CSE 211 Term Project

Project 8: Autonomous Vacuum Cleaner Multi-Room Coverage Planner

Ege Huriel
Pelin Görer
Emirhan Eren Elibol
Eylül Pirinçal
Selin Suna Kaya

Contents

Project Overview

Introduction

This project is developed as part of the **CSE 211 Term Project** and focuses on designing and implementing a grid-based autonomous vacuum cleaner simulation. The main objective is to model a vacuum cleaner that can autonomously navigate an indoor environment, clean all reachable floor areas, avoid obstacles, manage its battery intelligently and safely return to a charging dock when required.

The environment is represented as 2D grid loaded from a JSON input file. Each cell in the grid corresponds to specific type such as wall, floor, obstacle, door or charging dock. The vacuum cleaner operates under strict movement and battery constraints and must determine valid paths while respecting these environmental rules.

A key requirement of the project is the prohibition of STL containers for core data handling. Instead all fundamental data structures such as Linked List, Stacks, Queues and Hash Sets are implemented manually. This constraint emphasizes a deep understanding of data structures, memory management and algorithmic design.

The project is architected in a modular and layered manner, separating concerns across parsing, gird modeling, pathfinding, room decomposition, coverage planning, battery management and user interaction. Each module has a clearly defined responsibility, enabling both maintainability and extensibility.

From a functional perspective, the system performs the following high-level tasks:

1. Parses and validates grid-based input data from JSON files.
2. Identifiers rooms by decomposing the grid using door boundaries.
3. Plans efficient cleaning paths that ensure full coverage of reachable floor cells.
4. Monitors battery levels and enforces a safety margin to guarantee a safe return to the charging dock.
5. Monitors battery levels and enforces a safety margin to guarantee a safe return to the charging dock.
6. Visualizes the cleaning process step by step or in fact execution mode using a text based user interface.

System Overview

The autonomous vacuum cleaner system is designed as modular software architecture that simulates intelligent cleaning behavior within a grid based indoor environment. The system integrates environment modeling, custom data structures, path planning algorithms, battery aware decision making and user interaction into a cohesive workflow.

At a high level, the system operates by loading a grid based representation of the environment from an input file, initializing internal data structures and executing a coverage planning algorithm that ensures all reachable floor cells are cleaned while respecting battery constraints. Throughout execution the system continuously evaluates its current state including position, remaining position, remains battery and unclean areas to make safe and optimal navigation decisions.

High Level Architecture

The architecture follows a layered and responsibility driven design. Core logic components are isolated from input handling and visualization layers to improve maintainability and clarity. The main architectural layers are:

Input and Parsing Layer: Responsible for reading, validating and transforming JSON input files into internal grid representations.

Environment Modeling Layer: Represent the grid, cell types and spatial relationships between cells.

Planning and Decision Layer: Handles coverage path planning, room decomposition, shortest path computation and battery aware navigation.

Execution and Control Layer: Simulates vacuum movement, battery consumption, cleaning actions and docking behavior.

Visualizing Layer: Displays the state of the environment and the vacuum cleaner using a text based user interface.

Main Components

The system Is composed of the following main components:

Grid Model: Stores the 2D grid structure and cell metadata such as walls, floors, obstacles, door and the charging dock.

Vacuum Controler: Maintains the current state of the vacuum cleaner, including position, battery level and cleaning status.

Coverage Planner: Determines the order and direction in which floor cells are cleaned to ensure full coverage.

Room Decomposition Module: Identifies distinct rooms by analyzing door connections and spatial boundaries with the grid.

Pathfinding Module: Computes shortest paths between locations, particularly for safe returns to the charging dock.

Battery Manager: Enforces battery constraints and safety margins to prevent the vacuum from becoming stranded.

User Interface Module: Provides step by step or fast execution visualization of the cleaning process.

Assumptions and Constraints

The system is developed under several explicit assumptions and constraints:

1. The environment is static, obstacles and walls do not change during execution.
2. All reachable floor cells must be cleaned at least once.
3. The vacuum cleaner can move only to adjacent cells and consumes battery per movement.
4. At any time, the remaining battery must be sufficient to reach the charging dock.
5. STL containers are not used for core data storage, all primary data structures are implemented manually using pointer based designs.

These constraints directly influence architectural decisions, data structure choices and algorithmic strategies used throughout the system.

