MINOR PROJECT

Title: Robot Navigation Strategy for a Simple Maze using DFS

BY ADITYA KONWAR

INTRODUCTION

In robotics and artificial intelligence, navigation is considered one of the most fundamental and challenging problems. For a robot to operate effectively in a real-world or simulated environment, it must be capable of moving from a starting point to a desired goal while successfully avoiding obstacles. This ability forms the foundation for more complex robotic tasks such as autonomous driving, warehouse automation, rescue operations, and planetary exploration.

Mazes provide an excellent platform for studying and experimenting with robot navigation strategies because they clearly define two elements: pathways (open cells) that represent feasible routes and walls (blocked cells) that represent obstacles. By solving maze problems, we can gain insights into how robots make decisions, explore environments, and select optimal or feasible paths under different constraints.

In this project, we focus on designing a robot navigation strategy to solve a simple maze problem using the Depth-First Search (DFS) algorithm. DFS is a fundamental graph traversal technique that systematically explores paths in a depth-oriented manner until a solution is found or all possibilities are exhausted. Although DFS does not always guarantee the shortest path, it provides a valid solution and serves as a stepping stone toward more advanced algorithms.

The project demonstrates key aspects of robot navigation, including maze representation using grids, implementation of DFS, path reconstruction, and visualization of the optimal path from the start to the goal. Additionally, it emphasizes how algorithms can be adapted from computer science theory to practical applications in robotics. This work lays the groundwork for exploring more advanced pathfinding algorithms such as Breadth-First Search (BFS), Dijkstra's Algorithm, and A* Search in future developments.

OBJECTIVE

The objective of this project is to:

- Implement a navigation strategy that allows a robot to find a valid path in a maze.
- Apply the **Depth-First Search (DFS)** algorithm to explore possible paths systematically.
- Reconstruct and visualize the path taken from the **start position** to the **goal position**.
- Provide a foundation for more advanced pathfinding algorithms such as **BFS, Dijkstra, and A***.
- Demonstrate the practical application of algorithmic problem-solving techniques in robotics and artificial intelligence.

METHODOLOGY

The methodology adopted for this project involves a systematic approach to solving the maze navigation problem using the **Depth-First Search (DFS)** algorithm. The steps are outlined below:

1. Maze Representation

- The maze is represented as a **2D grid (matrix)** where each element indicates whether the cell is accessible or blocked:
    - 0 → Open cell (pathway).
    - 1 → Blocked cell (wall or obstacle).
- This representation simplifies the maze into a computationally manageable structure where robot navigation can be modeled as movement across grid cells.

Example:

```
maze = [
  [0, 0, 1, 0, 0],
  [0, 0, 0, 0, 0],
  [0, 1, 1, 1, 0],
  [0, 1, 0, 0, 0],
  [0, 0, 0, 1, 0]
]
```

- The **start position** is defined as (0,0) (top-left corner) and the **goal position** as (4,4) (bottom-right corner).

2. Algorithm Selection – Depth-First Search (DFS)

- **Why DFS?**
    - DFS is one of the simplest and most fundamental graph traversal algorithms.
    - It explores paths **deeply** before backtracking, making it suitable for finding a valid path even in complex mazes.
    - Although it does not guarantee the shortest path, it ensures that if a path exists, it will be discovered.
- DFS is implemented using a **stack** (either explicitly or via recursion) to manage the exploration order.

3. Neighbor Identification

- From any given cell, the robot can move to one of its four adjacent cells: **up, down, left, or right**.
- A helper function get_neighbors() is used to:
    - Check the validity of each move (staying inside maze boundaries).
    - Ensure the next cell is not a wall.
    - Return only feasible moves for further exploration.

4. DFS Implementation

- The algorithm starts at the **start position** and proceeds as follows:
    1. Push the starting cell into a **stack**.
    2. Mark the cell as **visited** to avoid reprocessing.
    3. Repeatedly pop cells from the stack and explore their valid neighbors.
    4. Keep track of the **parent relationship** for each visited cell so that the path can be reconstructed after reaching the goal.
    5. Stop once the **goal cell** is reached.

5. Path Reconstruction

- Once the goal is found, the path is reconstructed by backtracking from the goal to the start using the **parent dictionary**.
- This ensures that the robot can determine the exact sequence of steps needed to traverse from the start to the goal.

