HOW DOES ROBOT VACUUM CLEANER WORKS

Project submitted to the

SRM University – AP, Andhra Pradesh
for the partial fulfilment of the requirements to award the degree of

Bachelor of Technology
In
Computer Science and Engineering School of Engineering and Sciences

Submitted by
S.Srividya
(AP23110010696)
P.Sameera
(AP23110010296)
E.Tejaswini
(AP23110010318)
N.Yamini
(AP23110010289)

Under the Guidance of
Mrs.Kavitha Rani Karnena

SRM University–AP Neerukonda,
Mangalagiri, Guntur
Andhra Pradesh – 522 240
[NOV,2024]

Certificate

Date: November - 2024

This is to certify that the work present in this Project entitled "**HOW DOES ROBOT VACUUM CLEANER WORKS"** has been carried out by **Srividya, Sameera, Tejaswini,Yamini** under my/our supervision. The work is genuine, original, and suitable for submission to the SRM University – AP for the award of Bachelor of Technology/Master of Technology in
School of Engineering and Sciences.

Supervisor

(Signature)
Mrs. Kavitha Rani Karnena

Acknowledgements

I would like to express my special thanks of gratitude to my teacher Mrs. Kavitha Rani Karnena who gave me the golden opportunity to do this wonderful project on the topic **HOW DOES ROBOT VACUUM CLEANER WORKS,** which also helped me in doing a lot of Research and I came to know about so many new things. I am really thankful to them.

Secondly, I would also like to thank my Project team who helped me a lot in finalising this project within the limited time frame.

Table of Contents

Abstract:

The following code simulates the working of a robot vacuum cleaner. It is a system modelling simple functionalities of a robot vacuum cleaner used in a 5x5 grid-based room. The main works of the robot include its travelling around, detecting dirty spots, cleaning them, and managing battery life. The robot operates with random movements of directions and cleaning dirty spots as and when it finds them. Another feature of this model is returning a charging station whenever the battery level lowers. The movement of the robot can be monitored by the user, along with information regarding the current battery status with each step. Recharging of the battery is allowed automatically once depleted by the robot, and the cycle continues until all scheduled cleaning tasks are completed. This code simulates the main cleaning activity by a robot vacuum cleaner by using erratic movement, cleaning activities and managing a battery- all with true real-time updates in the status of the robot.

1. Introduction

The C++ code given here simulates a simple working performance of the robot vacuum cleaner, making the system easily and effectively implemented. Today, a world with emphasis on convenience and automation cannot live without such apparatuses like robot vacuum cleaners, becoming important factors in home maintenance: easy, low-manual-intervention techniques for floor maintenance. This code provides a bare, foundational framework that simulates all the core functions that a robotic vacuum cleaner is characterised by, including movement, cleaning, and battery management.

The basic robot vacuum cleaner system is designed with an inbuilt process for automatic cleaning. Thus, it should be capable of movement, cleaning dirty spots, and managing the supply of power. The system emulates the behaviour of a vacuum with a simple grid environment. In this environment, the robot will move around randomly, cleaning off dirt, and returning to a charging station once the battery level is low. Although this implementation is a bit of a basic model, it is clear and easily comprehended to serve as a starting point in developing more sophisticated systems that could expand on features like superior navigation and better cleaning strategies.

Background:

The vacuum cleaning robot shall be designed to traverse space, identify areas that need cleaning, and clean the space autonomously. It shall contain algorithms pertaining to navigation, obstacle avoidance, detection of dirt, and other issues concerning the consumption of power. This C++ simulation embodies the characteristic features of these core functionalities in a 5x5 grid environment. The tasks that it encompasses are traversing around the place, cleaning the cells from the dirt, maintaining its battery levels, and returning to the charging station when needed.

The code aims to present a simpler model reflecting the key components of a robot vacuum cleaner. Still, it must leave room for future growth in the modelling of smarter pathfinding or even object detection and accurate real-time environmental mapping.