Problem Analysis

The autonomous vacuum cleaner coverage problem can be classified as a constrained coverage path planning problem in a discrete environment. Unlike classical shortest path problems, the primary goal is not reaching a single destination but ensuring that all reachable floor cells are visited at least once while respecting environmental and battery related constraints.

Environment Representation

The indoor environment is represented as rectangle 2D grid loaded from a JSON input. Each cell in the grid corresponds to one of the predefined cell types:

Wall: Impassable boundaries that define the limits of rooms.
Floor: Traversable and cleanable cells.
Obstacle: Non traversable interior elements such as furniture.
Door: Passable cells that connect different rooms.
Charging Dock: A special traversable cell where the vacuum can recharge.

The grid based abstraction simplifies spatial reasoning and allows neighborhood based movement rules while still capturing the complexity of multi room environments.

Movement and Traversal Rules

The vacuum cleaner moves discreetly between adjacent cells. Movement is restricted to valid traversable cells (floor, door and charging dock). Walls and obstacles cannot be entered.

Each movement consumes a fixed unit of battery energy. Cleaning a cell is implicitly performed when the vacuum enters a floor cell, after which the cell is marked as cleaned. The system must track visited and unvisited floor cells throughout execution to guarantee complete coverage.

Coverage Requirements

The core requirement of the problem is complete coverage.

1. Every reachable floor cell must be visited at least once.
2. No cell should be skipped due to inefficient traversal or poor planning.
3. The system must correctly handle complex layouts with obstacles.

Unlike shortest path problems, coverage planning must balance efficiency with completeness, often revisiting already cleaned cells to access uncleaned regions.

Battery Constraints and Safety Margin

Battery capacity introduces a critical constraint that significantly increases problem complexity. At any point during execution, the vacuum cleaner must retain sufficient battery to safely return to the charging dock.

This is enforced using a **safety margin**, defined: remaining battery ≥ distance to dock

Before entering a new region or continuing coverage, the system evaluates whether the remaining battery is sufficient not only for cleaning actions but also for retuning to the dock using the shortest available path. If this condition is violated, the vacuum is forced to abort coverage temporarily and return to the charging dock.

Multi Room Structure and Doorways

The environment may consist of multiple rooms connected via door cells. Doorways often form narrow passages that can easily trap the vacuum if battery constraints are ignored.

To address this the grid is decomposed into logical rooms based in door boundaries. Cleaning is typically completed room by room to reduce the unnecessary traversal between distant regions and to simplify battery feasibility checks.

Data Structure Design

A central requirement of the project is the explicit prohibition of STL containers for core data handling. As a result, all fundamental data structures are implemented manually using pointer based designs. This section describes the custom data structures used in the system and the rationale behind each choice.

The selected data structures are designed to support efficient grid traversal, coverage tracking, pathfinding and battery aware decision making while maintaining clear ownership and memory safety.

Overview of Custom Data Structures

The system relies on multiple custom built data structures, each tailored to specific responsiblity within the application. The primary data structures include:

1. Linked List
2. Stack and Queue implementations
3. Hash based structures for fast lookup
4. Graph like adjacency representations derived from the grid

All structures are implemented without using STL containers and are based on dynamically allocated nodes connected through pointers.

Grid Representation Structure

The environment grid is stored using a 2D structure composed of linked nodes rather than array based containers. Each grid cell is represented by a node that contains:

1. Cell Type (wall, floor, obstacle, door, charging dock)
2. Grid Coordinates
3. Cleaning Status

This representation enables direct traversal between adjacent cells and supports neighborhood based algorithms without relying on index based access.

Graph and Connectivity Representation

For pathfinding and reachability analysis, the grid is conceptually treated as graph. Each traversable cell corresponds to a vertex and valid movements between adjacent cells form edges.

Instead of using STL based adjacency list, connectivity is modeled through pointer based neighbor relationships. The approach eliminates the need for external graph containers while still enabling shortest path computations and connectivity checks.

Traversal algorithms operate directly on these linked relationships, ensuring consisteny between the grid model and the navigation logic.

Auxiliary Data Structures

Several auxiliary data structures are implemented to support system operations:

Queue: Used in for shortest path calculations, particularly when computing safe return paths to the charging dock.

Stack: Used for depth first traversal and backtracking during coverage planning.

Linked List: Used to maintain collections such as uncleaned cells, planned movement sequences and room membership lists.

