|  |  |
| --- | --- |
| Capstone Project Report  Udacity Machine Learning Nanodegree | Icon  Description automatically generated with medium confidence |

*Paima Marbun*

*10/18/2021*

Robot Motion Planning

Plot and Navigate a Virtual Maze

1. **DEFINITION**

**Project Overview**

This project is based on Micromouse competition where a small robot should find autonomously the fastest possible path from predetermined starting node to one of the goals located in the center of an unknown maze. The maze is typically made up of a 16x16 grid of nodes with or without walls on its four edges. The robot can explore and discover the walls in the first run to map out the maze and detect the goals. From the learning of the first run the robot is expected to know the fastest path to one of the goals in the maze center and it attempts to reach it the second run.

I have selected this domain as robotics and autonomous-, intelligent-system are to me currently the most exciting, challenging, and fast-growing field.

**Problem Statement**

The use case should be simulated by applying virtual robot and virtual maze. The virtual robot should find the fastest path from the starting point in the bottom-left corner of the maze and facing upwards to one of the goals in center of the maze. The maze has n x n dimension of squares with minimum n value of 12 and maximum n of 16. Each square or node can have walls on their edges which restrict movement in respective direction. The robot is equipped with three obstacle sensors mounted in the front, left and right side. At any time, robot can move forwards or backwards up to three squares as well rotate clockwise or counterclockwise 90 degree.

The problem of this project is to find an algorithm for the robot as a guide to discover an unknown maze and reaching the goals in fastest time possible. Several algorithms should be investigated to determine the most optimal one to cope with three sample mazes as per the project requirements. To provide more general solution and to reduce biases for future mazes performances against different other samples should be considered too.

**Evaluation Metrics**

The performance of the robot takes both exploration (run1) and optimization (run2) into consideration and calculated as

Score = (run1 timesteps / 30) + run2 timesteps

The robot can spend only maximum of one thousand timesteps to complete both run1 and run2 for a single maze. Since the weight for run2 portion is 30 times higher than run1 portion the algorithm will prioritize getting better performance on the second run at the expense of timesteps required in run1.

1. **Analysis**

**Data Exploration and Visualization**

The only inputs fed into the robot are the maze dimension, start position plus heading, and the four possible goals in the center of a maze. For a test system the entire maze is provided as text file contains comma separated values which outline walls for respective square or node. The four-bit binary value represents four wall edges on a square. Bit value 0 means edge is open and 1 means edge is blocked. The three minimum samples for this project are provided by Udacity. Following scheme should illustrate how to create a new maze based on example test\_maze\_01.txt.

Diagram

Description automatically generated

The robot’s start position is in the bottom left-hand corner of the maze (coordinate [0,0]) and facing in an upward direction as shown below with the blue turtle. The four goals are centered in the maze marked with red dots. The robot has three obstacle sensors which provide number of permissible nodes in its respective directions, to the front, left, and right. As mentioned earlier the robot can visit a permissible node by performing forward or backwards movement of up to three nodes and -90° counterclockwise, 90° clockwise or 0° rotation. The robot should prefer a path with maximal possible steps for its movement to a targeted node than a path with many turns to get less timesteps for the A picture containing qr code

Description automatically generatedscoring if both paths require same single steps. The run1 can be terminated once the goal is found and the targeted exploration is completed. It then follows run2 to reach the goal in fastest timesteps. The optimal timesteps for the test\_maze\_01 is 17 as plotted in this figure.

**Algorithm and Techniques**

There are several proven algorithms for path finding available e.g., the popular A\*. But these all require some knowledge of the maze to output an optimal path from start to goals. And the best optimal path can certainly be achieved if entire maze is known or 100% discovered during first exploratory run. Thus, the main challenge in this project is to explore the maze as many possible in minimum timesteps to find as many as possible paths to goal so that the second optimization run can achieve optimal path close to the best optimal path popular algorithms would achieve with full knowledge of the maze. Since the robot is allowed to further map out the maze after visiting the goal in the first run two approaches will be performed. The one explores the given maze until maximum coverage and the other one explores only until one of the goals found.

