Lab 5: Mapping & Path Planning

CDA 4621
Fall 2024
Lab 5: Mapping & Path Planning
Due Date: 12-9-2024 by 11:59pm

Objective

The purpose of this lab is to introduce the fundamental concepts of mapping and path planning in mobile robotics, focusing on how a robot can represent and navigate a structured environment. Students will learn to create a world model, determine the robot's position within that model, and calculate the most efficient path to a specified goal location. In real-world robotics, mapping and path planning are essential for tasks such as autonomous navigation, search and rescue operations, and logistics, where robots need to operate in dynamic or predefined environments. By completing this lab, students will develop skills that are foundational for enabling robots to understand, plan, and execute movements through spatial environments.

Requirements

Programming: Python
Robot: Webots 2023b (FAIRIS)

Sensors Utilized

To successfully complete this assignment, you will need to utilize several sensors on the robot. Each sensor provides unique data essential for robust navigation and environmental awareness. Below is an overview of the sensors, their specific functionalities, and references to previous assignments for guidance on usage:

- **Encoders:** The encoders track the rotation of each wheel, enabling the calculation of distance traveled and rotational changes. By using encoder readings, you can estimate the robot's displacement and changes in orientation over time, aiding in precise position tracking. *Refer to Lab 1 for an introduction to encoder usage.*
- **Compass:** The compass sensor returns the robot's orientation relative to the world frame, typically measured in degrees from the east direction (the x-axis). This sensor is essential for maintaining and adjusting the robot's heading during navigation tasks. *Refer to Lab 1 for initial setup and utilization of the compass.*
- **Lidar:** The Lidar sensor captures a 360-degree range image of the environment by measuring distances to obstacles and landmarks at various angles. This information allows the robot to map surroundings, avoid obstacles, and make path-planning decisions based on spatial awareness. *Refer to Lab 2 to review Lidar integration and data handling.*
- **Camera with Object Recognition:** The onboard camera, integrated with object recognition capabilities, allows the robot to detect and differentiate various objects in its environment. Specific landmarks, such as colored cylinders, can be identified, providing visual cues for navigation and goal localization. *Refer to Lab 3 for implementation details on using the camera and object recognition.*

Each of these sensors provides critical information that will need to be integrated into your navigation algorithm to ensure accuracy and efficiency. Be sure to review the respective labs to understand how to utilize these sensors effectively as you design your solution.

Global Reference Frame and Grid Cell Numbering

The reference frame is set up as follows:

- The positive y-axis points North, while the negative y-axis points South.
- The positive x-axis points East, and the negative x-axis points West.
- The origin, (0,0), is located at the center of the arena.

The arena is divided into a 5x5 grid of 25 cells, numbered from 1 to 25. Each grid cell is a 1 x 1 meter square, creating a structured layout for navigation and localization.

Refer to this layout as you plan and execute navigation strategies within the defined grid.

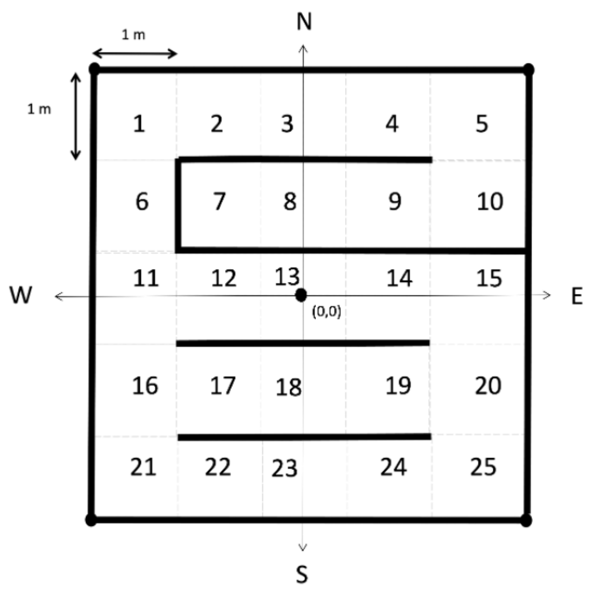
Wall Configuration Data Structure

For this assignment, you will need to implement a data structure to represent the wall configuration of each grid cell in the arena. This structure will allow your robot to recognize and navigate around obstacles effectively.