Hash Set: Used to efficiently track visited or cleaned cells and prevent redundant processing.

Memory Management Considerations

All dynamically allocated nodes are managed explicitly. Constructors and destructors are carefully designed to prevent memory leak, unused pointers.

Ownership of nodes is clearly defined at the module level. When data structures are destroyed or reset, all associated memory is properly released. This approach aligns with best practices in low level data structure implementation and reinforces the educational goals of the project.

Algorithm Design

The algorithmic core of the system combines coverage path planning with battery constraints navigation in a grid based environment. Unlike classical navigation problems that focus on reaching a single target, this project requires strategy that makes full coverage while ensuring that the vacuum cleaner can return to the charging dock.

To achieve this, the system integrates coverage planning, room based decomposition shortest path computation and battery aware decision making into a unified control loop.

Coverage Path Planning Strategy

The primary objective of the coverage algorithm is to visit every reachable floor cell at least once. The planner prioritizes systematic exploration rather than purely optimal shortest routes as coverage inherently requires already cleaned cells.

The algorithm maintains a global recored of cleaned and uncleaned cells. At each step, the vacuum selects a valid neighboring cell based on coverage priorities favoring uncleaned floor cells whenever possible. When no uncleaned neighbor exists the system backtracks through previously visited cells to reach remaining unclean regions.

This approach ensures completeness while avoiding unnecessary randomness in movement.

Battery Aware Navigation Logic

Battery management is deeply integrated into the coverage algorithm. Before moving into a new cell or continuing coverage within a room the system evaluates whether the remaining battery is sufficient to :

1. Perform the next movement and cleaning action
2. Return to the charging dock through the shortest route.

System Workflow

The system operates through a well defined execution workflow that coordinates environment initializing, coverage planning and battery. The workflow ensures that all system components operate in a synchronized manner while preserving safety and coverage.

Initialization Phase

Execution begins with loading the input JSON file. The grid is parsed and validated to ensure structure, valid cell types and the presence of exactly one charging dock.

Following the validation the grid model is constructed custom data structure are initialized and the vacuum cleaner is placed at its starting position. Internal state variable such as battery level, cleaned cell records and room are reset.

Cleaning Phase

After initialization, the system enters the main cleaning loop. During this phase, the vacuum cleaner selects its next movement based on coverage priorities.

If adjacent uncleaned floor cells are available, the vacuum advances into those cells and marks them as cleaned. If no such cells exists, the system performs controlled backtracking to reach remaining uncleaned areas.

Battery Monitoring and Decision Making

Battery level is evaluated at every moment step. Before committing to a movement the system computes the shortest path from the current position to the charging dock.

If the remaining battery is insufficient to complete the next action and safely return to the dock, the system immediately suspends coverage and switches to return mode.

Implementation Details

The project is implemented using a modular and structured approach that aligns with software engineering best practices and the constraints defined in the project specification. All components are organized to promote readability, maintainability and ease of testing.

Project Structure

The project is organized into logically separated directories:

Include/ : Contains header files (.h, .hpp) defining data structures, interfaces and core classes.

src/ : Contains source files implementing system logic and the main execution flow.

data/ : Stores JSON input files, rooms.

tests/ : Contains unit and integration tests for data structures and algorithms.

docs/ : Includes project report and additional documents.

Makefile: Defines build, run, test and clean targets.

Memory Management Strategy

All core data structures are implemented using dynamic memory allocation and pointer based node representations. Memory ownerships rules are clearly defined within each module.

Whenever nodes or structures are no longer needed, associated memory is explicitly released to prevent leaks. Constructors and destructors are carefully implemented to ensure proper cleanup during normal execution and early termination situations.

The explicit memory management approach reinforces understanding of low level data structure implementation and compiles with project requirements.

Testing

Comprehensive testing is a critical part of the project to ensure correctness and compliance with project requirements. The testing strategy focuses on validating individual data structures, verifying algorithm correctness and ensuring safe behavior under battery situations.

Unit Testing

Unit tests are written for all custom data structures implemented in system. Each data structure is tested independently to verify its correctness under various conditions.

Unit test covers:

1) Creation and initialization of structures.
2) Insertion and removal operations.
3) Traversal and access correctness.
4) Traversal and access correctness.
5) Boundary conditions such as empty and single element cases.

