

[CS-475] Assignment 4: Mapping with Known Poses

Changda Tian
cdtian@csd.uoc.gr

Release date: 09/04/2025
Deadline: 29/04/2025

1 Overview

In this assignment you are tasked with designing and implementing an occupancy grid mapping algorithm for the Turtlebot using the Gazebo simulator. In previous assignments, you have tackled state estimation and localization using either a GPS module or a pre-provided map. Now the problem is to build a map of a finite workspace using the robot's known poses (assumed to be highly accurate) and noisy lidar measurements.

The Turtlebot is equipped with a 2D lidar sensor that provides range measurements z_t at time t . These measurements (referred to as *pass-through/hit* information) are used to decide whether the grid cells in the robot's environment are free or occupied. In this assignment, you will first implement a simple binary mapping approach and then refine your results using a log-odds (probabilistic) update method based on techniques from the book *Probabilistic Robotics* (see Tables 9.1 and 9.2).

2 Theoretical Background

2.1 Occupancy Grid Mapping Fundamentals

Occupancy grid mapping is a technique in which the environment is discretized into a grid of cells, each associated with a probability of being occupied. Given sensor measurements and known robot poses, the mapping problem is formulated using Bayesian probability. For a grid cell m , we wish to estimate:

$$p(m \mid z_{1:t}, x_{1:t}),$$

where $z_{1:t}$ and $x_{1:t}$ denote the history of sensor readings and robot poses, respectively.

2.2 Binary Occupancy Grid Mapping

The binary mapping approach is a simplified method where each grid cell is updated with hard decisions:

- **Free Space:** Cells along a laser beam that are closer than the measured range are set as free (e.g., a value of 0).
- **Occupied Space:** The cell corresponding to the endpoint of a laser beam (where an obstacle is detected) is set as occupied (e.g., a value of 100).

To implement the binary method, the inverse sensor model is used in a simplified manner. For each cell, a precomputed log-odds increment is applied:

$$\Delta l = \begin{cases} 10 \times \log \frac{p_{\text{free}}}{1-p_{\text{free}}} & \text{if the cell is free,} \\ 10 \times \log \frac{p_{\text{occ}}}{1-p_{\text{occ}}} & \text{if the cell is occupied.} \end{cases}$$

After updating, the cell values are clamped to the range $[0, 100]$ to yield a clear binary interpretation.

2.3 Log-Odds (Probabilistic) Occupancy Grid Mapping

While the binary method makes hard decisions, the log-odds method provides a more refined probabilistic update. Instead of directly assigning binary states, each cell is maintained with a log-odds value:

$$l_t^{(m)} = \log \frac{p(m \mid z_{1:t}, x_{1:t})}{1 - p(m \mid z_{1:t}, x_{1:t})}.$$

This formulation allows an additive update:

$$l_t^{(m)} = l_{t-1}^{(m)} + l(m \mid z_t, x_t) - l_0,$$

where:

- $l(m \mid z_t, x_t)$ is the inverse sensor model output in log-odds form.
- $l_0 = \log \frac{p(m)}{1-p(m)}$ is the prior (with $l_0 = 0$ for a 50% initial probability).

To visualize the map, the log-odds value is converted back to a probability:

$$p(m \mid z_{1:t}, x_{1:t}) = \frac{1}{1 + \exp(-l_t^{(m)})},$$

which is then scaled to a 0–100 range.

3 Simulation Setup

3.1 ROS Package and Environment

You are provided with a ROS package named `assign4/`. To set up the simulation:

1. Copy the package into your ROS workspace.
2. Compile the workspace.
3. Launch the simulation using the provided launch file:

```
$ roslaunch assign4 burger.launch
```

4. To control the Turtlebot manually, use:

```
$ roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch
```

In the simulation, the Gazebo environment represents a finite workspace (a 6x6 m room), and Rviz is used to visualize your occupancy grid map.

