School of Computer Science and Applied Mathematics
University of the Witwatersrand, Johannesburg

# COMS4045A Robotics Assignment

Lisa Godwin (2437980), Brendan Griffiths (2426285),
Nihal Ranchod (2427378), Zach Schwark (2434346)

June 21, 2024

## 1 Map Generation

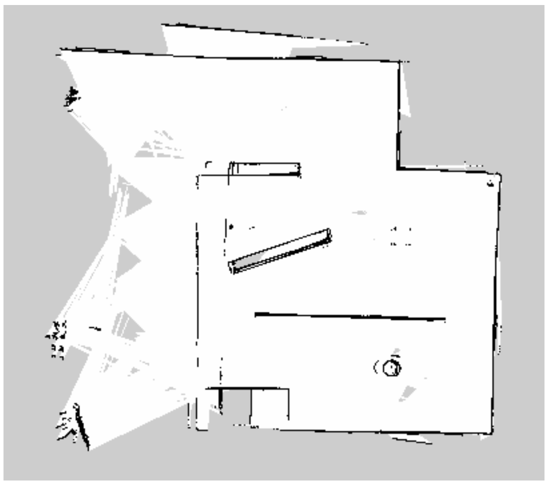

Figure 1: The map generated from GMapping

The GMapping algorithm was utilised to create a map of the environment shown in figure 1 by moving the robot forward and having it spin to scan its surroundings. Further navigation was done along the boundaries and obstacles to ensure accuracy.

## 2 Map Processing

### Erosion

Erosion is a preprocessing step used to clean up the occupancy grid map by removing small obstacles and smoothing out narrow passages. Our implementation converts the map to a grayscale image with free space as 255 (white) and obstacles as 0 (black) using thresholding. We apply erosion with a $10 \times 10$ pixel kernel to enlarge free spaces and remove small obstacles. This eroded map shown in figure 2 is then used to generate the navigation graph, ensuring that vertices and edges are placed in navigable areas with clearance for safe movement.

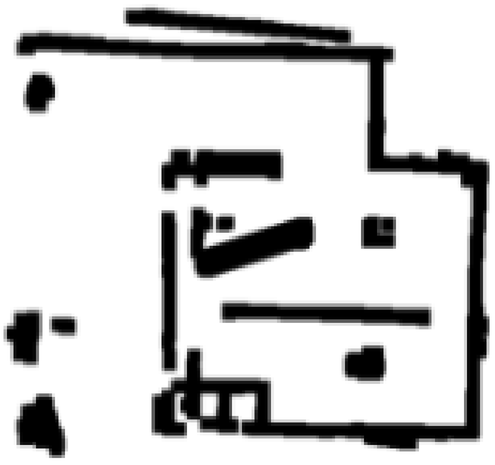

Figure 2: The eroded map used to generate the navigation graph

### Graph Generation Algorithm - PRM

The graph generation process uses the Probabilistic Roadmap (PRM) method, which is effective for high-dimensional and complex environments. We generate a large set of random points within the map and filtering out those that lie within obstacles using the eroded map. To reduce graph density, we use k-means to cluster nearby points and replace them with their centroid. K-Nearest Neighbours is then used to determine the closest vertices for each vertex, and each potential edge is validated to ensure it does not intersect with obstacles. Additionally, using a PID controller allows the robot to move in straight lines, simplifying navigation along the generated graph edges.

### Edge Invalidation Process

The edge invalidation ensures collision-free paths by using the `is_valid_edge` function, which interpolates points along each edge and checks for obstacles, considering a buffer area around the edge for added safety. During vertex insertion, the algorithm attempts to connect the new vertex to the nearest existing vertex. If an edge is invalid, it is removed and the next closest vertex is checked until a valid connection is found or all possibilities are exhausted.

### Graph Generation as a Preprocessing Step

Graph generation as a preprocessing step is done to optimise performance. This approach was used because the k-means algorithm is computationally expensive. By generating the map and the corresponding navigation graph before starting the robot, we can avoid the overhead of real-time graph generation.

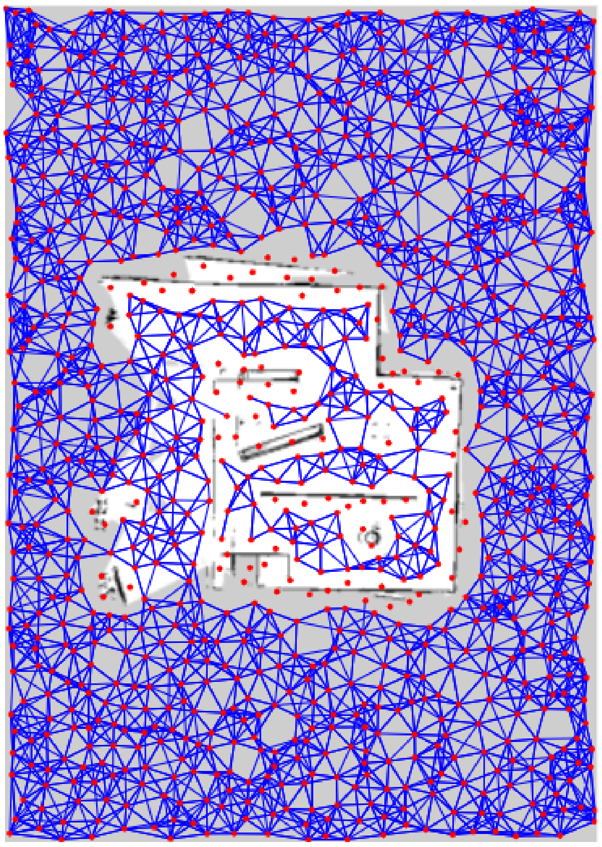

Figure 3: The navigation graph overlaid on the generated map

## 3 Surveillance Bot

SurvBot operates as a state machine with the following states:

- *IDLE* - Do nothing
- *PATHFIND* - Construct a move queue to the specified navigation target
- *NAVIGATE* - Move forward through the move queue, go to *IDLE* when the navigation target is reached
- *PATROL* - Move forwards and backwards through the move queue
- *SCAN* - Rotate in place for a specified amount of rotations

Communicating with SurvBot is done through the following ROS topics:

- */survbot/state* - Change the state of SurvBot
- */survbot/navigate/goal* - Change the navigation goal of SurvBot
- */survbot/scan/rotations* - Change the number of complete rotations and their speed, in a *SCAN*

### Pathfinding

The pathfinding algorithm employed is A* over the graph previously generated over the map shown in figure 3.

When a navigation command is received, the current position of SurvBot and the target location are added as vertices to this graph. A* is then executed where the algorithm constructs a sequence of way-points for SurvBot to follow. If the path is blocked or invalid, SurvBot dynamically replans to ensure it reaches the target.

### Moving

SurvBot's movement is controlled using a PID (Proportional-Integral-Derivative) controller. The PID controller adjusts SurvBot's yaw to face the next waypoint before moving forward in a straight line. This two-step control mechanism—first rotating to align with the waypoint, then moving towards it—creating stable navigation between waypoints.

### Resources

- Code: GitHub Repository
- Video: YouTube Demonstration