University of Trento

Industrial Engineering Department

Deambulator Control and Visualization with Ros2 and Gazebo

Mobile Robotics Project

Daniele Turrini (249485), Paolo Golinelli (247450)

Period of development: 18/10/2024 - 20/12/2024

1 Introduction

This report discuss the development and implementation of a mobile robotics simulation project conducted during the course Robotic Perception and Action. The primary objective of this project was to design and create a simulation environment using ROS2 and Gazebo. The simulation consists in driving a deambulator to follow a given path and visualize its environment.

A key requirement of the project was to implement the functionality to dynamically update both the path and the map, ensuring adaptability and real-time responsiveness. This capability is crucial for integrating the simulation with two other student projects: one that generates a height map using images of the environment taken by a depth camera, and another that generates an optimal robot path using the potential fields algorithm.

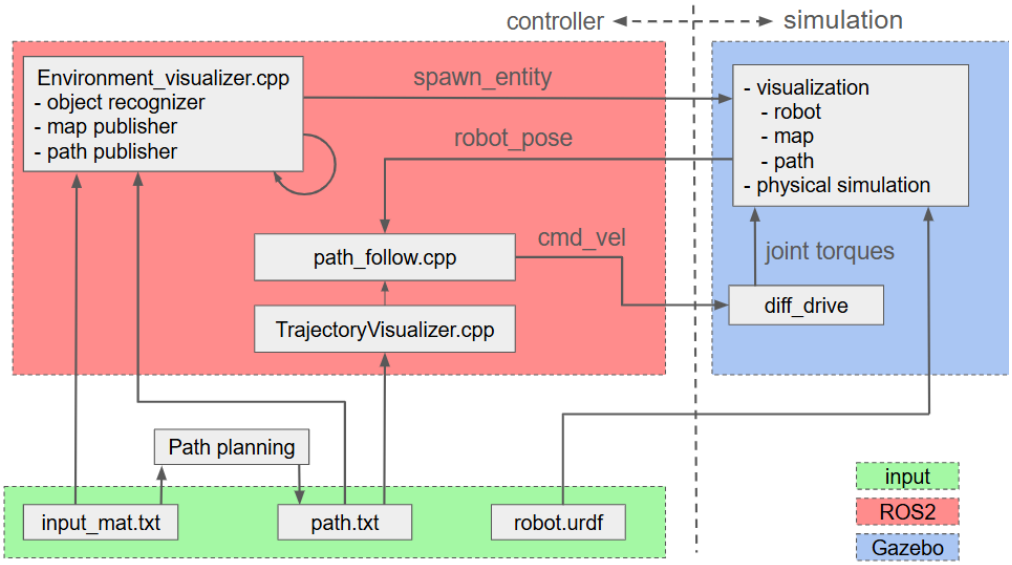

Figure (1) General Scheme of the whole simulation

A schematic representation of this project and its subdivision is provided in Fig. 1. Tree main parts are individuated: the input (in green) files, the ROS2 nodes working in collaboration and exchanging messages in and out of ROS2, and the physical simulator Gazebo.

2 Map and Path Visualization

The simulation environment incorporates both the predefined path and obstacles within the scene. These elements are defined through input provided in *.txt* files, which are appropriately parsed to extract relevant data by a ROS2 node called "object recognizer". The extracted information is subsequently used to update the visualization in Gazebo with the corresponding configurations. A schematic representation of this process is shown in Fig. 2.

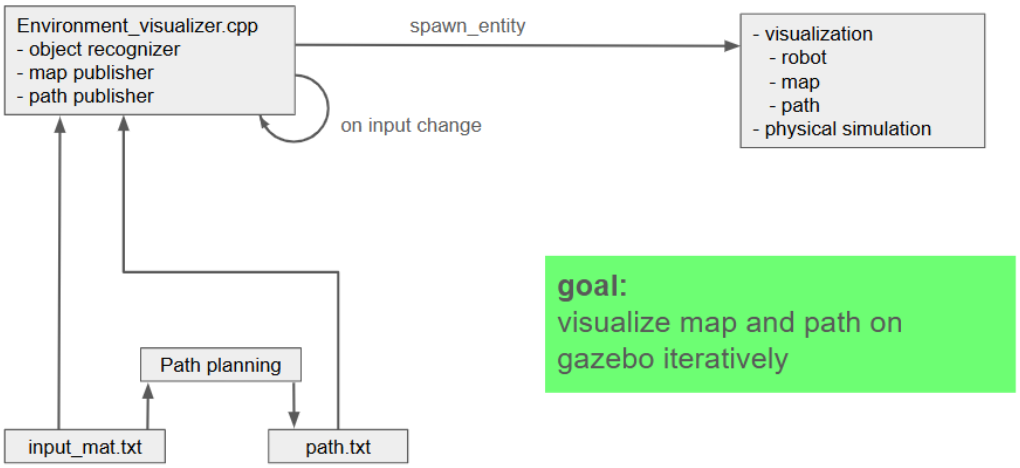

Figure (2) Scheme for ROS2 map and path visualization nodes

2.1 Map Parsing

The `input_map.txt` is a matrix of the xy environment plane, so it contains the heights of the obstacles. Since Gazebo is a physical simulator it is important to keep the number of objects as small as possible since for each object it calculates the contact forces and simulate its motion. To reduce the number of objects the map is passed to an algorithm which extracts rectangular features of same height into one bigger object (Fig.3).