For exploratory run1 I came up with following two approaches:

* Short-turn

With this approach the robot will first explore close neighbors (smallest distance from the robot current node) which are unexplored but already detected via sensors reading. A list containing open nodes should guide the robot during the exploration. The node in the list with the smallest distance from the current node should be selected for the next move and if there are several candidates with the same distance then one will be randomly chosen. Already visited nodes should be skipped. This process continues until either the goal is found, or a good exploration is reached.

* Heuristic-turn

The node with having smallest heuristic value plus the distance to the robot’s current position will be selected up for the next move. The heuristic value for the given maze dimension, start and goals is generated prior the exploration run1. Same termination logic from the Short-turn is also applied here.

For optimization run2 I decided to use A\* algorithm. The A\* algorithm first examines nodes which are currently having potentially the fastest path to target. The metric or f-cost is calculated from the sum of heuristic value and g-cost.

*f*(x) = *g*(x) + *h*(x)

A picture containing box and whisker chart

Description automatically generatedThe heuristic value *h*(x) is the theoretical distance of each node to the goal without knowing any walls as shown in the left figure calculated for test\_maze\_01. While *g*(x) g-cost is the distance from the start to each node in the maze. The node with the smallest f-cost will be prioritized to be expanded further. This loop continues until the goal node is examined. In the final step the predecessor node will be used to backtrack the path until the starting node.

**Benchmark Model**

The initial benchmark of maximal 1000 timesteps for both run1 and run2 as per the project requirement should be taken as the poorest baseline. To compare run2 performance which is the significant contributor to the final score A\* algorithm optimal results should be taken as reference.

With the two references, I will investigate four potential solutions as described above and select one candidate having descent performance on three provided mazes and additional new five mazes. The main ideal would be selecting one algorithm which in general also perform on future mazes.

1. **Methodology**

**Data Preprocessing**

Maze specification and samples are provided by the project hence no additional data preprocessing is required.

**Implementation**

The provided starter code will be re-used to and extended. The main functionality will reside in robot.py. The test function will be extended to accommodate various tests and repetition due to randomness and at the end prepare test statistics in a data frame for details analysis. An animation class is added to have animation using turtle module which is helping visualizing robot movement during troubleshooting in addition to logging.

Program workflow:

1. Initialization

The robot will be initialized with the inputs of maze dimension, start and goal position.

Attributes like map and heuristic table are then populated accordingly.

1. Interpreting Sensor Reading

In each time-step the robot receives sensor information about the distance to wall from its current position in the direction of respective sensor, left, upwards and right. The robot then interprets how many and which nodes are permissible.

1. Replicating Maze

The maze table with the dimension of the maze is initialized with all zero-value meaning all nodes are initially surrounded by walls. Based on workflow 2 results each actual detected walls of respective and adjacent nodes are updated accordingly. Walls surrounding node are fully uncovered once node is entered hence maze update for such nodes can be skipped.

1. Updating Robot Attributes and Internal States

At this point the robot is aware of newly detected permissible nodes and append them to a list containing actual unexplored nodes for the next step to assess. Moreover, for Heuristic turn and A\* algorithm support g-cost and f-cost will be calculated and populated in each table.

1. Executing Exploratory Run1

Main result of this step is to identify next possible move of the robot to explore the maze. At first distance between each node in the open list and current robot position must be calculated and depending on actual chosen algorithm for the Short-turn the smallest distance between two nodes will be selected while for the Heuristic-turn the smallest value of distance between two nodes plus the heuristic value of targeted node. Each time calculation of distance between two nodes is required A\* search will be triggered to obtain the optimal path. The robot will trigger single or consecutive movement according to the path to enter the targeted node. During the exploration robot must keep tracking of its location and heading, updating the table of node status indicating closed/undetected, open/detected, or done/visited, and updating the table of number of how frequent node is visited.

1. Terminating Exploratory Run1

The exploratory run1 will be terminated depending on actual chosen algorithm either once the goal is found or until exploration achieves targeted coverage after entering the goal. The Robot will reset its location back to the original start position and heading. To signal this event to calling (test-) unit “Reset” and “Reset” are returned instead of valid movement and rotation.

