Team SAY
SANCHIT GARG
ABHIRAM SIDDANTHI
YASH HARALE

PATH PLANNING

12th May 2023

OVERVIEW

Implementing Dijkstra's algorithm and A* algorithm to find the shortest path between two points on a map. We then imported the map from Google Maps and applied the algorithms on map, which is an example of a real life application.

PROGRAM REQUIREMENTS

1. MODULES - cv2, numpy, collections
2. Map image, with the location of the image mentioned in the code.

SPECIFICATIONS

Converting google map to black and white image

The image of google map is converted to a traversable black and white matrix of 3 Dimensions using threshold value of 10.

DIJKSTRA'S ALGORITHM:

Dijkstra's Algorithm works on the greedy approach that any subpath B -> D of the shortest path A -> D between vertices A and D is also the shortest path between vertices B and D, in the opposite direction.

- Function calcDist calculates the distance between the current point and another point
- Function isWell checks if the a certain point is within the boundaries of the matrix
- Dijkstra's function: Using the image, start point and end point, it does the following:
    1. It initializes a distance matrix dist with shape (h, w) and fills it with infinity values except for the starting point (start), which is set to 0.

2. It creates a parent matrix parent with shape (h, w, 2) to store the parent coordinates of each pixel in the shortest path.
3. It initializes a visited matrix with shape (h, w) to keep track of visited pixels. The starting point is marked as visited (1), and all other pixels are initially set to 0.
4. Beginning from the start point, It iterates over the neighbors of the current point (including diagonal neighbors) using nested loops. For each neighbor, it checks if it is within the image boundaries and if it satisfies certain color conditions.
5. If the neighbor passes the conditions, it calculates the distance from the current point to the neighbor and checks if this distance, combined with the distance to the current point, is smaller than the current distance stored in the dist matrix. If it is smaller, it updates the dist matrix and sets the parent of the neighbor point to the current point.
6. After updating the distances for all valid neighbors, it finds the unvisited pixel with the minimum distance from the matrix and sets it as the new current point.
7. It calls the showPath function to visualize the path found so far, passing the image (img), the current point, the starting point, and the parent matrix.

- Function showPath helps visualize the shortest distance between the start and the end function. It does the following:

2. It creates a copy of the original image using `np.copy()` and assigns it to the variable `new`.

3. It enters a loop that continues until the current point is equal to the start point.

4. Within the loop, it retrieves the parent coordinates of the current point from the `parent` matrix and assigns them to the variable `var`.

5. It updates the corresponding pixel in the `new` image to a green color using `new[int(var[0]), int(var[1])] = green`. This color change represents the path from the current point to its parent.

6. It updates the `current` point to the parent coordinates `(var[0], var[1])`.

7. After the loop finishes, it implies that the current point is equal to the start point, and the path visualization is complete.

8. It creates a named window called `'dijkstra's path'` using `cv2.namedWindow()'.

9. It displays the `new` image in the `'dijkstra's path'` window using `cv2.imshow()`.

A* ALGORITHM

What is A* Algorithm?

A* Algorithm is a path-finding smart algorithm to approximate shortest path between target and initial point avoiding obstacles on its way.

Analogy of Astarpath function in python code

This function performs the A* algorithm on the maze image to find the shortest path from the start to the end point. The function is implemented as follows:

1. The maze image is converted to a NumPy array (img) for easier manipulation.
2. The start and end points are converted to a different coordinate system (startastar and endastar) for A* algorithm implementation.
3. The get_min_dist_node function retrieves the node with the minimum distance from the open list.
4. The show_path function reconstructs and displays the shortest path on the image.
5. The get_dist function calculates the Euclidean distance between two points.
6. The obstacle function checks if a given position is an obstacle in the maze.
7. The goal_reached function checks if the current position is the goal position.
8. The astar_algorithm function is the main A* algorithm implementation:
    a. It initializes the open list and closed list.
    b. It iteratively explores the neighboring positions of the current position, calculates the cost, and updates the open list.
    c. The shortest path is visualized by calling the show_path function.
9. The astar_algorithm function is called with the startastar and endastar parameters.
10. Finally, the modified image with the shortest path is displayed using cv2.imshow.

A* ALGORITHM LOGIC

Consider a square grid having many obstacles and we are given a starting cell and a target cell. We want to reach the target cell (if possible) from the starting cell as quickly as possible.

A* Search Algorithm picks the node according to a value-‘f’ which is a parameter equal to the sum of two other parameters – ‘g’ and ‘h’. At each step it picks the node/cell having the lowest ‘f’, and process that node/cell.

g = the movement cost to move from the starting point to a given square on the grid, following the path generated to get there.

h = the estimated movement cost to move from that given square on the grid to the final destination.

We have to approximate the value of h using some heuristics (smart guess) like Euclidean distance, etc as it will consume less time.

Euclidean Distance

It is the distance between the current cell and the goal cell using the distance formula.

```
h = sqrt ( (current_cell.x - goal.x)2 +
           (current_cell.y - goal.y)2 )
```

A* PSEUDO CODE

// A* Search Algorithm

1. Initialize the open list

2. Initialize the closed list

   put the starting node on the open list

3. while the open list is not empty

   a) find the node with the least f on the open list, call it "q"

   b) pop q off the open list

   c) generate q's 8 successors and set their parents to q

   d) for each successor

      i) if successor is the goal, stop search

      ii) else, compute both g and h for successor

      successor.g = q.g + distance between successor and q

      successor.h = distance from goal to successor

successor.f = successor.g + successor.h

iii) if a node with the same position as successor is in the OPEN list which has a lower f than successor, skip this successor

iV) if a node with the same position as successor is in the CLOSED list which has a lower f than successor, skip this successor

otherwise, add the node to the open list, end (for loop)

e) push q on the closed list end (while loop)

PROJECT WORKING SCHEMATICS