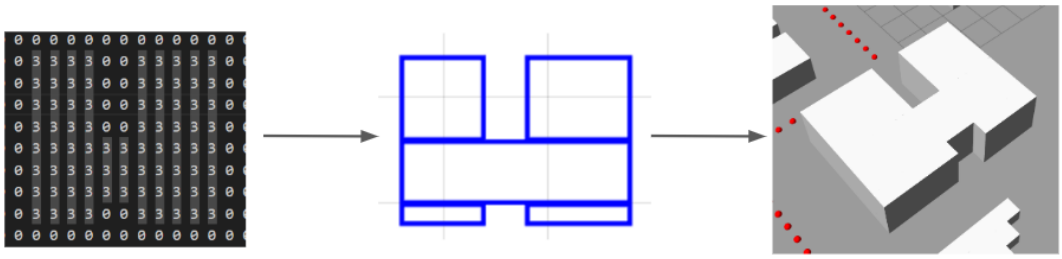

Figure (3) Algorithm extract rectangles of same height to ease the computation of the simulation in gazebo.

The obstacles are published by a ROS2 node in Gazebo for visualization, with communication handled via the MQTT protocol. Publishing all rectangles takes a few seconds, so updating the map visualization is done only when necessary. A change in the `input_map.txt` file triggers the process: the previous map is de-spawned, the new map is parsed, rectangles are extracted, the new path is planned and visualized, and the updated map is published in Gazebo.

2.2 Path Parsing

The path, generated from the Path Planning project, is provided as a *.txt* file, with each line specifying the coordinates of individual points. These points are processed by a file named `Environment_visualizer.cpp`, which performs two key functions. First, it generates a marker for each point in the Gazebo simulation, to have a visual representation of the path. Second, it publishes the trajectory to the ROS2 controller, enabling accurate navigation along the defined path. The scheme is shown in Fig.4.

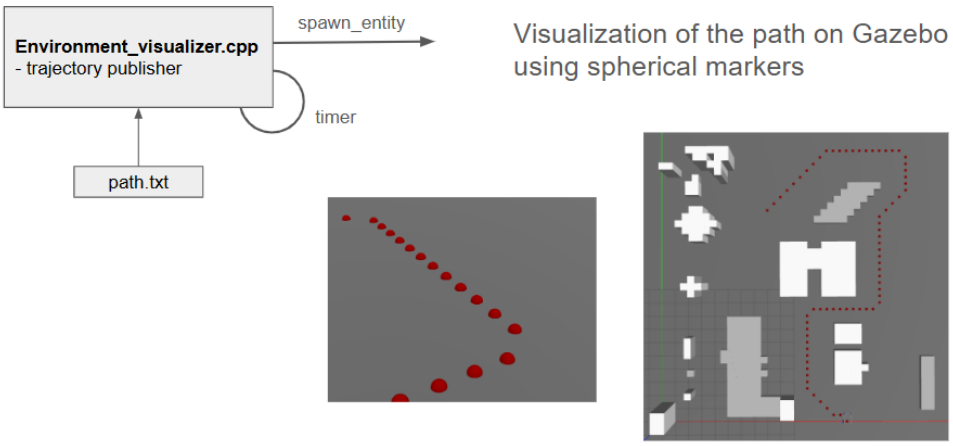

Figure (4) Scheme for ROS2 path parsing and publishing nodes

As previously discussed the process for visualizing the path in Gazebo is executed each time the input matrix is modified, since the path planning algorithm computes the new optimal path. The control node parses the input path at each iteration, so it is updated as soon as the path planning algorithm is executed.

3 Robot control

The robot control system takes the published trajectory and the current pose of the deambulator as inputs. Based on this information, it computes the velocity commands for each wheel to ensure accurate path following. Those velocities are then translated in torques by the ROS2 control node `diff_drive`. The control algorithm employed is the *Stanley* method, which is well-suited for path tracking tasks. The entire algorithm is implemented within the `pat_follow.cpp` ROS2 node. The scheme is shown in Fig.5.

