Multi-Map Navigation and Wormhole Implementation

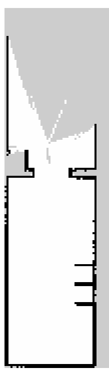
Robot Used: Turtlebot3

Environment Used: Turtlebot House environment

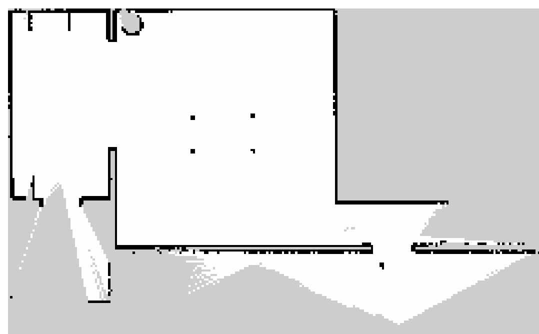
Ros version: ros2 humble

Mapping and Wormhole Creation

Room mapping was performed using the slam_toolbox and maps were saved using the mapsaver node in Nav2. Each saved map generates two files: a YAML file containing resolution, origin, and other metadata, and an image file.



Img1: room2 mapped



Img2: Room2 mapped

The YAML files were utilized to identify "wormholes" (overlapping areas between maps). In this initial implementation, maps of two rooms were created. A Python script was developed to detect these overlapping regions and store their coordinates in an SQL database.

Wormhole Detection Algorithm:

Written a python script for finding the wormhole between 2 maps, which mainly contains:

- Loads two occupancy grid maps (room1.yaml, room2.yaml).
- Converts the map images into real-world coordinates using resolution and origin.
- Determines overlapping occupied regions between both maps.
- Extracts the world coordinates of these overlap pixels.
- Saves them into an SQLite database (map.db) under a table named overlaps.

The script flow:

1. Load map data

    a. Loads room1 and room2 maps, assuming the same resolution.
2. Compute canvas size
    a. Finds min and max extents of the two maps in world space.
    b. Calculates combined canvas size in pixels that can hold both maps.
3. Create empty canvases
    a. Creates two boolean masks, one for each map.
4. Align and draw both maps
    a. Calculates pixel offsets from the shared canvas origin.
    b. Translates map pixels into the large canvas.
    c. Occupied pixels are identified as those not equal to 205 (205 = unknown in ROS maps).
5. Compute overlap
    a. Identifies pixels that are occupied in both maps.
6. Extract overlapping world coordinates
7. Insert overlaps into SQLite DB
    a. x, y: real-world coordinates of the overlapping point.
    b. region: a string indicating the map pair (e.g., room1&room2).

SQL Database Integration

1. The `sqlite3` library in Python was used to create an SQL database.
2. A table was created to store the coordinates of overlapping regions. The table includes x and y coordinates and a "region" identifier, named "room1&room2" to represent the overlapping region between room1 and room2.
3. The SQL table schema is defined as follows:

```
CREATE TABLE IF NOT EXISTS overlaps (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  x REAL,
  y REAL,
  region TEXT
);
```

The coordinates of the identified wormholes were inserted into the `overlaps` table.

Action Server Implementation

A custom action was created to receive the target pose and map name for robot navigation. The action definition includes:

```
#Goal
geometry_msgs/Pose target_pose
string map_name
#Result
bool success
#Feedback:
float32 progress
```

MultiMapManager ROS 2 Node

This ROS 2 node allows a robot to:

- Navigate across multiple maps.
- Transition between maps using a common overlap region stored in an SQLite3 database.
- Automatically switch maps after reaching the defined overlap point.
- Execute navigation goals using the standard NavigateToPose action.

It uses a custom action called multi_map_goal to accept map name + pose goals from the user

Action Server:

1. Action Server receives a goal: a target map name and a pose.
2. If the target map is different from the currently loaded map:
    - The node looks up an overlap point between the current and target map using an SQLite database.
    - The robot navigates to this overlap point.
    - After reaching it, the map is switched using /map_server/load_map.
3. After map switching:
    - The robot navigates to the final target pose inside the new map.

Handle_accepted method in action server

1. Parses the target map from the goal.
2. If the map is already loaded, send the final pose goal directly.
3. If the map is **different**:
    - Look up an **overlap region** between current and target map in the database.
        - First tries roomA&roomB by sending the following query

```
SELECT x, y FROM overlaps WHERE region = ? ORDER BY
RANDOM() LIMIT 1;
```

Where region is roomA&roomB, which results in a random point in the selected region

        - If not found, try reversed roomB&roomA, and send the query and check any points available.
    - Sends the robot to the overlap point using the NavigateToPose action server.
    - Waiting for completion.
    - Calls change_map() with the new map path, which calls /map_server/load_map service with the name of the new map which is given by the user.
    - After a short delay, sends the robot to the final pose using the NavigateToPose action server.

Project Structure

The project contains 2 packages

1. **Multi_map_action**: The custom action has been defined here
2. **Multi_map_navigation**: The multi map navigation using the custom action is performed here, The action server is defined here

A map folder is provided in the navigation package, which contains the map of the rooms and the python code used to find the wormhole between the maps.

To start the node

```
ros2 run multi_map_navigator multi_map_navigator_node
```

To send a goal

```
ros2 action send_goal /multi_map_goal
multi_map_action/action/MultiMapGoal  "{map_name: 'room2',
target_pose: {position: {x: 1.0, y: 2.0, z: 0.0}, orientation: {w:
1.0}}}"
```

github url: link

Submitted by,

Vidhun V Warrier

vidhunvwarrier@gmail.com

+91 8592802540