Building an Occupancy Grid (Map) From LiDAR Data in ROS2

Table of Contents

Introduction *2*

Objective 2

Background ***3***

Lab Assignment ***5***

Submission ***7***

Introduction

Objective

The goal of this lab is to help us understand the processes of deterministic mapping and world representation by implementing a mapping routine for our GoBilda bots.

The core of the exercise includes the proper implementation of Bresenham’s ray tracing algorithm to create a local 15 x 15 meter map of our robot’s surroundings.

By the end of the activity, we should be able to implement a ROS2 Python node that publishes *nav_msgs/OccupancyGrid*, construct a 2D grid with occupancy values ranging from 0 to 100, and visualize the grid in Foxglove or RViz.

Background

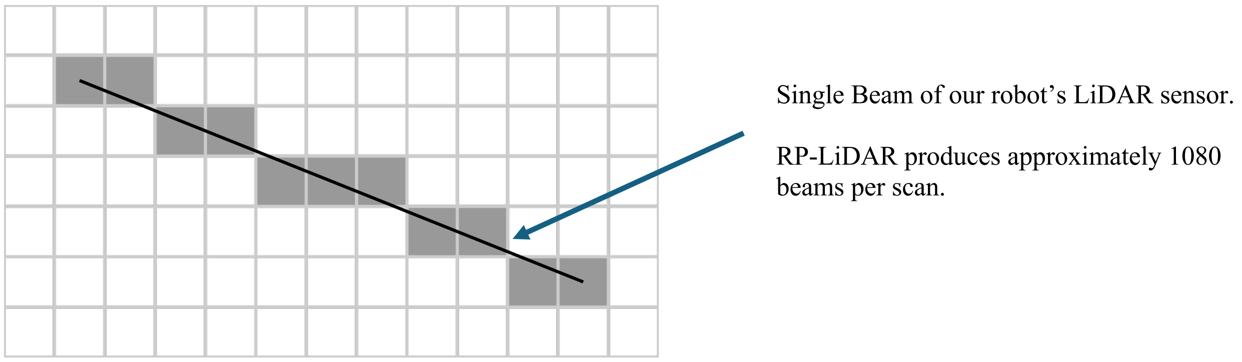
Bresenham’s Ray Tracing (or a line drawing algorithm)

The basic *line drawing* algorithm used in computer graphics is Bresenham’s Algorithm. This algorithm was developed to draw lines on digital plotters, but has found widespread usage in computer graphics.

It is commonly used to draw line primitives in a bitmap image (e.g. on a computer screen), as it uses only integer addition, subtraction, and bit shifting, all of which are very cheap operations in historically common computer architectures. It is an *incremental error algorithm*, and one of the earliest algorithms developed in the field of computer graphics.

While algorithms such as Wu's algorithm are also frequently used in modern computer graphics because they can support anti-aliasing, Bresenham's line algorithm is still important because of its speed and simplicity.

Example Grid World


Single Beam of our robot’s LiDAR sensor.
RP-LiDAR produces approximately 1080 beams per scan.

Bresenham’s algorithm will output the list of cells traced by the given line.

Example of the ray tracing algorithm working on a raster (or grid world). The point (0, 0) is at the top left corner, and the point (11, 5) is at the bottom right end of the line. The algorithm takes as input two endpoints of a line and decides which cells should be included in the trace of that line.

For our robot, the line pictured above translates to a single beam of the LiDAR; meanwhile, the cells that you trace should be marked as free space and the endpoint as occupied space.

