Robot Motion Planning

A proposal for the Capstone Project

NEELESHKUMAR SRINIVASAN MANNUR
Udacity Machine Learning Engineer Nanodegree

Contents

Domain Background

The project takes inspiration from Micromouse competitions, wherein a robot mouse is tasked with plotting a path from a corner of the maze to its centre. The robot mouse may make multiple runs in each maze. In the first run, the robot mouse tries to map out the maze to not only find the centre, but also figure out the best paths to the centre. Simply put, the mouse is in an exploratory mode trying to explore various ways to reach the centre. In subsequent runs, the robot mouse attempts to reach the centre in the fastest time possible, using what it has previously learnt. This project will create functions to control a virtual robot to navigate a virtual maze. The goal is to obtain the fastest times possible in a series of test mazes from a simplified model which is provided along with the specifications for the maze.

I could see that Q Learning Algorithm is used for solving the maze problem. This encourages me to take up this problem and solve it with some other algorithm

Source: http://ieeexplore.ieee.org/document/5967320/?reload=true

I have used the dataset from the Udacity Project files for solving the problem.

Problem Statement

The goal is to get to the centre of the maze as fast as possible. The robot will start in the bottom-left corner of the maze and must navigate from the start point to the goal in a reasonably short possible time, and in an optimal number of steps or moves. The maze is a square with equal rows and columns where the number of squares along each side (n) should be either 12, 14, or 16. The circumference / perimeter of the maze is surrounded by walls that act as a preventing barrier, causing the micromouse not to move outside the maze.

The origin will have barriers on 3 sides viz. left, right, and back. This will cause the micromouse to move only in one direction i.e. forward only. The goal zone will be a 2x2 square in the centre of the maze. The micromouse will run a first trial, exploratory in nature to understand the structure and shape of the maze, including all possible routes to the goal area. The micromouse must travel to the goal area to complete a successful exploration journey, but can continue its exploration after reaching the goal. The second trial is where the action happens. In this trial, the micromouse will recall the information from the first trial and will attempt to reach the goal area in an optimal route.

The micromouse's performance will be scored by adding the following:

1. Number of total steps in the exploration trial divided by 30 [Specification in Project Files]
2. Total number of steps to reach the goal in the optimization (second) trial [Specification in Project Files]

A maximum of one thousand time steps are allotted to complete both runs for a single maze. The micromouse can only make 90 degree turns in either clockwise or counter clockwise directions and can move up to 3 spaces forward or backward in a single move.

Datasets & Inputs

Examining the project files, we find out that the maze layout is available (test_maze_xx.txt). I will go ahead and use the maze files provided in the Udacity project files. The following are the observations from the data files.

1. First line determines the number of squares for each side in the maze.
2. Subsequent lines determine structure of maze via comma-separated binary values that define which sides have walls and which have openings.

To make a move, the micromouse must have following information –

1. The micromouse's exact location in the maze
2. The number of walls surrounding it
3. What is the intended action
4. Any previous knowledge and routes favouring the maze from prior trials.

The location of the micromouse will be defined by a pair of 2 digits, such as [1,10]. Each action contains movement which will be a number from -3 to 3, and rotation which will be -90, 0, or 90.

Brief Description of the Input [Source: Udacity Project Files]
The maze exists on an *n* x *n* grid of squares, *n* even. The minimum value of *n* is twelve, the maximum sixteen. Along the outside perimeter of the grid, and on the edges connecting some of the internal squares, are walls that block all movement. The robot will start in the square in the bottom- left corner of the grid, facing upwards. The starting square will always have a wall on its right side (in addition to the outside walls on the left and bottom) and an opening on its top side. In the centre of the grid is the goal room consisting of a 2 x 2 square; the robot must make it here from its starting square to register a successful run of the maze.

Mazes are provided to the system via text file. On the first line of the text file is a number describing the number of squares on each dimension of the maze *n* which is either 12, 14 or 16. On the following *n* lines, there will be *n* comma-delimited numbers describing which edges of the square are open to movement.

```
12
1,5,7,5,5,5,7,5,7,5,5,6
3,5,14,3,7,5,15,4,9,5,7,12
11,6,10,10,9,7,13,6,3,5,13,4
10,9,13,12,3,13,5,12,9,5,7,6
9,5,6,3,15,5,5,7,7,4,10,10
3,5,15,14,10,3,6,10,11,6,10,10
9,7,12,11,12,9,14,9,14,11,13,14
3,13,5,12,2,3,13,6,9,14,3,14
11,4,1,7,15,13,7,13,6,9,14,10
11,5,6,10,9,7,13,5,15,7,14,8
11,5,12,10,2,9,5,6,10,8,9,6
9,5,5,13,13,5,5,12,9,5,5,12
```

Each number represents a four-bit number that has a bit value of 0 if an edge is closed (walled) and 1 if an edge is open (no wall); the 1s register corresponds with the upwards-facing side, the 2s register the right side, the 4s register the bottom side, and the 8s register the left side. For example, the number 10 means that a square is open on the left and right, with walls on top and bottom (0*1 + 1*2 + 0*4 + 1*8 = 10). Note that, due to array indexing, the first data row in the text file corresponds with the leftmost column in the maze, its first element being the starting square (bottom-left) corner of the maze.