As a conceptual guide, consider representing the wall configurations using a 25x4 matrix format, as illustrated in Figure 1. In this example:

- The matrix corresponds to the 25 grid cells in the arena.
- Each cell in the matrix contains four entries representing the presence or absence of walls in the West-North-East-South (WNES) orientation.
- A ”W” (wall) entry indicates the presence of a wall, while an ”O” (open) entry denotes the absence of one.

While you are encouraged to design your own data structure, this example can serve as a reference to help you get started.



Cell	Walls (WNES)
1	WWOO
2	OWOW
3	OWOW
4	OWOW
5	OWWO
6	WOWO
7	WWOW
8	OWOW
9	OWOW
10	OOWW
11	WOOO
12	OWOW
13	OWOW
14	OWOW
15	OWOO
16	WOOO
17	OWOW
18	OWOW
19	OWOW
20	OOWO
21	WOOW
22	OWOW
23	OWOW
24	OWOW
25	OOWW

FIGURE 1. Example of a wall configuration representation. The left side illustrates the physical world, while the right side shows the corresponding 25x4 matrix representation using WNES orientation.

Think carefully about how your data structure will allow you to efficiently check for walls in each direction and update it as the robot moves through the arena.

Background: Path Planning with Wave Front Planner

The Wavefront Planner is a grid-based path planning algorithm commonly used in robotics for navigating a robot from a start position to a goal position. This algorithm operates on a grid map by propagating values from the goal cell to other cells in a wave-like manner, making it suitable for environments with known obstacles. The primary goal of the Wavefront Planner is to determine the shortest path to the target by expanding outwards from the goal cell until the start cell is reached.

How the Algorithm Works.

- **Initialize the Goal Cell**: The algorithm begins by assigning the goal cell a starting value, typically zero, indicating the destination.
- **Wave Expansion**: From the goal cell, values incrementally increase for each neighboring cell, representing the number of steps required to reach the goal. Only valid, traversable cells (non-obstacle cells) are considered in this propagation.
- **Marking Path from Start to Goal**: Once all cells are filled, the robot begins at the start cell and moves to adjacent cells with decreasing values, ultimately reaching the goal cell along the shortest path.
- **Obstacle Handling**: Cells representing obstacles are left unassigned or marked with an infinite or high value, ensuring the robot does not travel through them.

This approach, which supports four-point (or optionally eight-point) connectivity for movement, enables the robot to navigate efficiently in grid environments with obstacles. Although simple, the wavefront planner is effective for pathfinding in static maps where obstacles and the grid structure are known beforehand.

High-Level Pseudo Code.

```
WavefrontPlanner(grid, start, goal):
    1. Initialize all cells in the grid with a zero, indicating unvisited.
    2. Set the goal cell to zero.
    3. Set the start cell to 2.
    4. Set the wall cells to 1.
    5. Create a queue and enqueue the goal cell.

    6. While the queue is not empty:
        a. Dequeue the front cell (current cell) from the queue.
        b. For each neighbor of the current cell (4-point connectivity):
            i. If the neighbor is unvisited and not an obstacle:
                - Set neighbor's value to current cell's value + 1.
                - Enqueue the neighbor cell.

    7. Backtrack from the start cell to the goal:
        a. Initialize path list with start cell.
        b. While the current cell is not the goal cell:
            i. Select the neighboring cell with the lowest value.
            ii. Move to the selected cell and add it to the path list.

    8. Return the path list as the planned path.
```

This algorithm's complexity is manageable for grid environments and is widely used due to its straightforward implementation and reliable pathfinding capabilities. The Wavefront Planner's output

Background: Path Planning with Wave Front Planner (cont.)

path allows robots to navigate effectively by avoiding obstacles and reaching target destinations efficiently, which is essential for autonomous mobile robots operating in structured environments.