6. Visualization

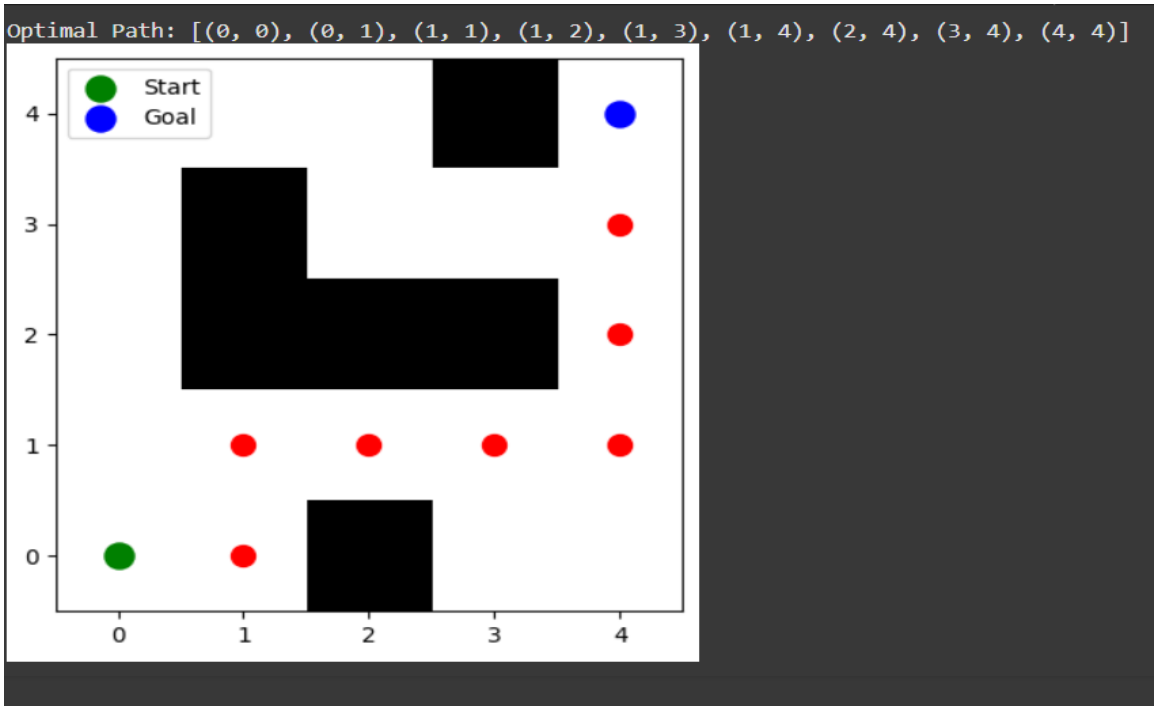
- **Matplotlib** is used to visualize both the maze and the path found by DFS.
- Key elements in the visualization:
    - **Black cells** → Walls.
    - **White cells** → Open paths.
    - **Green dot** → Start position.
    - **Blue dot** → Goal position.
    - **Red dots** → Path traversed by the robot.

- An **animated version** is also implemented to simulate step-by-step robot movement through the maze, making the navigation process more intuitive and engaging.

7. Extensions

- The methodology can be extended to:
    - Use **Breadth-First Search (BFS)** for shortest path solutions.
    - Apply **A*** search for heuristic-based efficient navigation.
    - Handle **dynamic mazes** with moving obstacles.

RESULTS AND DISCUSSION



The algorithm successfully found a path from the **start (0,0)** to the **goal (4,4)**.

- **Output Path:**

[(0, 0), (0, 1), (1, 1), (1, 2), (1, 3), (1, 4),

(2, 4), (3, 4), (4, 4)]

- **Visualization:**
  - The maze was displayed with walls (black), free paths (white), start (green), goal (blue), and the robot's path (red).
  - The animated version showed the robot moving step by step through the maze.

- **Key Observations:**
- DFS is simple and effective for smaller mazes.
- It may not yield the shortest path, unlike BFS or A*.
- The project can be extended with more advanced algorithms for complex navigation tasks.

CONCLUSION

This project successfully demonstrated a robot navigation strategy using the Depth-First Search (DFS) algorithm for maze traversal. By representing the maze as a grid, implementing DFS for systematic exploration, and reconstructing the path from start to goal, the robot was able to navigate the environment effectively. The addition of a step-by-step animation further enhanced the understanding of the search process, making the results more interactive and visually intuitive.

The project not only achieved its primary goal of finding a valid path but also highlighted the importance of algorithmic approaches in solving navigation problems in robotics and artificial intelligence. While DFS provides a simple and effective solution, it also revealed certain limitations, such as not always guaranteeing the shortest path. This opens the door for further exploration of more advanced algorithms.

Future Scope:

- Implement **Breadth-First Search (BFS)** to guarantee the shortest path in unweighted mazes.
- Extend to **A*** search by incorporating heuristics for more efficient navigation.
- Adapt the system for **dynamic mazes** where obstacles may change or move during execution.
- Explore real-world applications, such as integrating the algorithm into **autonomous robots** or **simulated agents** in virtual environments.