Objective:

The core objective of this project is to come up with an accessible simulation that could depict the basic functionality of a robot vacuum cleaner. In this regard, the system allows the robot to clean dirty spots by moving randomly and managing its own battery. The system would simulate the autonomous cleaning process based on the tracking of movements made by the robot, presenting a very basic yet effective display in the manner of how a robot vacuum works.

In addition, this system encourages a modular implementation: developers are left to expand the functionalities offered by the robot vacuum cleaner by incorporating new features such as advanced movement algorithms, complex room environments, and detailed user interfaces.

Scope:

To keep the scope of this project intentionally simple, I focused on some very basic robotic functionalities such as the following:

Random movement within a grid environment

Cleaning dirty spots

Managing battery consumption

Returning to a charging station when the battery is low

The current implementation is a very straightforward model; however, one can extend the scope in the future to include the following:

A large grid for more complex room simulation

Object detection to avoid obstacles much better than its current functionality.

Improved pathfinding algorithms to provide better coverage in cleaning
Integrate with real sensors from the physical world that bring a better interaction with the physical environment.

Significance:

This code will completely change the way homes are kept clean and could be the most basic line of code creating a fully functional robot vacuum cleaner application. It is simulating the operations of movement, cleaning, and management of the battery and can be used to build an advanced autonomous cleaning system.

Not only will this project offer insight into some of the very basic concepts of robotic technology, such as navigation, decision-making, and energy management, but it could also be an excellent tool for developers interested in robotics, artificial intelligence, and automation technologies.

Methodology:

The methodology employed in the project is a simulation of a vacuum robot by using a grid-based system in order to clean. The robot is in a random mode crossing the entire grid cleaning dirty spots and developing a power consumption over battery time. Key steps involved are:

1.Movement Simulation: It means that the robot moves randomly within the grid while taking into account boundaries so that it does not move out of the grid.

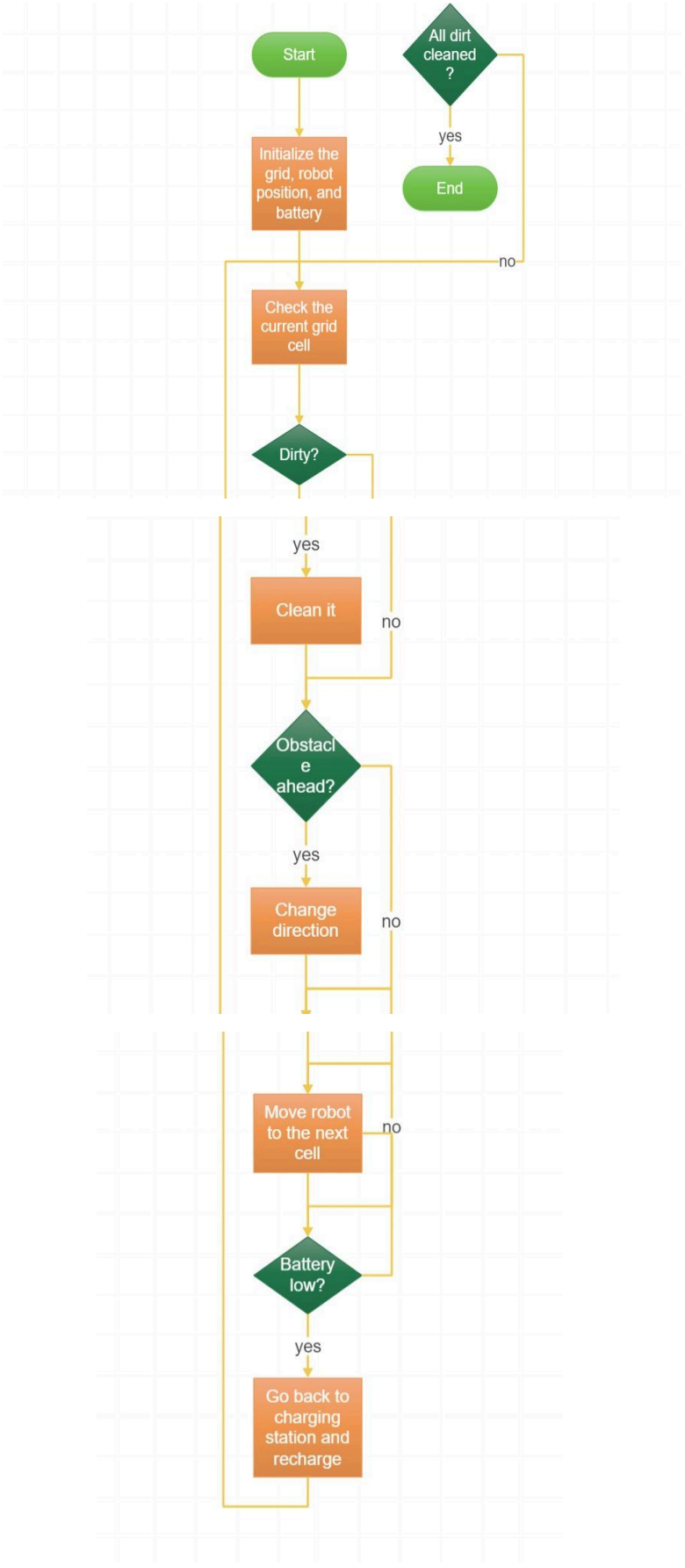
2. Cleaning Process: Upon reaching the spot, the robot will clean that area and the charge of its battery will reduce a fixed value.

