Project Title: Cleaning Robot Path Planning using A* and RRT Algorithms

Author: Mahmoud M. Shoieb

Description:

This project explores classical path planning algorithms for autonomous cleaning robots. It compares the performance of the A* search algorithm and Rapidly-Exploring Random Tree (RRT) in navigating a grid-based environment with obstacles. The implementation includes visualization using Pygame and evaluation based on efficiency and collision avoidance.

Introduction

The goal of the project is to create an involved management and navigation system for a robotic platform. Building a strong task management system to support the robot's operating skills, putting obstacle avoidance strategies into practice to ensure safe navigation, and developing an effective mapping system to enable the robot to navigate its surroundings on its own are the main goals. Delivery services, public space service robots, and warehouse automation are just a few of the applications for which the system is intended to improve the robot's performance.

Design Approach

This project's design strategy makes use of some of algorithms and techniques for task execution, robot movement, and room setup:

Room Setup: Using a grid-based methodology, the simulation generates a virtual room. The room configuration, including the arrangement of desks, walls, and a sofa, as well as the positioning of dust particles, is specified by the create_rooms function. To stop the robot from passing through the walls, they are designed as barriers.

Robot Movement: A pathfinding algorithm serves as the foundation for the robot's movement logic, which guides it toward arbitrary objects in the space. The robot can move through the environment and avoid obstacles on its way to its destination thanks to the move_towards technique.

Interaction with objects: The robot can identify and engage with dust particles and other objects like desks and furniture.The clean_dust method is activated when the robot enters a point where dust is present, enabling it to clean the area.

Real-Time Simulation: The robot continuously updates its position, looks for events (such user input to end the simulation), and displays the graphical display at a frame rate of 60 frames per second as part of the main loop of the simulation. This strategy guarantees easy mobility and interaction in the surroundings.

Implementation Details

The project's software architecture is set up as follows:

Programming Environment: Python and the Pygame library are used to implement the project, which allows for user interaction and graphical rendering.

Principal Elements:

Grid Initialization: By specifying the locations of the walls and furnishings on a 2D grid, the create_rooms function sets up the room's layout.

Robot Class: This class contains the characteristics and actions of the robot, such as its ability to move, find its way, and clean.

Obstacle Management: To guarantee that the robot can navigate successfully, the simulation has a mechanism to control both stationary (desks and sofas) and moving impediments.

Graphical Assets: Pygame's blit function is used to render graphical assets, such as pictures of dust, desks, and the sofa, on the screen for the simulation.

Challenges and Solutions

1. Managing Moving Obstacles

Problem: The hardest problem was to avoid dynamic obstacles; often, the robots collided or stuck in dynamically heavily moving areas.

Solution: a dynamic grid was utilized, which constantly refreshed the position of the obstacles, including any that were moving, as temporary "walls." Besides that, a function that predicts collisions, will_collide, checks if an obstacle will be at the next target position of the robot. This involves recalculating the path if necessary and thus makes paths more adaptable and reduces frequent path recalculation.

2. Balancing Dust Cover with Navigation

Problem: Sometimes robots preferred to move towards locations already cleaned because the pathfinding focus was on shortest paths, not dust density.

Solution: The robots were programmed to change pathfinding goals toward dusty areas by adjusting the goals to nearby dust particles. Finally, the robots, after clearing **dust in their** immediate location, re-calculated their paths towards the nearest dusty spots and optimized the cleaning performance.

3. Balancing dust coverage and traversal

Robots might opt to move towards locations that have been cleaned already because the algorithm looks for fastest paths, sights set on dust density.

Solution: Programs which controlled waypoints had the requirements updated to lead robots straight towards dusty zones by shifting their pathfinding goals close-by dust bunnies. Moreover, as they completed cleaning where dust was in sight, path recalculations were automated so that robots would move on to the next nearest spots of residue.

Testing and Results

Test scenario1: Dust Coverage

Random Dust Placement in the grid: This scenario places random dust particles on the grid with huge space to roam relatively freely.

Performance: Robots cover the whole room, to ensure to detect also clean fine dust particles. monitor the percentage of dust in the room that has been cleaned and how long it will take to cover your rooms.

Test Scenario 2: Static Obstacle Avoidance

Placing static objects such as desks, sofas and desks on the grid. Robots must find a way around those obstacles and still get to dust.

Performance: Robots should navigate effortlessly around obstacles, capture all visible dust particles while avoiding collisions with walls. Cleaning time, number of successful obstacle avoidance actions and any stuck points (recalculating paths due to an impediment in the way).

Algorithms comparison

Aspect	A* Algorithm	RRT Algorithm
Description	A* is a deterministic search algorithm. It	RRT is a probabilistic based algorithm used

	mainly finds the shortest path from a start to goal node by using a heuristic to guide the search process. It is doing balance between the cost of the path and the estimated cost to reach the goal ensuring optimal and efficient solution.	for pathfinding and planning. It builds a tree by randomly sampling the environment and incrementally connecting the new points, so that it is suitable for high-dimensional and complex spaces when the simple search methods struggle.
Algorithm type	The algorithm is deterministic, graph-based search algorithm	The algorithm is probabilistic, sampling-based search algorithm
Optimality	It guarantees an optimal path if the heuristic used does not overestimate the cost	It doesn't guarantee an optimal path
Search strategy	Uses a queue to always expand the most promising node based on the total cost (g + h)	It grows a tree from the start point, using random sampling to know where to extend the tree
completeness	It always will find a path if it exists	Has a high probability of finding a path
Memory usage	High memory requirements because it stores all the visited nodes	Less memory requirements because it only keeps track of the tree structure

Speed	Fast in environments with a clear structured layout but the performance of the algorithm becomes less in complex or large environment	Fast in environments that have high-dimensional spaces
Obstacle handling	Efficient in static obstacles and becomes less efficient with moving obstacles	Better in handling the dynamic obstacles
Applications	Game development, robot navigation in static indoor spaces also in navigation systems	Robot motion planning, autonomous vehicle navigation
Path smoothing	It reduces smoother paths without lots of preprocessing	Often require more preprocessing to produce smooth path