Background: Alternative Approach: Graph-Based Path Planning with Dijkstra's Algorithm

In this approach, students will represent the environment as a graph rather than using a wavefront expansion. This method is advantageous for environments that can be represented as nodes (cells) connected by edges (paths), making it flexible and adaptable to changes in the map structure. After creating a graph representation of the environment, students will implement Dijkstra's algorithm to find the shortest path between two cells.

Creating the Graph Data Structure.

- **Define Nodes**: Each cell in the grid environment is treated as a node in the graph. The position of the cell (such as cell numbers 1–25) can serve as the unique identifier for each node.
- **Define Edges**: An edge represents a direct connection between two adjacent cells that are free of obstacles. If two cells share an open boundary (no wall between them), they are connected by an edge. Each edge has a weight, which, in this case, can be set to a uniform value (e.g., 1) since all movements between adjacent cells are treated as equal.
- **Adjacency List Representation**: To represent the graph, use an adjacency list where each node (cell) has a list of neighboring nodes (cells) to which it connects. For example, if cell 1 connects to cells 2 and 6, the adjacency list entry for cell 1 would be {2: 1, 6: 1}, with each neighbor's entry containing the edge weight.

```
Example Adjacency List for a 3x3 Grid:

Cell 1: {2: 1, 4: 1}
Cell 2: {1: 1, 3: 1, 5: 1}
Cell 3: {2: 1, 6: 1}
...
```

Implementing Dijkstra's Algorithm on the Graph. Once the graph structure is complete, Dijkstra's algorithm can be implemented to determine the shortest path between a start cell and a goal cell. Dijkstra's algorithm works by exploring paths from the start node to the goal node with the shortest cumulative cost.

- **Initialization**: Set the distance to the start node as 0 and all other nodes as infinity. Use a priority queue to keep track of nodes to explore based on their distance from the start node. Keep a dictionary to record the predecessor of each node, which helps reconstruct the path once the goal is reached.
- **Algorithm Execution**:
  - While the priority queue is not empty:
    - ∗ Dequeue the node with the smallest distance (this is the current node).
    - ∗ For each neighbor of the current node:
      - · Calculate the tentative distance by adding the current node's distance to the edge weight connecting to the neighbor.
      - · If the tentative distance is less than the recorded distance for the neighbor, update the distance and set the current node as the neighbor's predecessor.
      - · Enqueue the neighbor with the updated distance.
- **Path Reconstruction**: Once the goal node is reached, backtrack from the goal to the start node using the predecessor dictionary to reconstruct the shortest path.

Background: Alternative Approach: Graph-Based Path Planning with Dijkstra's Algorithm (cont.)

High-Level Pseudo Code.

```
Dijkstra(graph, start, goal):
    1. Initialize distance for all nodes as infinity, except the start node
       (set to 0).
    2. Initialize a priority queue and enqueue the start node with distance 0.
    3. Initialize an empty dictionary for predecessors.

    4. While the priority queue is not empty:
        a. Dequeue the node with the smallest distance (current node).
        b. If current node is the goal, break the loop.
        c. For each neighbor of the current node:
            i. Calculate tentative distance = current distance + edge weight.
            ii. If tentative distance < recorded distance for neighbor:
                - Update neighbor's distance.
                - Set current node as the predecessor for the neighbor.
                - Enqueue the neighbor with the updated distance.

    5. Reconstruct path from goal to start using the predecessors dictionary.
    6. Return the reconstructed path as the shortest path.
```

This approach emphasizes creating a graph structure and implementing Dijkstra's algorithm from scratch, which is valuable for understanding data structures and algorithms in robotics pathfinding. By using this method, students will gain a deeper comprehension of graph theory and optimization, which are foundational concepts in robotic navigation and autonomous systems.

Pose Estimation and Print Statements

In all tasks, you are required to print the following information once per cell visited. **Be sure to include these print statements clearly in your videos, ensuring that they are readable on screen.**

(1) **State Pose:** The robot's pose should be printed in the form $s = (x, y, n, \theta)$, where:
- (x, y) represents the robot's position in global coordinates,
- n is the grid cell number, and
- θ is the robot's orientation relative to the global frame.

