Simulation of an Autonomous Mop Robot

ID	Team member	Contribution
2023557	احمد جمال شعلان	BFS implementation and analysis
2023234	مؤمن احمد نوح فرغلي	BFS implementation and analysis
2023584	حبيبة حسن بدير علي حسام الدين	DFS implementation and analysis
2023073	حنين عبدالحافظ عبدالمجيد	DFS implementation and analysis
2023036	اسراء محمد ابراهيم سالم الخولى	A* implementation and analysis, report writing, github management
2023171	محمد حسنى ابو مصطفى	A* implementation and analysis

Project Background

Cleaning robots are becoming widely useful for their ability to do repetitive tasks without human intervention. They operate in an environment that changes constantly, where they need to achieve full floor coverage while avoiding obstacles like furniture, walls and narrow routes. Which means that pathfinding search algorithms are critical to cover the full floor with the least time and energy consumption.

Project Description

This project models the function of an autonomous mop robot that cleans floors using minimal human effort by navigating the environment represented as a grid while avoiding obstacles. It uses AI search algorithms to navigate its way efficiently by constantly making decisions about the next cell to move to based on the algorithms used. The project demonstrates a robot's behavior, analyzes how each of the search algorithms changes that behavior, and provides a visual analysis of the cleaning process.

Project Objective

Implementing search algorithms in a robot mop simulation by doing the following:

- Explanation of the working mechanisms of BFS, DFS, A* search algorithms
- Visualization of the cleaning process

- A comparison of the metrics: time complexity, space complexity, path cost, solution optimality.
- Analysis of the differences between the used search algorithms

Robot Logic and Workflow

Grid setup:

- 25 × 25 grid.
- 0 = free cell (dirty)
- 1 = blocked cell (obstacle)

Robot Movement Rules:

- It moves up, down, left, and right.
- Moves within the grid boundaries
- Does not enter the cells represented as obstacles
- Enters free cells in a way determined by the used search algorithm

Process Workflow:

A cycle of the following steps continues until all available free cells are cleaned:

1. The robot identifies its current position in the grid and checks the surrounding cells to find which cell is free, cleaned, or an obstacle.
2. It selects the next cell according to the used search algorithm aiming for the most optimal path.
3. It moves to the chosen cell and avoids obstacles following the allowed movements (up, down, left, right)
4. The free cell is marked as cleaned after the robot has entered it which makes sure that the already-cleaned cells are not entered again unless necessary.

Technologies Used

- Programming language: Python
- IDE: Visual Studio Code
- Code hosting platform: GitHub

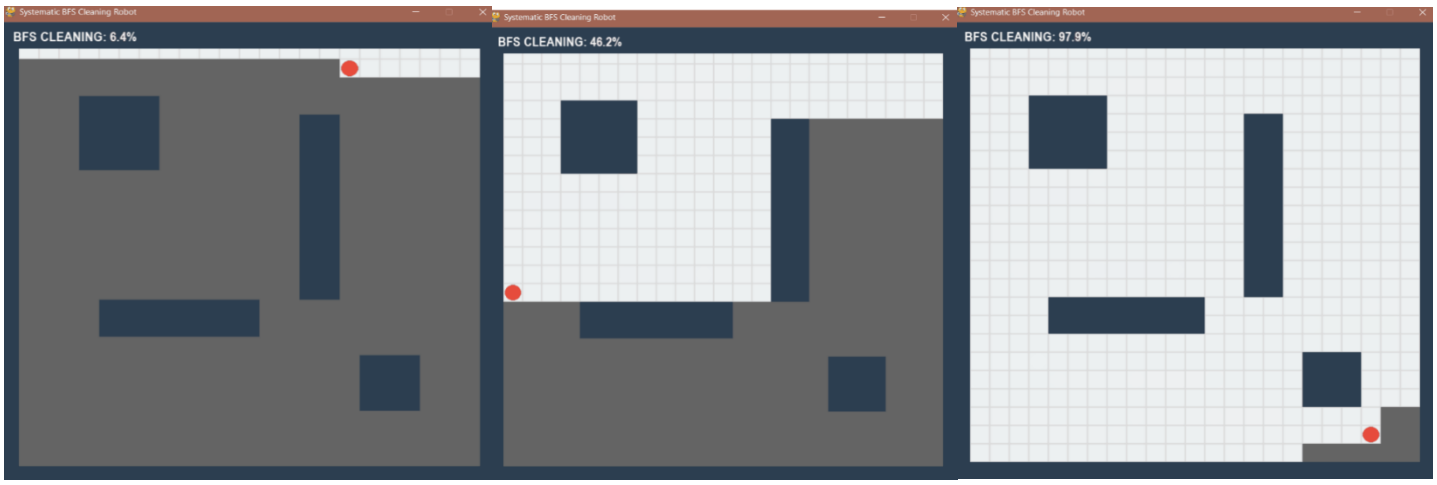
Algorithms Used

Breadth-First Search (BFS)