Pseudo-code

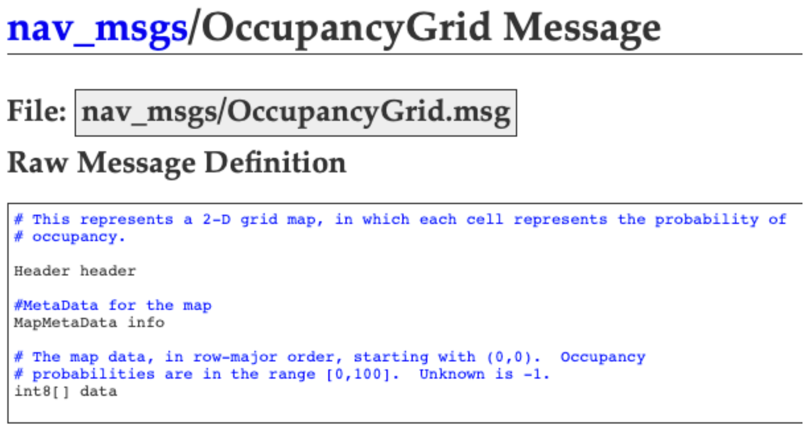
The general form for Bresenham’s algorithm can be implemented in the following way:

1. Compute the direction of both the *x* and *y* coordinates
2. Compute the *error term*
3. Start the while loop by *illuminate(x, y)*
4. Use the error term to decide whether to increment either the x coordinate, the y coordinate, or both
5. Increment/decrement the error term as necessary

```
plotLine(x0, y0, x1, y1)
    dx = abs(x1 - x0)
    sx = x0 < x1 ? 1 : -1
    dy = -abs(y1 - y0)
    sy = y0 < y1 ? 1 : -1
    error = dx + dy

    while true
        plot(x0, y0)
        e2 = 2 * error
        if e2 >= dy
            if x0 == x1 break
            error = error + dy
            x0 = x0 + sx
        end if
        if e2 <= dx
            if y0 == y1 break
            error = error + dx
            y0 = y0 + sy
        end if
    end while
```

Obstacle Layer & *nav_msgs/OccupancyGrid*



nav_msgs/OccupancyGrid Message

File: nav_msgs/OccupancyGrid.msg

Raw Message Definition

```
# This represents a 2-D grid map, in which each cell represents the probability of
# occupancy.

Header header

#MetaData for the map
MapMetaData info

# The map data, in row-major order, starting with (0,0).  Occupancy
# probabilities are in the range [0,100].  Unknown is -1.
int8[] data
```

Definition of the nav_msgs/OccupancyGrid ROS type. The ***info*** *variable (of type MapMetaData) contains information about the dimensions and resolution of the map. The* ***data*** *variable contains the actual map data, in row-major order.*

An occupancy grid map is a probabilistic tessellation of the robot's environment into cells, each storing a value representing the probability that an obstacle occupies the cell. The Obstacle Layer is a critical component that updates this grid using sensor data. Planar LiDAR (Light Detection and Ranging) provides a set of distance measurements (ranges) at known angular intervals, forming a scan.

The transformation from a LiDAR scan to a grid update is achieved through a ray-tracing algorithm (Bresenham's Line Algorithm or a similar discrete grid traversal method). For each laser beam:

- Free Space Update: The cells between the sensor's origin and the measured endpoint of the beam represent free space. The ray-tracing algorithm iterates through these cells, updating their occupancy 0 cost to represent the likelihood of being unoccupied.

- Occupied Space Update: The cell containing the measured endpoint of the beam, where the laser reflected off a surface, is updated to cost 100 to reflect a higher likelihood of being occupied.

This process, repeated over successive scans as the robot moves, incrementally builds a consistent and accurate map of the surrounding environment.

Lab Assignment

This assignment builds upon the provided starter code to implement a real-time obstacle layer node. The node will subscribe to LiDAR data from the */scan* topic and dynamically maintain an occupancy grid map centered on the robot. The implementation must correctly classify grid cells as free (0), occupied (100), or unknown (-1) based on live sensor readings.

