ symbol represents the current working directory.**

1. Clone the package

```
~/ros_ws/src$ git clone https://github.com/SESASR-Course/turtlebot3_simulations.git
```

2. Switch the simulation packages

```
~/ros_ws/src$ rm turtlebot3_simulations/turtlebot3_ignition/COLCON_IGNORE
~/ros_ws/src$ touch turtlebot3_simulations/turtlebot3_gazebo/COLCON_IGNORE
```

3. Move to base folder of the workspace, install dependencies and build the workspace

```
~/ros_ws/src$ cd .. # change directory to the base folder of workspace
~/ros_ws$ rosdep install --from-path src --ignore-src -y -r
~/ros_ws$ colcon build --symlink-install
```

Run the Turtlebot3 simulation and visualize robot and sensors

To open the simulation that you need to complete the final exercise, you have to follow three steps. In the following example it is assumed that the current working directory is your ROS 2 workspace base folder. **Only the command after the $ symbol must be executed in the terminal, the text before the $ symbol represents the current working directory.**

1. Source your local workspace

```
~/ros_ws$ source ./install/setup.bash
```

2. Set the Turtlebot 3 model

```
~/ros_ws$ export TURTLEBOT3_MODEL=burger
```

3. Run the simulation

```
~/ros_ws$ ros2 launch turtlebot3_ignition lab02.launch.py
```

HINT: To avoid running the **first two commands** every time you need to start the simulation, you can put these commands in your `~/.bashrc` **after** the command to source ROS 2.

You can visualize the robot and its sensors using the following launch file:

```
$ ros2 launch turtlebot3_bringup rviz2.launch.py
```

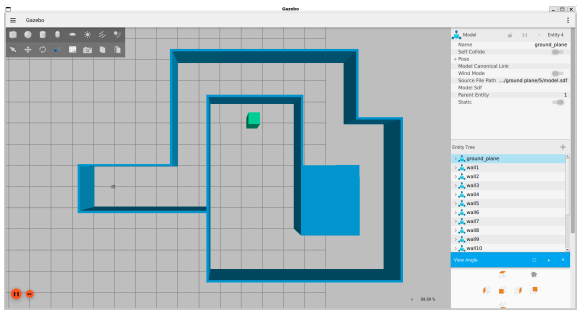

Figure1. Ignition simulation

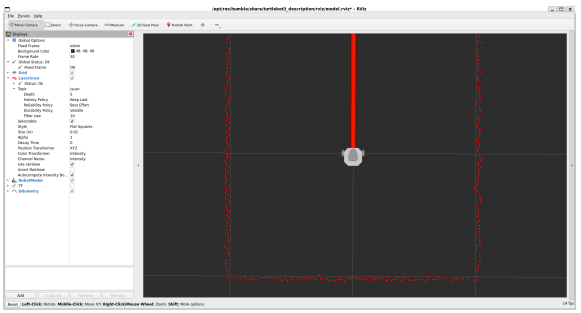

Figure2. Rviz2 with TB3 and LiDAR points