Integration Testing

Integration tests validate the interaction between multiple system modules. These tests focus on scenarios where good modeling, pathfinding, coverage planning and battery management operate together.

Typical integration test cases include:

1) Complete cleaning of a single room
2) Enviroments connected by doors
3) Environments with obstacles and narrow passages

Battery Safety Testing

Battery specific tests ensure that the vacuum cleaner never enters an unsafe state.

Test cases verify that:

1) The vacuum always returns to the charging dock before battery depletion
2) Coverage is paused when battery levels fail below the safety.

Team Contributions and Responsibilities

This project was developed collaboratively, with each team member taking ownership of a specific subsystem. Responsibilities were clearly divided to ensure modular development, parallel progress and seamless integration. Below is a detailed breakdown of individual contributions.

Data Structure Owner (Custom Data Structures):

Ege Huriel

Responsible for the design and implementation of all core custom data structures used throughout the project. All structures were implemented without using STL containers and strictly followed pointer based designs.

Responsibilities:

1. Implementation of Linked List. (Node based)
2. Implementation of Queue for algorithms
3. Implementation of Stack for path tracking and coverage backtracking.
4. Implementation of Hash Set using chaining for tracking cleaned cells.
5. Writing unit tests for all custom data structures.

All implementations comply with project constraints prohibiting STL containers and emphasize manual memory management.

Input and Environment Modeling Owner (Parser + GridModel)

Eylül Pirinçal

Responsible for handling all input related functionality and modeling the grid based environment.

Responsibilities:

1. JSON input parsing (grid, legend, vacuum configuration)
2. Implementation of Grid, Cell and Position model classes.
3. Input validation including: Ensuring exactly one charging dock exists, Validating start position, Verifying grid rectangularity
4. Creation of multiple input scenarios under the data/ directory

Pathfinding and Battery Management Owner

Selin Suna Kaya

Responsible for guaranteeing safe navigation under battery constraints and implementing all return to dock logic.

Responsibilities:

1. Shortest path computation to the charging dock using BFS on an unweighted grid.
2. Implementation of *distanceToDock(position)* calculation
3. Enforcement of battery safety rules using a safety margin check at each step
4. Design and implementation of the battery state machine

This component ensures the vacuum never enters an unrecoverable state.

Coverage Planning and Room Strategy Owner

Pelin Görer

Responsible for the coverage logic that guarantees all reachable floor cells are cleaned.

Responsible:

1. Room decomposition using flood based on door connectivity.
2. Determining room cleaning order using simple measurement.
3. Implementation of a coverage pattern.
4. Adapting coverage behavior around obstacles and irregular room shapes.
5. Safety margin for battery

User Interface, Integration

Emirhan Eren Elibol

Responsible for visualization (TUI), integration testing.

Responsibilities:

1. Text based user interface (TUI) implementation.
2. Step by step and fast execution modes
3. End to end integration testing

Results and Discussion

The implemented system successfully fulfills all functional and technical requirements defined in the project specification. The autonomous vacuum cleaner is able to navigate complex, room grid environment, clean all reachable floor cells and safely manage its battery constraints throughout execution.

Coverage Completeness

Experimental runs across different input scenarios confirm that the system guarantees complete coverage of all reachable floor cells. The use of systematic coverage planning combined with room based decomposition prevents unvisited regions, even in rooms with obstacles and narrow doorways.

The system correctly identifies completion only when no uncleaned floor cells remain, ensuring correctness rather than termination.

Battery Safety and Reliability

Battery aware decision making proved to be critical factor in system reliability. In all tested scenarios, the vacuum cleaner successfully returned to the charging dock before battery depletion. The safety margin mechanism effectively prevented the vacuum from entering unsafe or unrecoverable areas.

After recharging, coverage was resumed from the docks position.

Conclusion

This project demonstrates the successful design and implementation of battery aware autonomous vacuum cleaner simulation using custom data structures and modular design. By strictly avoiding STL containers, the project reinforces a deep understanding of data structures, pointer based management and algorithmic reasoning.

The system effectively combines coverage path planning room decomposition, shortest path navigation and battery safely mechanism to solve a complex real world problem. Comprehensive testing and clear modular separation further contribute to system and maintainability.

Overall the project meets all specified requirements and provides a solid foundation for future extensions such as optimized coverage environment, dynamic obstacle handling with graphical user interfaces (TUI).