1. Executing A\* search For Optimization Run2

After gathering sufficient or full knowledge of the maze Run2 should now find the best optimal path from the starting position to the goal using A\* search algorithm as described earlier. A dynamic list containing open nodes for expansion will be maintained in the process. New node will be added to the list as result of predecessor expansion. Already expanded node will be recorded in the look-up table and removed from the running list. To identify the next promising node for expansion, distance between each candidate in the list and the robot’s current position plus heuristic value must be calculated. The node with the lowest f-cost or sum of both values will be further expanded. If multiple candidates exist, one will be randomly chosen. The expansion continues until the goal node is expanded and backtrack the path to the starting node to compile optimal path. Finally the robot triggers consecutive movement and rotation until the goal node entered.

Classes / Modules:

* **Robot class** (robot.py)

Main Functions:

***\_\_init\_\_( ):***

This will initialize the needed attributes and states as described in the workflow 1

Input parameters: Maze dimension, selected algorithm for run1

***fill\_map\_heuristic( ):***

This routine will do the tasks defined in the program workflow 2 - 4

Input parameters: sensors reading

***next\_move( ):***

This routine will do the tasks defined in the program workflow 5 - 6

Input parameters: sensors reading

Return parameters : -90°/0°/90° rotation and forward/backward movement

***find\_best\_path( ) :***

This routine will do the tasks defined in the program workflow 7

Input parameters: start node, heading, and goals node

Return parameters : -90°/0°/90° rotation and forward/backward movement

Data Structure

* Maze class (maze.py)
* Mazeanim class (mazeanim.py)
* Algorithm Tester module (alg\_tester.py)
* Showmaze module (showmaze.py)
* Tester module (tester.py)

Main functions

Data Structure

Testing and Animation

**Refinement**

1. **Results**

**Model evaluation and validation**

**Justification**

The first step will be to review the provided starter code and get an understanding of the program flow and conclude on what part can be taken for full re-use and where the main function should reside. To get better overview and orientation of the provided mazes graphical visualization need to be plotted. Since A\* will be used as my benchmark model this should be first implemented and tested. The function should return best optimal path given maze, start and goal position. For fast reproduction and easy debugging purpose simple maze should be taken as an input. It has a dimension of 6x6 squares as shown below. The start is marked as blue, goal is red and black line should indicate walls.

The goal and start can be placed everywhere in the squares so the function should be able to deal with general use cases and not only tailored for predetermined start as centered goals as this function is also needed for run1. Once this is verified that it works properly then it should be further tested against three provided mazes and verify it manually with eyeball that best optimal path is returned.

For exploratory run1 two approaches will be investigated. The first one is a random movement of robot to visit permissible closest nodes (smallest distance from current node). A priority list containing already seen nodes as result of sensor feedback should guide the robot during the exploration. The node with smallest distance (utilizing max move of the robot) should be put on the top of the list and already visited nodes are marked and should be skipped. This process continues until either the goal is entered, or high maze coverage is reached. The second one is following heuristic value or f-cost in A\* term with same termination logic as the first one. A test function should be extended to accommodate various tests and repetition due to random nature of the solution and at the end provide test statistics in a panda dataframe for details analysis. I also plan to have animation support using turtle module to visualize robot movement in addition to logging.

**Sources:**

Udacity Project Files Link:

<https://docs.google.com/document/d/1ZFCH6jS3A5At7_v5IUM5OpAXJYiutFuSIjTzV_E-vdE/pub>

Udacity starter code:

<https://www.google.com/url?q=https://drive.google.com/open?id%3D0B9Yf01UaIbUgQ2tjRHhKZGlHSzQ&sa=D&source=editors&ust=1633951177995000&usg=AOvVaw20wxDnXySWm08F-g3vMcnc>

Micromouse Wikipedia:

<https://en.wikipedia.org/wiki/Micromouse>

A\* Search Algorithm:

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

[https://en.wikipedia.org/wiki/A\*\_search\_algorithm](https://en.wikipedia.org/wiki/A*_search_algorithm)

A\* Pathfinding (E01: algorithm explanation) Video:

<https://youtu.be/-L-WgKMFuhE>