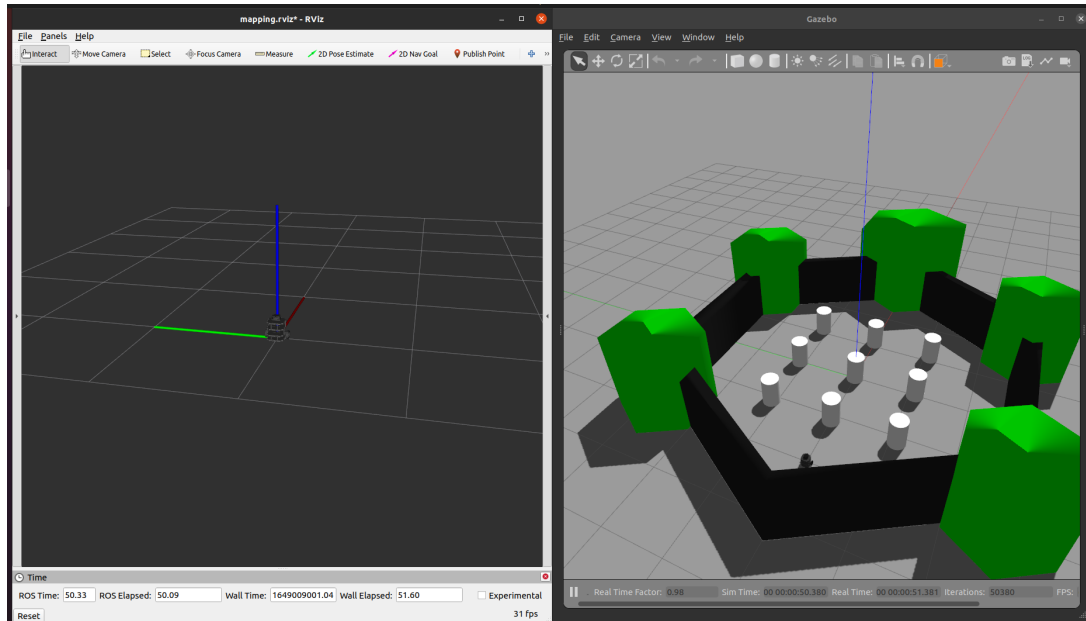


Figure 1: Gazebo + Rviz

3.2 Topics and Data Structures

The node `mapping.py` will subscribe to and publish the following topics:

- `/scan` → `sensor_msgs/LaserScan` (lidar readings)
- `/odom` → `nav_msgs/Odometry` (robot pose)
- `/map` → `nav_msgs/OccupancyGrid` (the occupancy grid to be visualized)

The `OccupancyGrid` message has the following fields:

```
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
nav_msgs/MapMetaData info
  time map_load_time
  float32 resolution
  uint32 width
  uint32 height
geometry_msgs/Pose origin
  geometry_msgs/Point position
    float64 x
    float64 y
    float64 z
  geometry_msgs/Quaternion orientation
    float64 x
    float64 y
    float64 z
    float64 w
int8 [] data
```

It is recommended to set the map origin to $(-3, -3, 0)$ so that the map aligns well with the workspace.

4 Implementation Details

4.1 Initialization

- **Map Parameters:** Set the resolution (e.g., 0.01 m) and the map dimensions (6 m by 6 m).
- **Map Initialization:**
 - For binary mapping, initialize the occupancy grid with an intermediate value (e.g., 50 on a scale from 0 to 100) to denote unknown occupancy.
 - For log-odds mapping, initialize an internal map with a neutral value (0), corresponding to a 50% probability.
- **Message Setup:** Create and configure the `nav_msgs/OccupancyGrid` message with the correct `frame_id` (e.g., "odom"), resolution, width, height, and origin.

4.2 Coordinate Transformation

The robot's pose is given with respect to the world frame. To update the occupancy grid, translate these coordinates into map indices. If the map origin is (x_0, y_0) and the resolution is r , then the cell indices (i, j) corresponding to a world point (x, y) are calculated as:

$$i = \left\lfloor \frac{x - x_0}{r} \right\rfloor, \quad j = \left\lfloor \frac{y - y_0}{r} \right\rfloor.$$

This transformation is essential for associating sensor measurements with the correct cell in the occupancy grid.

4.3 Laser Scan Processing and Map Update

The mapping update is performed using two approaches:

1. Binary Update:

- Traverse each cell in the grid.
- Convert the cell index (i, j) to its world coordinate (x_a, y_a) .
- Compute the Euclidean distance from the robot to the cell and the relative bearing.
- Identify the corresponding laser scan measurement based on the bearing.
- If the cell lies before the detected obstacle, update it as free; if the cell is at the obstacle, update it as occupied.
- Clamp cell values to the range $[0, 100]$.

2. Log-Odds Update:

- Process the laser scan data (extracting `angle_min`, `angle_max`, and `ranges`).

- For each valid laser beam, update cells along the beam:
 - Increment the log-odds of cells along the beam (free space) by adding a negative value.
 - Increment the log-odds of the cell at the measured obstacle by adding a positive value.
- Convert the log-odds values to probabilities using:

$$p(m \mid z_{1:t}, x_{1:t}) = \frac{1}{1 + \exp\left(-l_t^{(m)}\right)},$$

and scale them to the 0–100 range for display.