Note: You are not allowed to use GPS for this assignment.

At the start of your code, use the following steps to access the robot's initial position and orientation attributes:

(a) Access the `starting_position` attribute of the robot object.
(b) Extract the x and y coordinates, representing the robot's initial horizontal and vertical positions, respectively.
(c) Extract the θ value, which gives the initial orientation angle of the robot.

Below is an example code snippet to obtain these initial values:

```
# Move robot to a random starting position listed in the maze file
robot.move_to_start()

start_pos = robot.starting_position
x = start_pos.x
y = start_pos.y
theta = start_pos.theta
print(x, y, theta)
```

Pose Estimation and Print Statements (cont.)

Once the initial values are acquired, you must continuously update the robot's current position (x, y) and orientation (θ) as it moves. Use the encoders and compass sensors to calculate and adjust the pose over time. This logic should be embedded within the main loop, updating pose data at regular intervals.

(2) **Shortest Path:** Regardless of how you calculate the shortest path from the start to goal cells, your program should clearly print the path between the cells. For example we should something similar to what is shown below printed at some point in your video.

$$C_s \rightarrow C_{t_1} \rightarrow C_{t_{i+1}} \rightarrow \cdots \rightarrow C_G$$

Where:
- C_s is the start cell, where the path begins,
- $C_{t_1}, C_{t_{i+1}}, \ldots$ are intermediate cells along the shortest path, ordered sequentially. Each C_{t_i} represents a cell that is part of the optimal route from C_s to C_G,
- C_G is the goal cell, which is the final destination of the path.

This notation shows the path as a progression from the start cell through intermediate cells to reach the goal cell, ensuring minimal distance or cost from C_s to C_G.

Task 1: Path Planning

In this task, you will implement a solution for navigating a robot along the shortest path from a starting cell to a specified goal cell within a maze. This involves constructing a data structure for representing maze configurations, implementing a pathfinding algorithm, and ensuring that the robot accurately follows the calculated path. The requirements for Task 1 are as follows:

(1) **Maze Representation**: Implement a data structure to represent the maze, capturing details about cell connectivity and obstacles. Each cell should be able to store information about walls or open paths to neighboring cells.

(2) **Shortest Path Calculation**: Develop an algorithm to compute the shortest path between a given starting cell and a goal cell. This pathfinding algorithm should be efficient and should return a sequence of cells that the robot must traverse to reach the goal.

(3) **Path Display**: Once the shortest path is calculated, print the complete path sequence, showing each cell that the robot will pass through from the start to the goal. This should be displayed in a clear, ordered format.

(4) **Path Navigation**: Program the robot to navigate along the calculated path. The robot should move sequentially from the starting cell through each cell in the path until it reaches the goal.

(5) **Pose Estimation Printing**: Each time the robot enters a new cell, print the robot's current pose estimate, which includes its position and orientation in the maze. This estimate should update as the robot progresses along the path.

(6) **Task Completion**: The robot should stop once it reaches the goal cell. At this point, the robot should print a final message indicating that it has reached its destination.

(7) **Video Requirements**: For the video submission, test your algorithm by setting **ten** unique starting locations, with the goal cell set to cell 7 in each test. Use the maze configuration provided in `maze8.xml` for this task. The video should demonstrate the algorithm calculating the shortest path from each of the **ten** starting cells to the goal and then show the robot navigating along the calculated path. Ensure that each example starts from a different cell and that the robot correctly follows the path to reach the goal. For each starting location demonstrated in the video, include the calculated path in your report to provide a complete record of the algorithm's output and path execution.

Task 1: Path Planning (cont.)

