ROS 2 C++ Coding Test Instructions: TurtleBot3 Maze Navigation

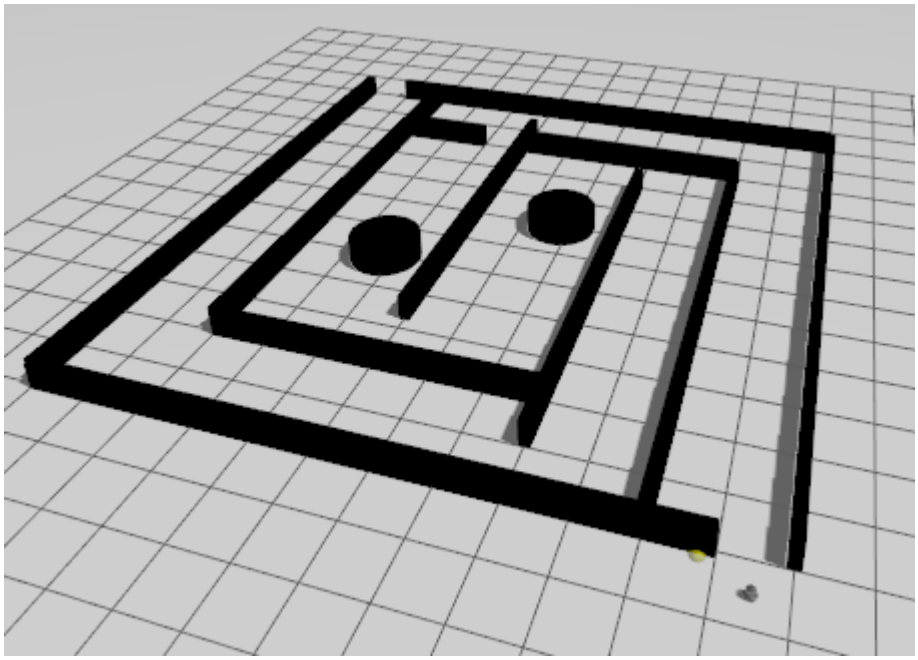
Duration: 24 hours

Overview

This test evaluates your ability to work with core ROS 2 concepts, implement fundamental robotics algorithms, and write clean, modular C++ code. You will implement a motion planning and control stack to autonomously navigate a TurtleBot3 robot through a maze environment.

Scenario

You are provided with a gazebo maze environment with a turtlebot3 spawned inside. You are given a 2D occupancy grid map of a maze (/map) and fixed entry and exit points to the maze. Your TurtleBot3 starts at the **entry point**, and your goal is to safely reach the **exit point** without colliding into any obstacles.



Task Description:

Implement a ROS 2 system (in C++) which is modular and cleanly structured. The following topics are available to interface with the simulation:

Subscriptions:

- /virya_test/map -> nav_msgs/OccupancyGrid -> maze map as an occupancy grid where 0 represents free space and any value > 0 represents an obstacle. -1 means that the status of that tile unkown.
- /virya_test/entry-> geometry_msgs/PoseStamped -> the coordinates of the exit
- /virya_test/exit-> geometry_msgs/PoseStamped -> the coordinates of the exit
- /virya_test/odom -> nav_msgs/Odometry -> the robot's pose in the map frame

Publishers:

- /cmd_vel : geometry_msgs/TwistStamped -> to control the turtlebot

Coding instructions:

- The installation instructions are provided in the readme file of the repo.
- Create a new ROS 2 package to contain your solution.
- Implement a **waypoint generation and tracking system** using C++. Your implementation should rely only on the provided topic interfaces for input and control.
- The **entire motion planning and control stack must be implemented from scratch**.
- Use only **standard C++** and **core ROS 2 packages** (rclcpp, geometry_msgs, nav_msgs, etc.).
- Use of third-party or external libraries (e.g., OpenCV, Boost, Eigen, navigation2) is not allowed.
- Ensure your planner includes a **safety padding mechanism** (e.g., an inflated obstacle buffer of 0.2 meters) to prevent the robot from colliding with maze walls.
- Clearly document any **assumptions or safety considerations** you have taken into account.
- The robot **must stop** once it reaches the maze exit.

- Your solution will be evaluated on:
    - Functional correctness (does the robot successfully solve the maze?)
    - Code quality and modularity
    - Adherence to ROS 2 best practices

Submission Instructions:

- You'll have 24 hours to submit your solution. However, **earlier submissions will have an added advantage!**
- Create a video recording of the turtlebot3 moving from Maze's entry to exit.
- Create a plot of the planned trajectory and the actual path taken by the robot. This should be visualized on rviz2.
- Create a zip folder containing your solution code, plots and screen recordings and share it via mail.
- For your reference, this repository includes example test results to help guide your implementation (demo_path.png).
- Feel free to reach out in case you run into any issues (especially in installation).