4.4 Visualization and Comparison in Rviz

Your implementation should publish two separate `OccupancyGrid` messages:

- One for the binary map (e.g., published on `/map/binary`).
- One for the log-odds (probabilistic) map (e.g., published on `/map/logodds`).

In Rviz, add two separate `Map` displays, one subscribing to `/map/binary` and the other to `/map/logodds`, so that you can directly compare:

- The abrupt state changes in the binary map.
- The gradual, uncertainty-reflecting updates in the log-odds map.

5 Evaluation and Expected Results

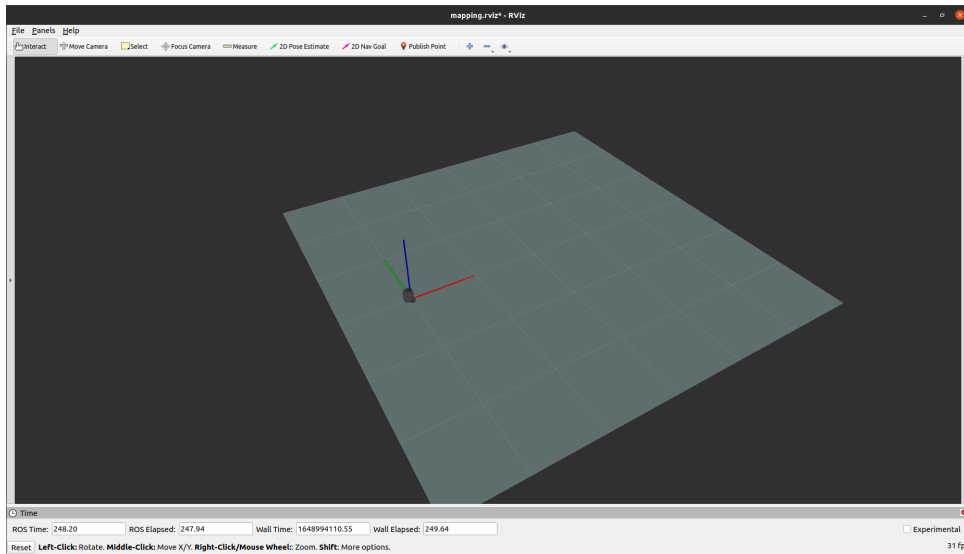


Figure 2: Initial grid (shown in a neutral color to denote unknown occupancy)

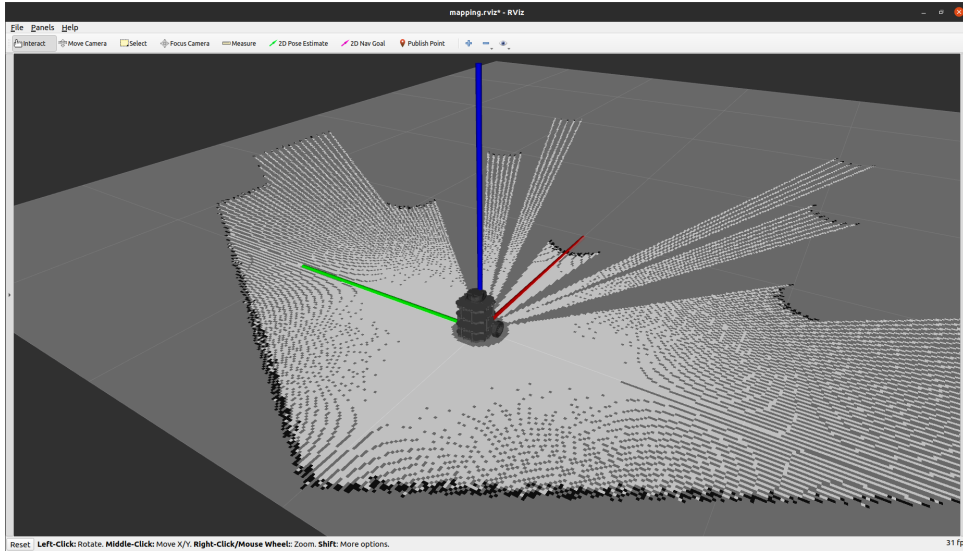


Figure 3: Initial mapping results, demonstrating early updates (using either binary or log-odds updates)