3. Battery Management: The robot's charging level is monitored always and supposed to come back to a particular charging station once its charging level goes low.

4. Charging: Upon reaching zero the robot with its battery charging level goes back to the charging station and recharges up to 100%.

5. User Feedback: The software is always giving the user real-time feedback about the current position of the robot, the state of the battery, and the actions taken, such as cleaning or moving around.

FLOW CHART


Start
Initialize the grid, robot position, and battery
Check the current grid cell
Dirty?
yes
Clean it
no
Obstacl e ahead?
yes
Change direction
no
Move robot to the next cell
no
Battery low?
yes
Go back to charging station and recharge
All dirt cleaned ?
yes
End
no

2 Code Implementation:

* Index.cpp

```
#include <iostream>
#include <cstdlib>  // for rand() and srand()
#include <ctime>    // for time()

using namespace std;
```

`#include <iostream>` allows us to use input and output functions, like `cout` and `cin`.
`#include <cstdlib>` includes the `rand()` and `srand()` functions, which generate random numbers.
`#include <ctime>` lets us get the current time, which we use to seed the random number generator.
`using namespace std;` makes it easier to use standard library functions without repeatedly typing `std::`.

```
const int GRID_SIZE = 5;  // A smaller 5x5 grid
const int CLEAN = 0;
const int DIRTY = 1;
const int OBSTACLE = 2;
const int CHARGING_STATION = 3;

class RobotVacuum {
```

`GRID_SIZE` defines the grid as 5x5, a small room for the vacuum to clean.

`CLEAN`, `DIRTY`, `OBSTACLE`, and `CHARGING_STATION` are constants representing different cell types in the grid:

- `CLEAN` = a clean cell.
- `DIRTY` = a dirty cell.
- `OBSTACLE` = a cell that the robot can't enter.
- `CHARGING_STATION` = the robot's starting and charging position.