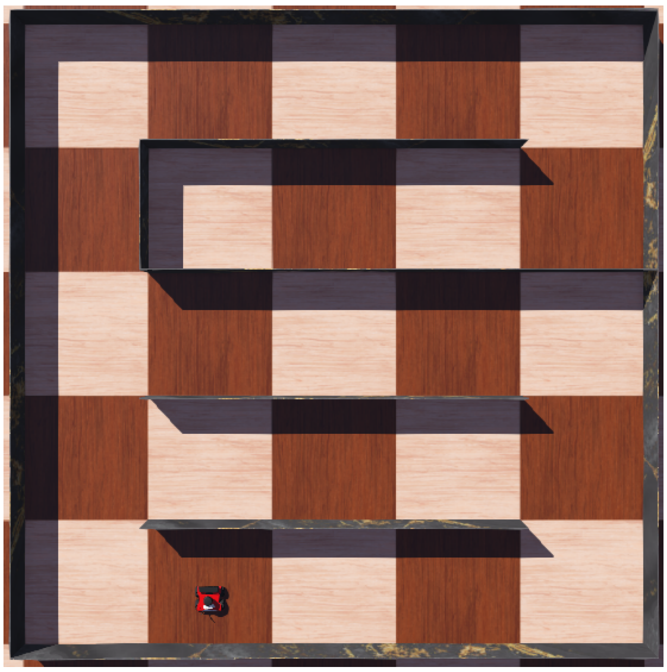


FIGURE 2. Maze layout for Task 1 (maze file *maze8.xml*).

Extra Credit Opportunity (10 points)

The student who achieves the shortest navigation time from cell 25 to cell 7 will be awarded 10 points of extra credit. To qualify, the student must include a video showing the robot navigating the shortest path from cell 25 to cell 7. Additionally, the report must display the calculated path and state the total simulation time taken to reach the goal.

Code Evaluation Task 1 (90%)

- **Maze Representation (15 Points)**
  - **Full Marks (15 points)**: Maze data structure is complete, accurate, and includes all required details about cell connectivity and obstacles.
  - **Acceptable (10 points)**: Maze data structure is mostly accurate, with minor errors or omissions that do not significantly impact functionality.
  - **Poor (5 points)**: Maze data structure is incomplete or has major errors, impacting the algorithm's ability to calculate paths correctly.
  - **No Marks (0 points)**: No data structure implemented for maze representation.
- **Shortest Path Calculation (20 Points)**
  - **Full Marks (20 points)**: Algorithm calculates the shortest path from any starting cell to the goal cell accurately and efficiently.
  - **Acceptable (15 points)**: Algorithm calculates the shortest path with minor inefficiencies or small errors that don't prevent completion of the task.
  - **Poor (10 points)**: Algorithm calculates a path, but it is not consistently the shortest, or contains significant errors.
  - **No Marks (0 points)**: No algorithm implemented to calculate the shortest path.
- **Path Display (10 Points)**
  - **Full Marks (10 points)**: The calculated path is clearly printed in sequence, showing each cell from the start to the goal.
  - **Acceptable (7 points)**: Path is printed, but with minor formatting or sequence errors.

Task 1: Motion to Goal (cont.)

- **Poor (3 points)**: Path is printed, but it is unclear or significantly incorrect.
- **No Marks (0 points)**: No path display provided.

- **Path Navigation (25 Points)**
  - **Full Marks (25 points)**: Robot follows the calculated shortest path accurately from start to goal without deviations.
  - **Acceptable (18 points)**: Robot follows the path with minor deviations or inconsistencies.
  - **Poor (10 points)**: Robot attempts to follow the path but frequently deviates or does not complete it.
  - **No Marks (0 points)**: No path navigation implemented or path not followed at all.
- **Pose Estimation Printing (10 Points)**
  - **Full Marks (10 points)**: Pose (position and orientation) is printed accurately each time the robot enters a new cell.
  - **Acceptable (7 points)**: Pose is printed but with minor inaccuracies or occasional omissions.
  - **Poor (3 points)**: Pose is printed inconsistently or inaccurately.
  - **No Marks (0 points)**: No pose estimation printed.
- **Task Completion (10 Points)**
  - **Full Marks (10 points)**: Robot stops correctly upon reaching the goal cell and prints a final message indicating completion.
  - **Acceptable (7 points)**: Robot stops at the goal cell but final message is unclear or missing.
  - **Poor (3 points)**: Robot stops at the goal cell inconsistently or with significant delays.
  - **No Marks (0 points)**: Robot does not stop at the goal cell or no indication of task completion.

