Project I Report for COM S 4720 Spring 2025: Path Planning

Tanvi Mehetre[1] and Tejal Devshetwar[2]

I. INTRODUCTION

This report summarizes the implementation of the path planning algorithm for navigation in a 30x30(0 to 29) grid environment. The implementation uses Python and employs a priority queue to explore the most promising nodes efficiently. The path planning alorithmn uses A* algorithmn as it balances optimality and efficiency. It combines cost and heuristic cost to determine the best available path.

II. IMPLEMENTATION

A. Data Structures

The data structures used in the algorithmn are listed below.

- **Priority Queue**: They are used to store nodes based on their total cost $f(n) = g(n) + h(n)$ using Pythons's `heapq`.
- **Grid Representation**: The grid environment was provided by the professor which was a 30 x 30 (0 to 29) 2D array where obstacles are marked as impassable paths.
- **Node Structure**: The contents of each node are defined below:
  - **Current Position**: Holds the (x, y) coordinates
  - **g_score**: Holds the cost from the start node
  - **h_score**: Holds the heuristic cost estimate to the goal.
  - **f_score**: The sum of g_cost and h_cost is stored here
  - **came_from**: Reference to the parent node of the current node is stored here.

B. Algorithm Steps

The function plan_path in the planner.py file implements the A* algorithm to find the optimal path from the start node to the end node.

1) Initialization: The initialization of the grid is done in the main.py file which was provided to us from the start. The file defines a 30 x 30 grid environment with obstacles, start node and end node. This file also contains the functions that visualize, check and plan the path from the start to end nodes.

2) Heuristic Function: The heuristic function computes the Euclidean distance between the current node and the goal. This is the value of h(n) which is the estimate of the remaining distance to the goal.

3) Valid Neighbors: The function get_neighbors takes the current node as the input and returns all the neighbors of the current node in the 8 directions defined. It also ensures that the grid boundries are respected and not crossed as well as avoiding obstacles which are marked as 1.

directions = [(-1, 0), (1, 0), (0, -1), (0, 1), (-1, -1), (-1, 1), (1, -1), (1, 1)]

4) A Algorithm:* This algorithm is implemented in the path_plan function where it searches for the most optimal path from the start node to the end node.

First we initialize the start and end nodes positions to the integer tuples. Proceeding we initialize the priority queue with the start node and cost 0 which will later keep on changing. g_score keeps the track of cost of reaching one node from the start and f_score keeps a track of the total cost required to reach the goal.

Then the priority queue extracts the node with the lowest total cost f(n) in the while loop and when the goal is reached the path is reconstructed.

To explore the neighbors a tentative g_score is calculated, and if this new calculated value is better than the previous the values are updated accordingly. The neighbor is then added to the priority queue.

5) Path Reconstruction: If a path is found from start to end nodes then the algorithm backtracks using came_from dictionary and reconstructs a path the start to the end nodes.

C. Heuristic Function

The heuristic function determines the estimated cost from a node to the goal. We used the Euclidean distance:

$$h(n) = \sqrt{(x_{\text{goal}} - x_{\text{current}})^2 + (y_{\text{goal}} - y_{\text{current}})^2}. \quad (1)$$

which accounts for diagonal movement in the 8-connected grid environment.

Equation (1) represents the heuristic cost, where x and y denote the Cartesian coordinates of the current and goal nodes. This heuristic ensures an optimal path estimation while considering diagonal movements in the search space.

III. TIME COMPLEXITY

Best Case: O(n), where n is the number of steps in the shortest possible path.

Worst Case: O(N^2), where N is the size of the grid. In our case 30x30.

[1]Tanvi Mehetre is a student in the Department of Electrical and Computer Engineering, Iowa State University, Ames, Iowa, United States tanvim@iastate.edu

[2]Tejal Devshetwar is a student in the Department of Electrical and Computer Engineering, Iowa State University, Ames, Iowa, United States tejal@iastate.edu

IV. CONCLUSIONS

In conclusion, A* is algorithm that effectively finds the shortest path and avoids obstacles by combining heuristic estimation, cost tracking, and priority exploration. Euclidean distance heuristic is used to estimate the remaining cost to the goal and prioritizes nodes based on their g_score (actual cost) and f_score (estimated total cost). While obstacle avoidance is managed by taking into account only legitimate neighbors, efficient exploration is ensured using a priority queue (min-heap). The path id reconstructed using the dictionary. Therefore, this combination produces a favorable result.

REFERENCES

https://www.geeksforgeeks.org/a-search-algorithm/