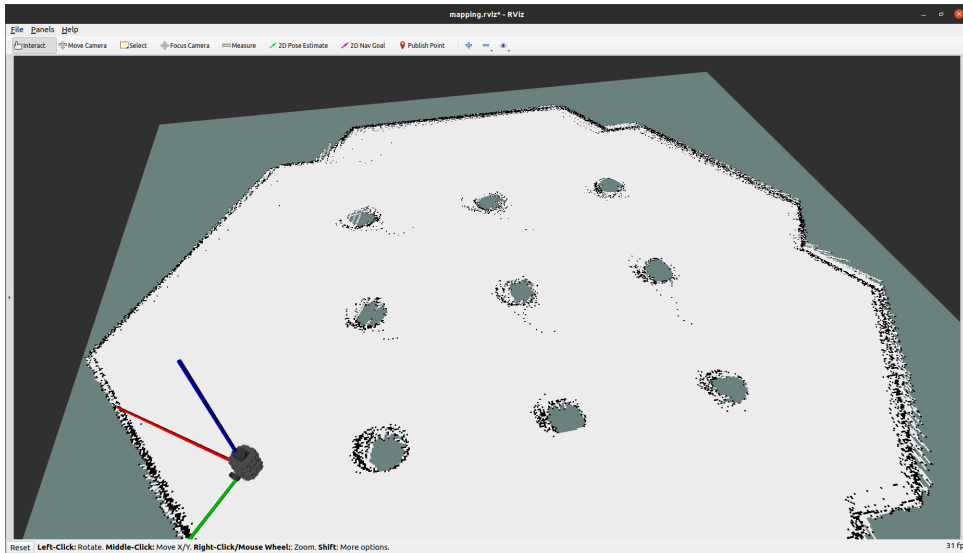


Figure 4: Binary mapping results

5.1 Visualization with Rviz

Once the occupancy grids are updated based on laser scans and robot movement, publish them on separate topics (e.g., `/map/binary` and `/map/logodds`). In Rviz, the occupancy grid displays should show:

- Initially, a grid with uniform color representing unknown occupancy.
- As the Turtlebot navigates, free cells appear in light colors (white) and occupied cells in dark colors (black).
- The log-odds map will display a nuanced gradient, whereas the binary map will show hard transitions.

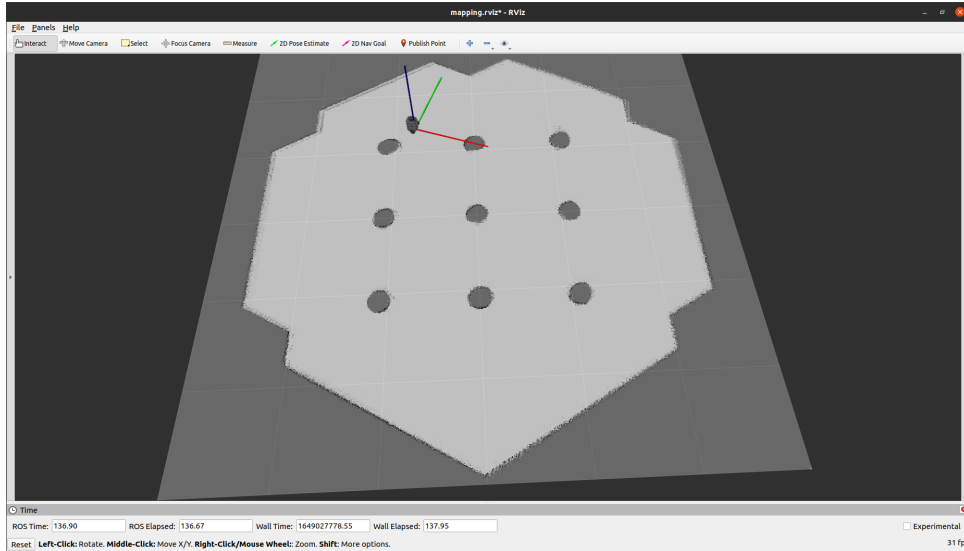


Figure 5: Final Result with log-odds

5.2 Testing and Debugging

- Verify that the coordinate transformation correctly aligns the robot's pose with the grid.
- For each laser beam, sketch the intersection with grid cells to ensure correct updates.
- Compare the binary map and the log-odds map; the log-odds method should more effectively reduce noise and reflect sensor uncertainty.

6 Submission

Send your node (`mapping.py`) as an attachment via email to: **cdtian@csd.uoc.gr** with subject "[CS-475] Assignment 4 submission". Remember to include your name and registration number. The deadline is **29/04/2025 23:59**.