Faculty of Engineering
Cairo University

Robotics Project

Team 7

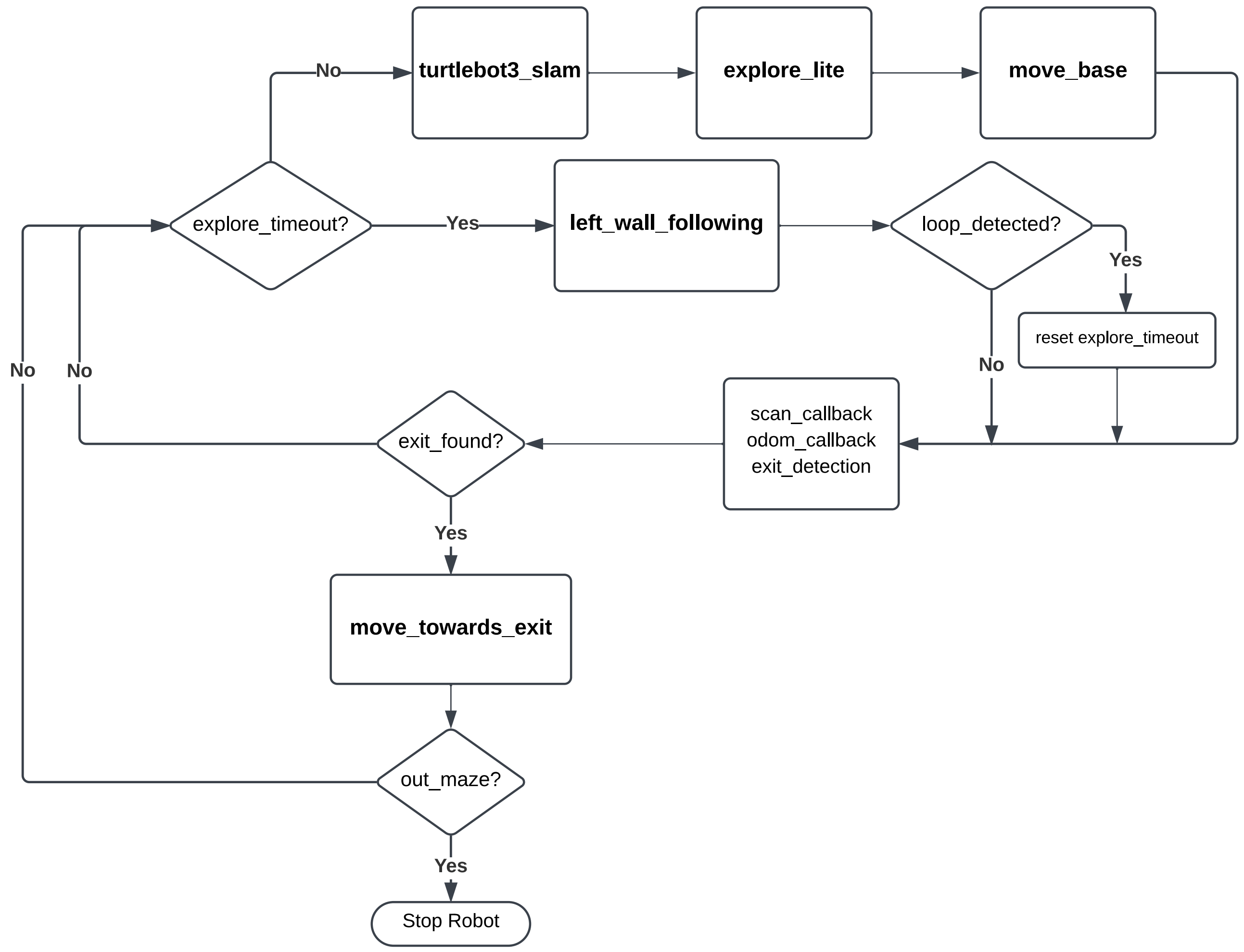
Name	SEC	BN
كريم علاء عبد الله	2	3
أحمد أسامة فوزي زهران	1	6
محمود أسامة محمود خطاب	2	15
محمد عادل محمد عز الدين	2	9

