Multi-Map Navigation and Wormhole Implementation

Task Overview

In this assignment, you will work on a multi-map navigation system where a robot can navigate between different mapped rooms. Each room will be mapped in separate sessions, and a "wormhole" mechanism will be created to allow the robot to switch between rooms. This mechanism will include an overlapping region around the wormhole and use an SQL database to save the wormhole positions. The assignment will require you to write C++ code, create an action server, and manage navigation between maps.

Challenge

Creating a multi-map navigation system requires mapping different rooms, establishing connections (wormholes) between them, and ensuring seamless navigation. The robot must be able to recognize when it needs to switch maps and navigate accordingly.

Scenario

Imagine a robot operating in a facility with multiple rooms. Each room is mapped separately, and the robot needs to switch between these maps seamlessly. The wormhole mechanism enables the robot to transition from one room to another through overlapping regions.

Task Objectives

1. Mapping and Wormhole Creation:
   - Map each room in separate sessions. Each map should only contain one room.
   - Identify and create overlapping regions (wormholes) between rooms.

2. SQL Database Integration:
   - Use an SQL database to save the positions of the wormholes

.

3. C++ Code Implementation:
   - Write C++ code to manage navigation between maps.
   - Implement an action server to handle navigation goals.

4. Action Server Implementation:

- Create an action server where the goal contains the position of the goal and a string with the target map name.
- If the robot is in the same map, directly send a move_base goal.
- If the robot needs to switch maps, navigate to the wormhole, switch maps, initialize, and move to the target position.

Deliverables

1. Mapped Rooms:

   - Separate maps for each room with defined wormhole regions.

2. SQL Database:

   - Database schema and entries for wormhole positions.

3. C++ Code:

   - Source code implementing the navigation logic and action server.

4. Action Server:

   - Fully functional action server handling multi-map navigation goals.

5. Documentation:

   - Detailed documentation of the code functionality.

Candidates can score additional points by :

- including the algorithm or pseudocode for their trajectory collection, storage, and visualization processes in the README file.
- Designing the nodes in a modular fashion, allowing for easy integration into existing ROS systems and flexibility for future enhancements or modifications.
- Providing clear and comprehensive documentation for the codebase, including comments within the source code, README file

- Utilizing Object-Oriented Programming (OOP) principles in C++, including classes with separate header and source files for enhanced modularity and maintainability.
- Avoiding hard coding of file paths and topic names in the source code, ensuring flexibility and ease of configuration.
- Adhering to coding guidelines for improved readability and maintainability.

Note:

- The candidate can make use of our ROS wiki or any existing projects for AMR functionality (eg turtlebot3)
- Make a recording showcasing the functionality of your ROS package, including navigation between each map, switching maps, and localisation in a new map.
- Once completed, please submit the recording along with the ROS package you've developed to the provided Submission Form. The maximum allotted timeline for completing the assignment is 48 hours from the reception of the task.

Below are helpful links to guide the candidate in their development process:

https://wiki.ros.org/actionlib#Tutorials
https://wiki.ros.org/database_interface
https://wiki.ros.org/move_base
http://wiki.ros.org/AnscerRobotics/AR100