Deductions

- **Print Statement Visibility (-20 points maximum):** Up to 20 points may be deducted if required print statements are not clearly visible in the submitted video. The severity of the deduction will depend on the clarity and frequency of missing print statements.
- **Not Showing Multiple Starting Locations (-10 points):** A 10-point deduction will be applied if the video for Task 1 does not show multiple starting locations.
- **Robot Collisions with Wall (-5 points):** A 5-point deduction will be applied each time the robot collides with a wall, up to a maximum of 5 points.

Report & Video Evaluation (10%)

A project submitted without a report, or without including the corresponding task video, will not be graded and will receive a "0" for the complete lab.

Report Sections: The report should include the following (points will be deducted if anything is missing):

- **Mathematical Computations:** Show how you calculated the speeds of the left and right servos given the input parameters for each task.
- **Conclusions:** Analyze any issues encountered when running the tasks and discuss how these could be improved. Discuss your results, including any navigation errors. Conclusions need to reflect an insight into what was learned during the project. Statements like "everything worked as expected" or "I enjoyed the project" will not count as valid conclusions.
- **Video:** Upload the video to Canvas showing the robot executing the task (details below).
- **Authorship Statement:** Include and sign the following statement at the end of the report:

  "I developed all the code and written report with the following exceptions:

  [Student name: signature, date]

Report & Video Evaluation (cont.)

[Code sections (specify lines): references and additional comments]
If any external tools were used to generate code, you need to include the prompt and output."

Video Requirement for Task 1

(1) **Demonstrations from Multiple Starting Locations**
- Show at least ten unique starting positions in the maze.
- For each demonstration:
  - Begin by displaying the initial starting cell and the goal cell (cell 7).
  - Show the calculated shortest path from the starting cell to the goal cell.

(2) **Robot Navigation for Each Starting Position**
- For each starting cell, allow the video to show the robot navigating along the calculated shortest path.
- Highlight:
  - Each time the robot enters a new cell, with an emphasis on the pose estimation (position and orientation) printed for each cell transition.
  - The robot's ability to accurately follow the path without deviating.
- Conclude each demonstration when the robot reaches the goal cell and prints the completion message.

Lab Submission

For Lab 5 there are two separate Canvas Assignments: Lab 5 Code Submission and Lab 5 Report Submission. Please follow the Submission policy outlined below. **Note:** Failure to adhere to the submission policy will result in a reduction of 20 points.

Code Submission Policy

Students will need to upload a Python file (i.e. Webots controller) for each Task. The Python file should follow this naming convention *USF_ID_LabX_TaskY.py*, where *X* is the lab number and *Y* is the task number. Example *rockybull5_Lab1_Task1.py*. Additionally, if a student has created any functions that are not present in the controller but rather the *MyRobot.py*, the student will also need to upload that python file as well using the following naming convention *USF_ID_MyRobot.py*.

If custom functions are used in a student's submission and they are not in the controller or the MyRobot Python files, an automatic Zero will be assigned for the code submission.

Each file should be uploaded to the same submission. **DO NOT** upload a zip file, Canvas allows multiple uploads per submission. Failure to meet these submission policies will result in 20 points off.

Report Submission Policy

Students will need to upload their videos to their USF OneDrive or USF Sharepoint and generate a shareable link. Please ensure that you make it viewable to everyone or at the very least the TA and the Professor (use their emails to add them to the access list).

Students will need to upload a PDF file of their lab report. Be sure that the lab report contains the link to the videos and any Metrics requested by the Lab. These Metrics will be used to determine which student receives the extra credit. Use the naming convention *USF_ID_LabX_Report.pdf*. Only PDFs will be accepted.

If the student used ChatGPT to generate code they will also need to provide the conversation used to generate the code. This can either consist of screenshots of the conversation or a shareable link that the system provides. This should be a separate document and not part of the report.

Each file should be uploaded to the same submission. **DO NOT** upload a zip file, Canvas allows multiple uploads per submission. Failure to meet these submission policies will result in 20 points off.

TA may ask you additional questions to gauge your understanding via Canvas Message or MS Teams. Failure to reply may result in point deduction.