BFS is a level-based uninformed search algorithm that explores the environment level by level using a queue to store all free (uncleaned) cells. BFS enters a cell, dequeues it, marks it as entered (cleaned) and enqueues all the neighboring free (uncleaned) cells. It repeats the process until the queue is empty (the floor is completely cleaned). The goal is to find the shortest path to the first neighboring cell.

Pros	Cons
Finds the shortest path to the first neighboring cell	High use of memory for needing to store all the current level cells
The algorithm is complete	Slow to implement for larger environments
The level-by-level approach prevents the robot from being trapped in a long path like DFS.	
Simple to implement	

Output Demonstration

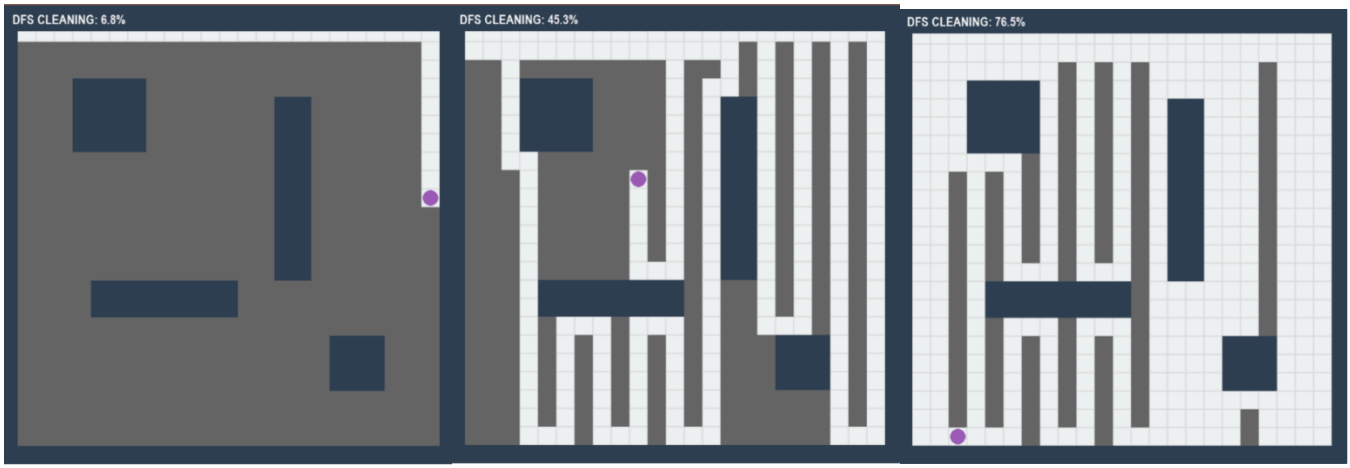


Depth First Algorithm (DFS)

DFS is also an uninformed search algorithm that stores the cells of one path in a stack and explores until the maximum depth before going back to another. Which means that the robot keeps moving in one direction cleaning the cells in the way until it reaches a dead end or an obstacle then goes back the closest free cell and repeats the process.

Pros	Cons
Uses less memory than BFS because it only stores the cells of the current path	Does not guarantee finding the shortest path unlike BFS
Faster than BFS for larger environments as it explores until the maximum depth of one path	Can get stuck in a single path
Simple to implement	

Output Demonstration



A* Search Algorithm

A* is an informed search algorithm that selects the nearest free (uncleaned) cell as the target and finds the most efficient path between it and the robot's current position, after cleaning it the algorithm is applied again to the next nearest cell until the whole floor is completely cleaned using the cost function: $f(n) = g(n) + h(n)$ where:

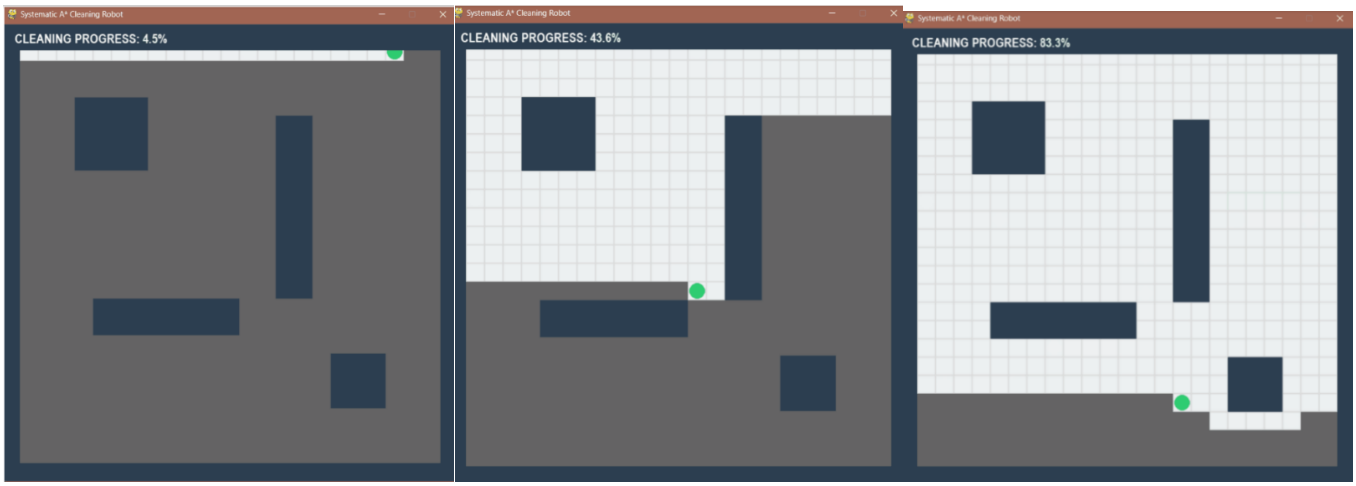
- $g(n)$ is the cost (distance) from start to the current cell
- $h(n)$ is the heuristic function that is an estimate of the cost from current node cell to the target
- $f(n)$ the total cost of the path

A* uses open and closed lists which are priority queues:

- All the neighboring cells are pushed into the open list.
- The cells that have been visited are stored in the closed list.
- Blocked cells are ignored.
- The cell with the least $f(n)$ is popped from the list and the robot moves to it next.

Pros	Cons
Optimal as it finds the shortest most efficient path to the target	High use of memory due to using priority queues (open list and closed list)
The algorithm is complete	The efficiency depends on the heuristic function
Models real-world robotics	More complex to implement

Output Demonstration



Performance Metrics

Where:

- b is the maximum number of neighboring cells
- d is the minimum number of moves to reach a target cell
- m is the maximum number of moves taken in a single path

Metrics	**BFS**	**DFS**	**A***
Time Complexity	$O(b^d)$	$O(b^m)$	$O(b^d)$ - $O(n)$
Space Complexity	$O(b^d)$	$O(bm)$	$O(b^d)$ - $O(n)$
Path Cost	Minimum path cost for uniform-cost environment	Does not guarantee minimum path cost as it explores deep path without considering their length	$f(n) = g(n) + h(n)$
Solution Optimality	Optimal for uniform-cost environment	Not optimal cost as it explores deep path without considering their length	Optimal as it uses admissible heuristic (never overestimates the distance to the goal, always $\leq$ the minimum remaining cost

Conclusions and Limitations

Project Conclusions

- **Breadth-First Search Algorithm** finds the shortest path to every target cell ensuring optimality. But it also needs high memory usage to store all the current level cells
- **Depth-First Search Algorithm** though uses less memory but takes longer, less efficient path which leads to higher path cost and unrealistic robot movement. Therefore, it doesn't find the optimal solution.
- **A* Search Algorithm** uses the cost function $f(n) = g(n) + h(n)$ which allows the robot to reach each target efficiently with minimal total steps using admissible heuristic and actual path cost. It finds the optimal paths and reduces unnecessary exploration compared to BFS.

The general conclusion is that A* is the most efficient search algorithm for a realistic autonomous robot mop movement.

Project Limitations

- **The environment is simplified:** the robot operates in a 2d grid where the cells are uniform which is different from real-world scenarios where movement cost varies based on surface type (wood, tile, carpet, etc.) or small obstacles that might not be detected.
- **Unincorporated physical constraints:** factors like friction between the robot and the floor, waterflow from the mop, and motor behavior all affect the speed and cleaning ability of the robot.
- **Non-moving obstacles:** moving obstacles like people or pets are not considered.