RobotVacuum Class The `RobotVacuum` class defines the robot's properties and behaviours.

```
private:
    int x, y;       // Robot's position
    int battery;    // Battery level
    int grid[GRID_SIZE][GRID_SIZE];  // 5x5 room grid
```

`x` and `y` represent the robot's current position in the grid.
`battery` keeps track of the robot's battery level.
`grid` is a 2D array representing the room. Each cell can be clean, dirty, an obstacle, or a charging station

```
public:
    RobotVacuum() : x(0), y(0), battery(100) {
        // Initialize random grid with dirty spots and obstacles
        srand(time(0));
        for (int i = 0; i < GRID_SIZE; ++i) {
            for (int j = 0; j < GRID_SIZE; ++j) {
                int randomValue = rand() % 3;  // Randomly place clean, dirty or obstacle cells
                grid[i][j] = randomValue;
            }
        }
        grid[0][0] = CHARGING_STATION;  // Start position is the charging station
    }
```

Initialises `x` and `y` to `0` (top-left corner) and `battery` to `100`.
`srand(time(0));` seeds the random number generator so that the grid setup varies each time you run the program.
A nested loop fills each cell with a random value between 0 and 2 (`rand() % 3`). This makes the cell either clean, dirty, or an obstacle.
Sets the starting position `[0][0]` as a charging station.

```
void move() {
    if (battery <= 0) {
        cout << "Battery empty. Returning to charging station.\n";
        returnToChargingStation();
        return;
    }

    // Move robot randomly (left, right, up, down)
    int direction = rand() % 4;
    switch (direction) {
        case 0: if (x > 0) x--; break;          // Move left
        case 1: if (x < GRID_SIZE - 1) x++; break;  // Move right
        case 2: if (y > 0) y--; break;          // Move up
        case 3: if (y < GRID_SIZE - 1) y++; break;  // Move down
    }
    battery -= 10;  // Reduce battery
    cout << "Moved to (" << x << ", " << y << "). Battery: " << battery << "%\n";
}
```

`move()` checks if the battery is depleted; if so, the robot returns to the charging station.
`direction` chooses a random move (0–3) for the robot:

- `0`: move left if `x > 0`.
- `1`: move right if `x < GRID_SIZE - 1`.
- `2`: move up if `y > 0`.
- `3`: move down if `y < GRID_SIZE - 1`.

```
void clean() {
    if (grid[x][y] == DIRTY) {
        cout << "Cleaning at (" << x << ", " << y << ")\n";
        grid[x][y] = CLEAN;
        battery -= 5;
    }
}
```

`clean()` checks if the current cell is dirty. If it is, the robot cleans it (sets it to `CLEAN`) and reduces the battery by 5%.

```
    void returnToChargingStation() {
        x = 0;
        y = 0;
        battery = 100;
        cout << "Returned to charging station. Battery full at 100%\n";
    }
```

`returnToChargingStation()` moves the robot to the top-left corner (charging station) and fully recharges its battery.

```
    void run() {
        while (battery > 0) {
            clean();
            move();
            if (battery <= 20) {
                cout << "Battery low! Returning to charging station.\n";
                returnToChargingStation();


    }
        }
        cout << "All tasks completed.\n";
    }
};
```

`run()` runs the robot in a loop, where it:

1. Attempts to clean the current cell.
2. Moves randomly.
3. If the battery is 20% or lower, it returns to the charging station.

The loop ends when the battery runs out, displaying "All tasks completed."

```
int main() {
    RobotVacuum robot;
    robot.run();
    return 0;
}
```

In `main()`, a `RobotVacuum` object is created and its `run()` function is called. The program then runs the robot's cleaning and movement cycle.