Under Supervision of:
Eng. Hassan Mohamed

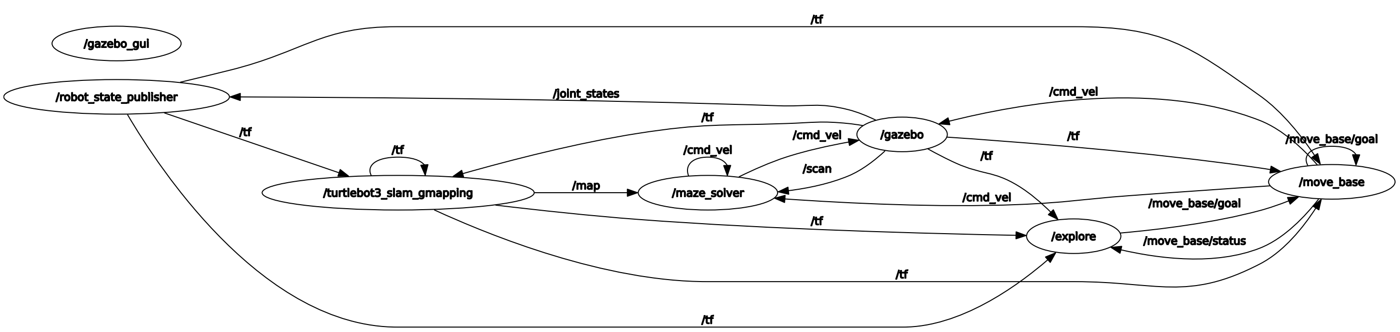
Contents

System Design 3
Ros Nodes 3
System Design Explanation 4
1. gazebo_ros 4
2. turtlebot3_description 4
3. turtlebot3_slam 4
4. explore_lite 4
5. move_base 4
6. left_wall_following 4
7. exit_detection 4
8. move_towards_exit 4
Topics 5
1. Subscribers 5
2. Publishers 5
Maze Solving approach 5
Output Visualizations 5
1. Maze 1 5
2. Maze 2 6
How to Run the Project 6

System Design



Ros Nodes



System Design Explanation

1. gazebo_ros

This package serves as a bridge between ROS and the Gazebo simulation environment. It is used for spawning robot, controlling simulated sensors and actuators, and synchronizing simulation physics with real-world robot behaviour.

2. turtlebot3_description

This package provides the URDF (Unified Robot Description Format) and necessary files to define the physical and visual characteristics of the TurtleBot3 robot.

3. turtlebot3_slam

This package is used for implementing Simultaneous Localization and Mapping (SLAM) on the TurtleBot3. It enables the robot to build a map of its environment while simultaneously tracking its position within map.

4. explore_lite

This package is an exploration tool that allows a robot to autonomously explore an unknown environment. It uses frontier-based exploration, where the robot identifies unexplored regions (frontiers) and navigates to them, gradually building a complete map of the environment.

5. move_base

This package provides the navigation stack for ROS, enabling a robot to plan and execute paths to a target location. It is used by explore package to move the robot.

6. left_wall_following

1. **Approach a Wall:** The robot starts by moving forward, adjusting its angular and linear velocities until it detects a wall on its left side. This is done by monitoring sensor readings for proximity to obstacles.
2. **Follow the Left Wall:** Once near a wall, the robot follows it by maintaining a consistent distance. If too close, it backs away slightly; if too far, it moves closer. If an obstacle is detected directly ahead, the robot stops and turns until the path is clear, then resumes wall-following.
3. **Visited Locations Tracking:** Monitors the robot's path to detect loops. Revisits within a tolerance and time threshold trigger loop detection, clearing the visited locations list.

7. exit_detection

1. **Subscribe to Laser Scan and Odometry Topics:**
   - i. Process scan data to get distances and angles to obstacles.
   - ii. Obtain the robot’s position and orientation from odometry.
2. **Calculate Potential Points:**
   - i. For each open direction in the laser scan:
     - Transform the point using the robot’s current position and orientation.
     - Store the global coordinates of the potential point.
3. **Boundary Check:**
   - i. Compare each potential point with the maze’s predefined boundaries.
   - ii. If a point lies outside these boundaries, flag it as the exit.

8. move_towards_exit

1. **Calculate Position and Target:** Retrieve the robot's position and orientation, then compute the distance and angle to the exit point.
2. **Move or Avoid Obstacles:** Move towards the exit by setting linear and angular velocities. Stop and reorient if obstacles are detected.