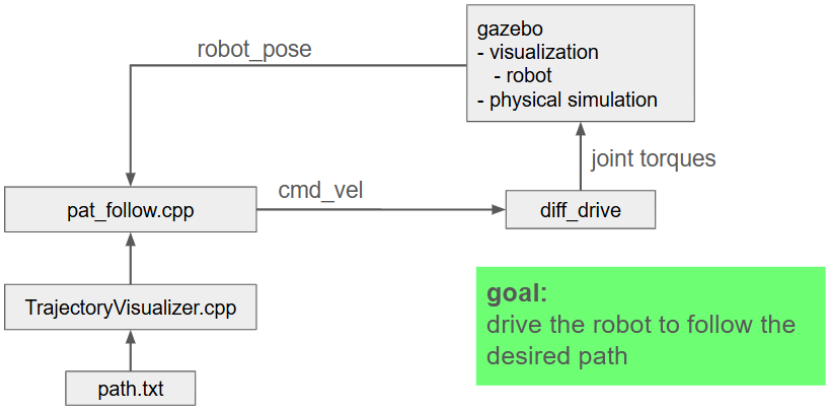

Figure (5) Control Scheme

3.1 Stanley Heuristic Method

Given a reference trajectory and the current pose of the robot (position and orientation), the Stanley method compute the linear and angular velocities to apply on the robot, those are then translated in wheels velocities by a ROS2 node. The algorithm adjusts the heading of the deambulator by considering two main factors:

- **Cross-Track Error Correction**: Aligning the robot's position with the reference trajectory by steering towards the closest point on the path.
- **Heading Error Correction**: Minimizing the angular difference between the robot's orientation and the trajectory direction at the target point.

The ROS2 node, implemented in `path_follow.cpp`, computes these velocities and publishes them to the robot's motor controllers, enabling precise path tracking. The Stanley formula to define the steering angular velocity is:

$$\omega(t) = k_p \left(\theta_e(t) + \arctan \left(\frac{k_e \cdot e_{fs}(t)}{v(t)} \right) \right) \tag{1}$$

Where $\theta_e(t)$ is the heading error, e_{fs} is the distance to the closest point in the path, $v(t)$ is the forward velocity and k_p, k_e are two proportional gains.

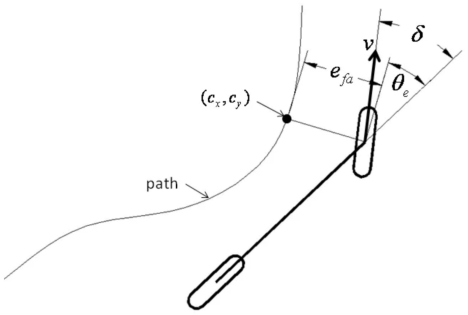

Figure (6) Example of quantities and angles used in Stanley

4 Deambulator URDF

The geometry of the deambulator is defined in the URDF (Universal Robot Descriptor Format) file. A URDF file is an XML-based structure used in ROS2 to describe the robot's physical and visual properties.

It defines the robot's links which are the robot's rigid components. The description includes the geometry for visualization, collision, and the inertial properties necessary for the simulation (mass and inertia).
Then contains the description of the joints, which specify how the links are connect and move relative to each other, their limits and if they have friction; joints can be fixed, revolute, prismatic and continuous. The positions of links and joints is described by roto-transformations between the various reference frames.
Finally it include the velocity-to-torque translator, for example the differential drive converts the forward velocity (along x axis) and the angular velocity (z axis) into torques to be applied to the traction wheels of the robot.

The figure Fig.7 contains the scheme followed in writing the URDF for the deambulator and a picture of the robot visualization.

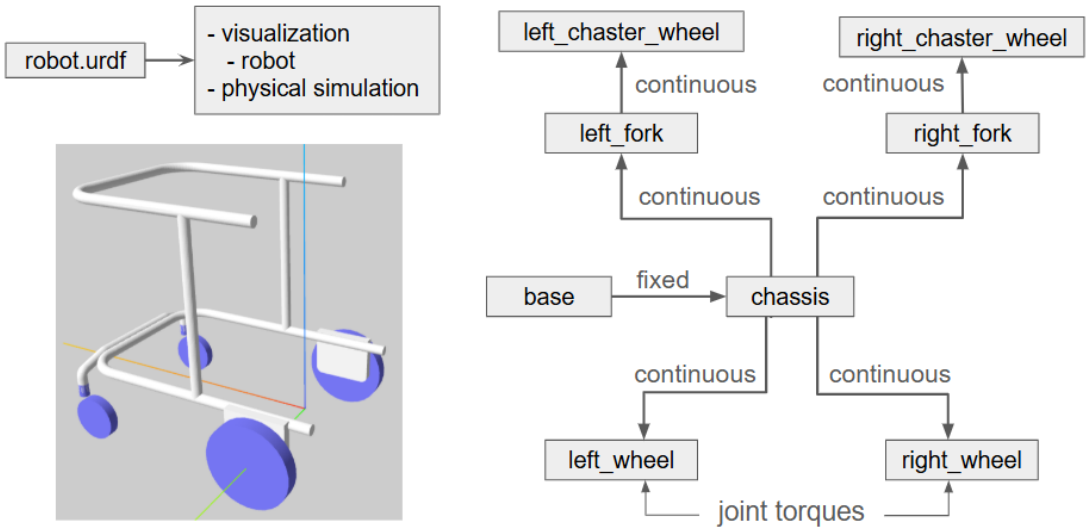

Figure (7) Deambulator's URDF scheme