OUTPUT

```
Moved to (0, 0). Battery: 90%
Moved to (0, 0). Battery: 80%
Moved to (0, 1). Battery: 70%
Cleaning at (0, 1)
Moved to (0, 1). Battery: 55%
Moved to (0, 1). Battery: 45%
Moved to (0, 1). Battery: 35%
Moved to (0, 1). Battery: 25%
Moved to (0, 2). Battery: 15%
Battery low! Returning to charging station.
Returned to charging station. Battery full at 100%
Moved to (1, 0). Battery: 90%
Cleaning at (1, 0)
Moved to (0, 0). Battery: 75%
Moved to (0, 0). Battery: 65%
Moved to (1, 0). Battery: 55%
Moved to (0, 0). Battery: 45%
Moved to (0, 1). Battery: 35%
Moved to (0, 2). Battery: 25%
Moved to (0, 2). Battery: 15%
Battery low! Returning to charging station.
Returned to charging station. Battery full at 100%
```

```
Moved to (0, 0). Battery: 90%
Moved to (0, 0). Battery: 80%
Moved to (0, 0). Battery: 70%
Moved to (1, 0). Battery: 60%
Moved to (1, 1). Battery: 50%
Moved to (1, 0). Battery: 40%
Moved to (0, 0). Battery: 30%
Moved to (0, 1). Battery: 20%
Battery low! Returning to charging station.
Returned to charging station. Battery full at 100%
Moved to (1, 0). Battery: 90%
Moved to (1, 0). Battery: 80%
Moved to (1, 0). Battery: 70%
Moved to (1, 0). Battery: 60%
Moved to (1, 0). Battery: 50%
Moved to (0, 0). Battery: 40%
Moved to (1, 0). Battery: 30%
Moved to (1, 0). Battery: 20%
Battery low! Returning to charging station.
Returned to charging station. Battery full at 100%
Moved to (0, 0). Battery: 90%
Moved to (0, 0). Battery: 80%
Moved to (0, 0). Battery: 70%
```

```
Moved to (0, 0). Battery: 60%
Moved to (0, 0). Battery: 50%
Moved to (0, 1). Battery: 40%
Moved to (0, 0). Battery: 30%
Moved to (1, 0). Battery: 20%
Battery low! Returning to charging station.
Returned to charging station. Battery full at 100%
Moved to (0, 1). Battery: 90%
Moved to (0, 0). Battery: 80%
Moved to (0, 0). Battery: 70%
Moved to (0, 1). Battery: 60%
Moved to (1, 1). Battery: 50%
Moved to (2, 1). Battery: 40%
Cleaning at (2, 1)
Moved to (1, 1). Battery: 25%
Moved to (1, 2). Battery: 15%
Battery low! Returning to charging station.
Returned to charging station. Battery full at 100%
Moved to (0, 1). Battery: 90%
Moved to (0, 0). Battery: 80%
Moved to (0, 0). Battery: 70%
Moved to (0, 0). Battery: 60%
Moved to (0, 0). Battery: 50%
```

3.Concluding Remarks

In short, the above code provides an easy yet reliable example of how a robot vacuum functions in a self-cleaning environment. The above implementation is quite simple and successfully encompasses the core functionalities of a robot vacuum in terms of its random movement, cleaning dirty spots, management of batteries, and return-to-charging behaviour. This system design, then, clearly gives a point of departure for those who are interested in understanding fundamental concepts of robotic automation and cleaning systems.

Although this is relatively simple code, yet much scope is there for its expansions to be undertaken by further developers. The features of intelligent navigation, obstacle avoidance, even more path-finding algorithms might engage developers in taking this building further. This simulation can be much closer to real life robot vacuum cleaners if it includes real-time mapping, sensor-based interactions, and better energy management. Modular nature makes the code quite easily expandable so that developers could add advanced features in structured and order scalable terms.

Again, the Robot Vacuum Cleaner code is focusing on an efficient and self-controlling system design. The fact that this robot can handle its own battery and clean without user interference in actual operations makes it a big deal because feedback from users at each point of actual service ensures full transparency in the cleaning process. This sort of focus in automation and in energy management goes a long way to depict what is important in real-world systems where long-lasting batteries and smart navigation are indispensable for performance in those systems.

Basically, though this project provides a simple illustration of how a robot vacuum cleaner works, its potential capabilities are not limited to what it achieves in the present but can be enhanced from now and into the future as developers continue to improve and enhance this system and make it into a sophisticated and extensive solution for the needs of users who are keen on advancements in robotic automation and smart home technology. This is an initial important step toward developing more complex and efficient robot vacuum cleaners that are in the position to provide great, senseless cleaning solutions for homes and businesses.