Topics

1. Subscribers

a. /scan

To retrieve laser scan data for obstacle detection, navigation, and wall-following.

b. /odom

To track the robot's position, orientation and current speed.

c. /map

To receive the occupancy grid map for maze representation.

2. Publishers

a. /cmd_vel

To control the robot's linear and angular velocities.

Maze Solving approach

1. **Starting the Exploration**: The robot begins by recording the time and starting its exploration of the maze. It will keep moving and searching for the exit.
2. **Exploring the Maze:** As the robot moves around, it constantly checks its surroundings, trying to find the exit. It has a time limit for exploration — if it takes too long (25 minutes), it stops and reassesses.
3. **Exit Found:** If the robot finds the exit, it will move towards it and try to leave the maze.
4. **Handling Failures:** If the robot doesn't find the exit after several attempts, it will change its approach. It tries a strategy where it follows the left wall, which can help it navigate out of the maze.
5. **Time and Distance Reporting**: After finishing, the robot calculates the time and the distance taken to solve the maze.

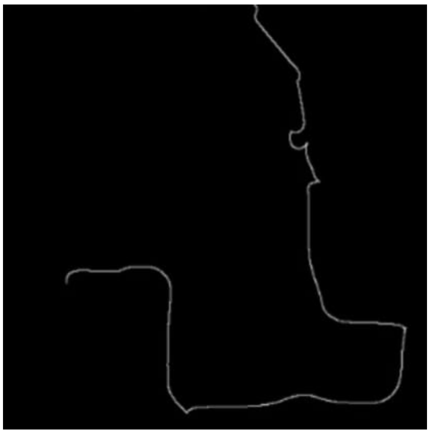
Output Visualizations

1. Maze 1

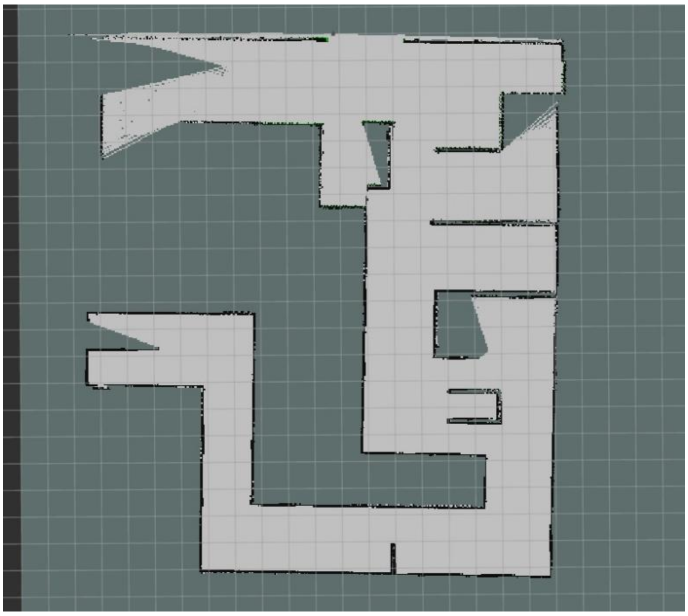
Elapsed Time: 4.9 minutes

Total Distance: 47 meters

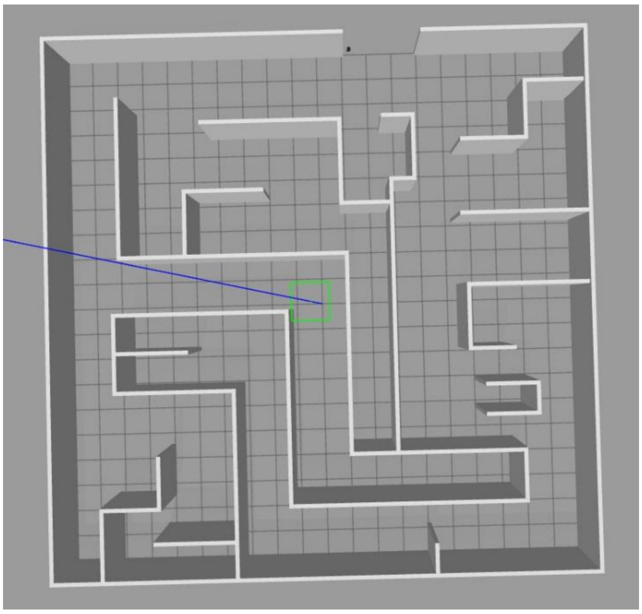
Average Speed: 0.22



Path 1



Rviz Map 1



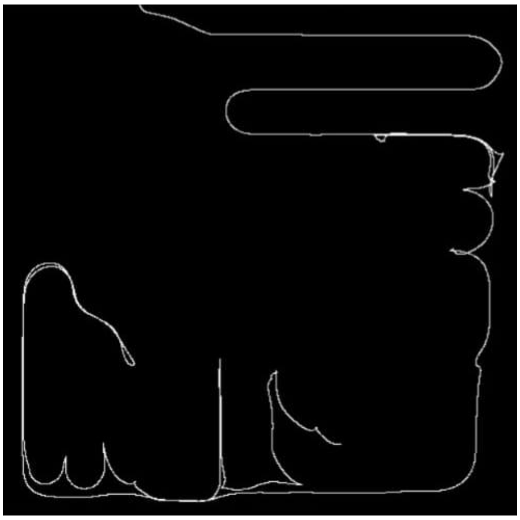
Gazebo Map 1

2. Maze 2

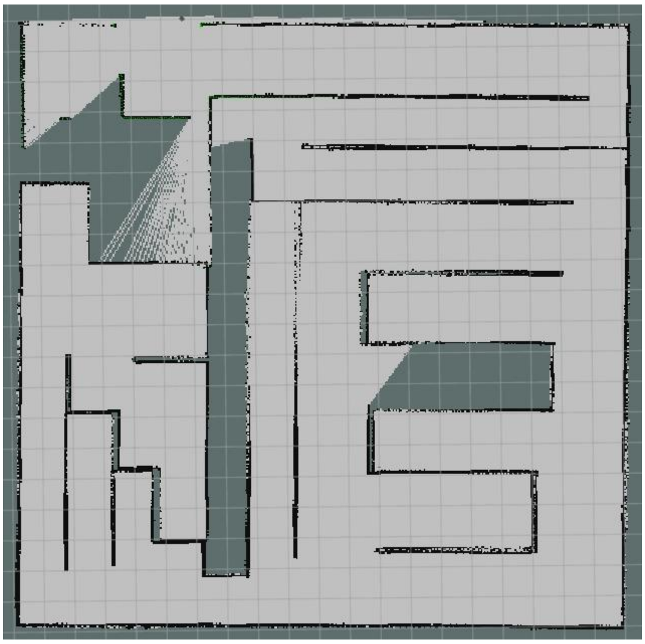
Elapsed Time: 13.6 minutes

Total Distance: 149 meters

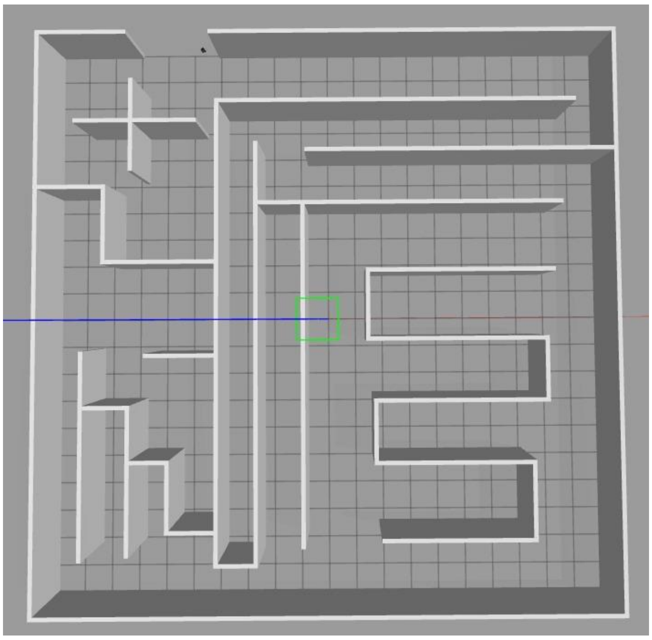
Average Speed: 0.22



Path 2



Rviz Map 2



Gazebo Map 2

How to Run the Project

- Edit ‘maze_world.launch’ to change robot initial position and the world location
- Open first terminal and type ‘roslaunch turtlebot maze_world.launch’
- Open second terminal and type ‘rosrun turtlebot maze_solver.py’