Robot Specification [Source: Udacity Project Files]
The robot can be considered to rest in the centre of the square it is currently located in, and points in one of the cardinal directions of the maze. The robot has three obstacle sensors, mounted on the front of the robot, its right side, and its left side. Obstacle sensors detect the number of open squares in the direction of the sensor; for example, in its starting position, the robot's left and right sensors will state that there are no open squares in those directions and at least one square towards its front. On each time step of the simulation, the robot may choose to rotate clockwise or counter clockwise ninety degrees, then move forwards or backwards up to three units. It is assumed that the robot's turning and movement is perfect. If the robot tries to move into a wall, the robot stays where it is. After movement, one time step has passed, and the sensors return readings for the open squares in the robot's new location and/or orientation to start the next time unit.

More technically, at the start of a time step the robot will receive sensor readings as a list of three numbers indicating the number of open squares in front of the left, centre, and right sensors (in that order) to its "next_move" function. The "next_move" function must then return two values indicating the robot's rotation and movement on that time step

Solution Statement

In the first trial, we try to find the solution to the question - what is the layout of the maze? So, the first trial can be considered as a learning trial wherein the micromouse tries to understand or rather collect information about the layout of the maze. Also, it will find all the possible paths to the centre of the maze. In the second trial, we try to reach the centre of the maze using an optimal path, so that the micromouse reaches the centre in the least possible number of moves.

We constantly assess the number of total moves taken, along with the total time the micromouse is navigating the maze. Thus, it can be conveyed that the environment is quantifiable. After the first trial, the micromouse must have recorded multiple possible routes to the centre of the maze, and must be able to choose the optimum route. The solution can be replicated by repeating the optimization trial for additional rounds. Assuming the micromouse initially identified and voyaged

on an optimal route, its path to the centre of the maze should remain the same if it were subject to few more trials.

I will consider the following algorithms to solve the maze problem:

1. Dijkstra's Shortest Path Algorithm
2. A* Best First Search Algorithm
3. Breadth First Search Algorithm
4. Flood Fill Algorithm

Benchmark Model

The benchmark model can be evaluated as:

score = [Number of Steps in Trial 2] + [Number of Steps in Trial 1 / 30]

Because we limit the number of steps to 1000, this will be the most basic form of a benchmark performance. The benchmark model will consist of a micromouse that makes a random movement decision at each step. If the mouse can navigate to the goal in fewer than 1000 steps during its exploration trial, a new benchmark should be set to the total number of steps in Trial 1. Because the micromouse has gained knowledge of the maze layout in its exploration trial, it should then optimise its route to the goal in Trial 2. Regardless of the number of steps taken during Trial 1, by nature of design, each maze has an optimal route - which is the minimum number of steps required to get from the origin to the centre area.

Apart from the aforementioned model, I would also try out Breadth First Search Model as a benchmark.

Evaluation Metrics

The main evaluation metric to determine the performance of the benchmark and solution model is

score = [Number of Steps in Trial 2] + [Number of Steps in Trial 1 / 30]

To demonstrate with an example, let us consider the micromouse takes 600 steps during the first trial and 30 steps in the second trial, then the score will be

30 + (600 / 300) = 32

Project Design

I will follow the following procedure to tackle the problem

1. Defining and understanding all essential project parameters
2. Analysis of the starter code
3. Consider specific '.py' files to determine what actions must be taken in order that the solution is obtained
4. Once the solution is obtained, I will evaluate the score.
5. I will optimise the code to get a better score if possible.
6. Finally, I will document all the steps followed and actions taken in my project report.

I will consider the following algorithms, but not restrict myself to:

1. Dijkstra's Shortest Path Algorithm: Dijkstra's algorithm is an algorithm that finds the shortest path from one node to all other nodes in its network. By finding the distance to all

nodes in its path, it will inevitably create a connection with the nodes represented by the goal area.

2. A*: A* is a best-first search algorithm that searches for all possible solutions to the goal area and will choose the shortest path from the starting node to the ending node. Starting from a specific node, it creates a tree path of each possible direction until it reaches the goal area.

3. Breadth First Search: This algorithm will start at a tree root and consider its closest neighbours before looking on to its next level neighbours

4. Flood Fill Algorithm: This algorithm starts in the centre area and will fill each surrounding cell with a number of its relative distance from the goal area

   And other algorithms which will help me in finding the shortest path / optimal path

References

Udacity Project Files
https://docs.google.com/document/d/1ZFCH6jS3A5At7_v5IUM5OpAXJYiutFuSIjTzV_E-vdE/pub

Maze Solving Algorithms on YouTube
https://www.youtube.com/watch?v=NA137qGmz4s

Micromouse USA
http://micromouseusa.com/

USC Micromouse Project
http://robotics.usc.edu/~harsh/docs/micromouse.pdf