4.Future Work

Better Navigation Algorithms:

The current algorithm that it uses is more of a random movement across the grid. Smarter navigation algorithms such as A* or Dijkstra's algorithm can be incorporated to optimise its cleaning path. Using pathfinding techniques, the robot would be able to cover the area in a smarter way without lots of backtracking. Thus, ensuring that the entire area was covered properly.

Obstacle Detection and Avoidance:

Currently, the system lacks the functionality of finding objects to evade. In real-life applications, robot vacuums need the ability to evade furniture and walls among others they are on their path. This can be added to future work through integrating algorithms for obstacle detection that depend on either simulated or actual sensors for sensing the existence of obstacles in the path of the robot. After detecting the object, the system can make adjustments and change its motion appropriately to evade the collision with the object.

Battery Management and Charging Optimization:

Although the current version manages battery levels, more improved versions of such robot systems could include advanced battery systems wherein the robot would return to charging only at precisely needed time periods instead of occasionally throwing back random behaviours. Tracking the battery levels intelligently can ensure it returns to the charging station just in case and optimise battery consumption in terms of area cleaned as its cleaning behaviours can be adjusted according to the existing battery level, enhancing efficiency and increasing the operational time of the robot.

Mapping and Localization:

The system can add functionalities of mapping and localisation. Here, the robot would build a room's real-time map; thus, it is likely not to miss some parts as it cleans. The algorithms like SLAM will allow the robot continuously to build its environment's map while relating its cleaning path. Such functionalities will make the vacuum cleaner more efficient in larger or complex environments.

User Interface and Monitoring:

By bringing in the user interface, users could then easily monitor and manage the performance of the robot. For example, the user can retrieve the present map of the cleaning area, monitor the real-time battery level status of the robot, and know their status regarding the cleaning process. Taking on the remote control or scheduling feature would give even more viability for increasing user interaction with regards to letting them initiate or cancel cleaning at their convenience.

Smart Home Integration:

One major improvement could be in smart home integration. This might include compatibility with home automation systems like Google Home, Amazon Alexa, or Apple HomeKit. After integrating, it would be possible to integrate the robot vacuum with voice commands or through a mobile app to control its operation. Thus, more an intelligent and interactive user experience can be created.

Multitasking Ability:

In the future, the system may be upgraded for multi-parallel operation among many different robots in order to clean bigger spaces. This would include multi-robot coordination whereby a number of robots can work in harmony, share tasks, and optimise coverage for cleaning. This feature would be especially important for larger homeowners or commercial spaces with the potential for needing one robot alone to clean up a large area in a short amount of time.

Scaling Up to Actual Implementation:

Currently, the offered code is just a simple simulation. Future research can be in the direction of actualizing the robot vacuum cleaner into a physical model. It would include sensors, actuators, and real-time environmental measurements that result in a tangible prototype. It might utilise tangible elements such as LIDAR, which could map up and contain collision and sensor response to dirt and obstacles.

REFERENCE

1. **"C++ Primer" (5th Edition, 2012)** by Stanley B. Lippman et al.
    - Covers C++ programming fundamentals, useful for understanding loops, functions, and classes.

2. **"Object-Oriented Programming in C++" (4th Edition, 2002)** by Robert Lafore.
    - Explains object-oriented concepts like encapsulation, constructors, and methods used in your robot class.

3. **"Data Structures and Algorithm Analysis in C++" (4th Edition, 2013)** by Mark A. Weiss.
    - Offers insights into grids and algorithms that could optimize movement and decision-making in your robot simulation.

4. **"Programming: Principles and Practice Using C++" (2nd Edition, 2014)** by Bjarne Stroustrup.
    - Provides foundational knowledge and practical programming practices for C++.

5. **"Introduction to Autonomous Robots" (2nd Edition, 2017)** by Nikolaus Correll et al.
    - Covers the basics of robot navigation and decision-making, similar to the logic in your code.