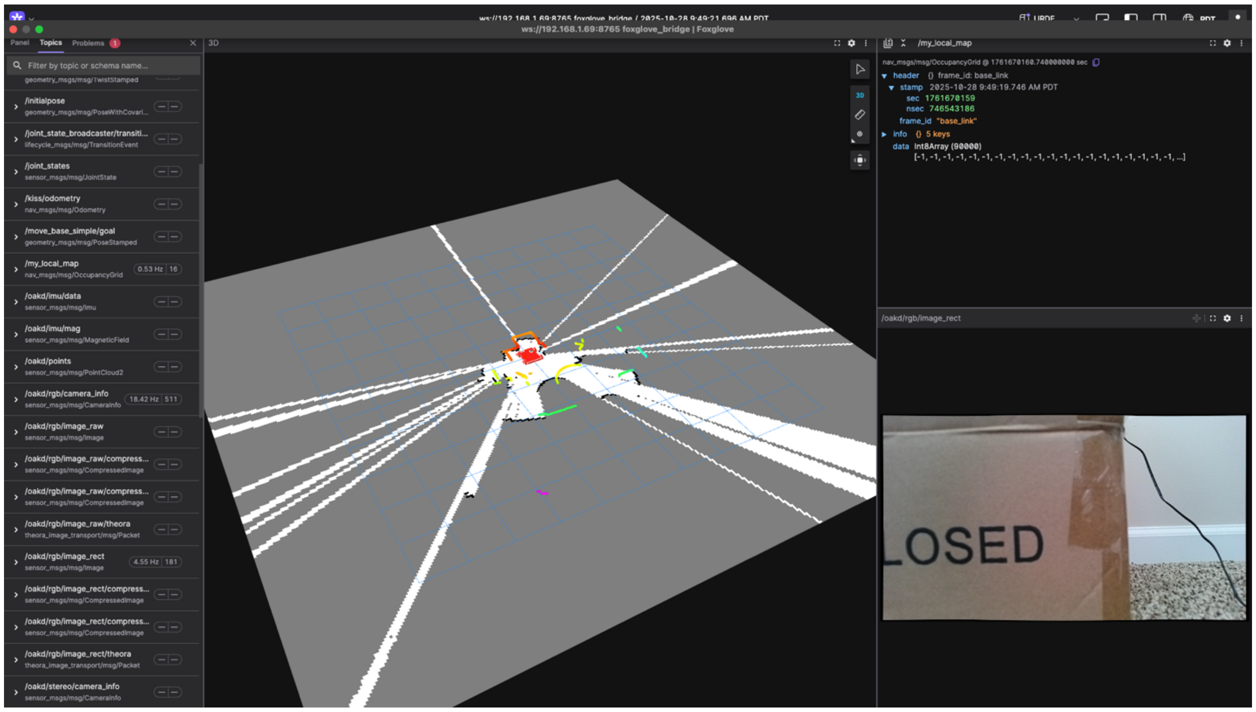
Task:

1. Starting from the provided example code, **implement a simple obstacle layer node** that will produce a live map for the robot. The robot should be at the center of the map.
2. **In a paragraph**, explain the differences you see when your solution runs on the simulated robot versus when it runs on the actual hardware.

Tips:

- The LiDAR topic is /scan, and the ROS2 type for the map is OccupancyGrid
- Should edit the Gobilda's URDF file to align with your real LiDAR sensor
- Use the map dimensions and resolution provided in the starter code
- Maintain the robot's position at the exact center of the map as specified in the initial code
- Ensure proper transformation from the laser frame to the map frame
- */scan* topic uses: *sensor_msgs/msg/LaserScan*
- **Note:** This assignment can be completed in *both* the simulation and hardware! Demo should be done in hardware.

Screenshot: A proper solution should look like the following



```
nav_msgs/msg/OccupancyGrid @ 1761670160.740000000 sec
▼ header  {} frame_id: base_link
  ▼ stamp  2025-10-28 9:49:19.746 AM PDT
      sec  1761670159
      nsec  746543186
    frame_id  "base_link"
▶ info  {} 5 keys
  data  Int8Array (90000)
        [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, ...]
```

Submission

You are expected to demo your code in the next lab section. Please ensure that I record your name after your successful demo.

After you’ve demo’ed your code and passed, I will need you to submit your source code. Compress your entire ROS2 workspace and upload the files to Canvas.