1. Overview

This project experiment implements a sampling-based global planner—Rapidly-Exploring Random Tree (RRT)—for mobile-robot navigation in a known occupancy map.

The objective is to:

1. Load and interpret an occupancy grid created from prior SLAM exploration (my_map.pgm/.yaml).
2. Generate a collision-free path between arbitrary start/goal poses supplied at run-time.
3. Publish the path to downstream modules and visualise it for evaluation.

2. Implementation Details

a. Workspace & Map Server

Item	Value
Package	rrt_pathfinding
Map files	my_map.pgm, my_map.yaml (root of workspace)
Launch order	Terminal 1: roscore Terminal 2: rosrun map_server map_server my_map.yaml

The map server publishes /map (nav_msgs/OccupancyGrid). Each message carries:

- info.resolution = 0.05 m / pixel
- info.origin = [−10 m, −10 m] → lower-left world corner
- data[] = -1 (unknown) | 0 (free) | 100 (occupied)

b. RRT Node (RRT_node.py)

- **Map Handling (_map_callback)**
  - Decode metadata → resolution, origin, width, height.
  - Reshape 1-D data[] → 2-D array (height × width).
  - Greyscale conversion:
    255 = free, 0 = obstacle, 127 = unknown.
  - Flip vertically with np.flipud so the pixel frame matches the ROS world frame.
- **Planning Pipeline (_start_goal_callback)**

Step	Action
1	Extract start & goal in meters from /start_goal
2	Call rrt_pathfinder (from rrt.py) Internally it: • Converts meters → pixels • Builds an RRT with collision tests is_free against map_img • Returns a list of points path_px (pixels)
3	Flatten path_px and publish on /trajectory
4	Visualise path: Red poly-line, green start "S", blue goal "G"

	Filename encodes real-world start/goal
5	Publish /path_ready = True so the Motion Planner can continue to unblock (prompt new inputs)

The node is encapsulated in a class (RRTNode) for clean state management and easier debugging.

c. Planning Pipeline (_start_goal_callback)

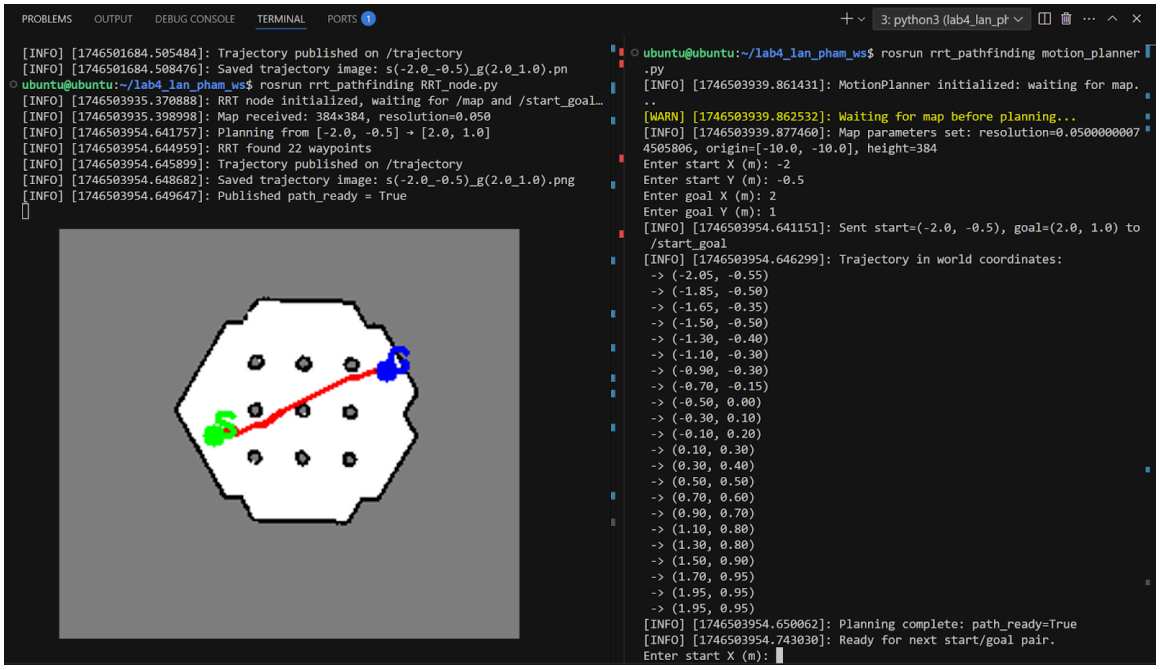
A user-interactive CLI that cycles indefinitely:

1. Wait for /map → cache resolution, origin, map_height.
2. Prompt users for start (x,y) and goal (x,y) in meters.
3. Publish /start_goal, then block until /path_ready = True.
4. Upon /trajectory, convert every (px, py) back to (wx, wy):
   $w_x = x_{origin} + p_x * resolution$
   $w_y = y_{origin} + (map_{height} - p_y) * resolution$
5. Print the world-space path to the console and re-prompt.

3. Results

Figures below overlay the path on map_img; start is green, goal is blue, obstacles are black.

- Fig. 1 – Start (-2.0, -0.5) to Goal (2.0, 1.0)



- Fig. 2 – Start (-0.5, -2.0) to Goal (1.0, 2.0)