Robot Motion Planning

Plot and Navigate a Virtual Maze

Capstone Proposal

Vishwas V
April 11th, 2018

Domain Background

This project takes inspiration from 'Micromouse' competitions, wherein a robot mouse is tasked with plotting a path from a corner of the maze to its center. The robot mouse may make multiple runs in a given maze. In the first run, the robot mouse tries to map out the maze to not only find the center, but also figure out the best paths to the center. In subsequent runs, the robot mouse attempts to reach the center in the fastest time possible, using what it has previously learned.

The maze is made up of a 16 by 16 grid of cells, each 180 mm square with walls 50 mm high. The mice are completely autonomous robots that must find their way from a predetermined starting position to the central area of the maze unaided. The mouse will need to keep track of where it is, discover walls as it explores, map out the maze and detect when it has reached the goal. Having reached the goal, the mouse will typically perform additional searches of the maze until it has found an optimal route from the start to the center. Once the optimal route has been found, the mouse will run that route in the shortest possible time.

Problem Statement

In this project, we will create functions to control a virtual robot to navigate a virtual maze. A simplified model of the world is provided along with specifications for the maze and robot; our goal is to obtain the fastest times possible in a series of test mazes. On each maze, the robot must complete two runs. In the first run, the robot is allowed to freely roam the maze to build a map of the maze. It must enter the goal room at some point during its exploration, but is free to continue exploring the maze after finding the goal. After entering the goal room, the robot may choose to end its exploration at any time. The robot is then moved back to the starting position and orientation for its second run. Its objective now is to go from the start position to the goal room in the fastest time possible.

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 center of the grid is the goal room consisting of a 2 x 2 square; the robot must make it here from its starting square in order to register a successful run of the maze.

The robot can be considered to rest in the center 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 a distance of 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.

Datasets and Inputs

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*. On the following *n* lines, there will be *n* comma-delimited numbers describing which edges of the square are open to movement. 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).

As for the robot, 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 timestep. Rotation is expected to be an integer taking one of three values: -90, 90, or 0, indicating a counter clockwise, clockwise, or no rotation, respectively. Movement follows rotation, and is expected to be an integer in the range [-3, 3]

inclusive. The robot will attempt to move that many squares forward (positive) or backwards (negative), stopping movement if it encounters a wall.

Solution Statement

The robot will explore the maze in the first run where it will learn and map the structure of the maze, and find all possible paths to the goal. Then in the next run the robot will try to reach the goal following an optimal path in the fastest time or precisely in minimum number of steps.

The number of steps can determine whether it is the best path to reach the goal as the robot will have the knowledge of number of steps needed to reach the goal for all possible paths, and the path with the fewest steps is the fastest one. In order to test if our robot has learned the optimal policy we can perform multiple runs and for each run the path should remain same if it is indeed the best path to the goal.

Benchmark Model

The Benchmark Model for the agent directly relates to how the evaluation metrics are done for scoring the agent's path planning regarding steps taken within the environment. The robot's score for the maze is equal to the number of time steps required to execute the second run, plus one thirtieth the number of time steps required to execute the first run and the agent is restricted to a total of one thousand time steps total for both runs within the environment.

For example, one baseline model would be that of taking five hundred actions for each run, maximizing the allowed one thousand time steps. These actions could be random actions to create a true baseline model. The following score would result for the baseline model:

$$\text{Score} = \#\text{steps-run2} + (1/30) * \#\text{steps-run1}$$
$$\text{Score}_{\text{Baseline}} = 500 + (1/30) * 500$$
$$\text{Score}_{\text{Baseline}} = 516.67$$

Evaluation Metrics

For each of the test mazes, the virtual robot moves through the maze twice. In the first run, the virtual robot can move freely through the maze in an attempt to explore and map it. It is free to continue exploring the maze even after entering the goal area. Once the virtual robot has found the goal, it may choose to end the exploration run at any time.

The second run of the environment, the agent will be returned to the starting position and orientation. The agent's goal is to then navigate to the goal area in fastest time possible, minimizing actions (time steps) taken by the agent.

The score awarded to the robot for each maze is the sum of the two following terms:

- The number of steps taken while exploring the maze during the first run divided by 30.
- The number of steps taken to reach the goal during the second run of the maze.

Each maze is limited to a maximum of 1000 steps for both runs combined.
A virtual robot with a smaller score performs better than one with a larger score.

It is important to note that this scoring metric penalises both robots that fail to find the optimal path to the goal as well as those that explore the maze in an inefficient manner. That said a robot that limits exploration and fails to find the optimal path to the goal is likely to, but may not necessarily, perform worse than a robot that takes the time to explore the maze sufficiently and finds the optimal path to the goal.

Project Design

These are the steps to be followed as part of the project:

- Analyze the given starter code - robot.py, maze.py, tester.py, showmaze.py.
- Setup the mazes using sample mazes through test_maze_##.txt and check for walls upon robot movement or sensing using maze.py.
- Visualize the maze using showmaze.py.
- Modify robot.py by adding appropriate algorithms and functionality suited for exploration and optimization trials.
- Run the benchmark model and record the performance.
- Run the optimized model and record the performance.
- Test the results using tester.py and graphically visualize the robot performance in each maze.
- If the robot does not find an optimal solution, then revise robot.py and repeat the process until it finds the optimal solution.
- Document the findings by producing a report.

Some of the algorithms which will be considered for this project are:

- Random Mouse Algorithm: is a trivial method that can be implemented by a very unintelligent robot or perhaps a mouse. It is simply to proceed following the current passage until a junction is reached, and then to make a random decision about the next direction to follow. Although such a method would always eventually find the right solution, this algorithm can be extremely slow.
- Breadth-First Search: is an algorithm for traversing or searching tree or graph data structures. It starts at the tree root and explores the neighbour nodes first, before moving to the next level neighbours.

- Depth-First Search: is an algorithm for traversing or searching tree or graph data structures. One starts at the root and explores as far as possible along each branch before backtracking.
- Dijkstra's: 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.
- A*: is an informed search algorithm, or a best-first search, meaning that it solves problems by searching among all possible paths to the goal for the one that incurs the smallest cost (least distance travelled, shortest time, etc.), and among these paths it first considers the ones that appear to lead most quickly to the solution.
- Flood Fill: also called seed fill**,** is an algorithm that determines the area connected to a given node in a multi-dimensional array. The flood-fill algorithm takes three parameters: a start node, a target color, and a replacement color. The algorithm looks for all nodes in the array that are connected to the start node by a path of the target color and changes them to the replacement color.

References

Capstone Project Link

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

Micromouse – Wikipedia

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

Maze Solving Algorithm – Wikipedia

https://en.wikipedia.org/wiki/Maze_solving_algorithm

Motion Planning Algorithms

https://en.wikipedia.org/wiki/Motion_planning#Algorithms

Breadth-First Search Algorithm

https://en.wikipedia.org/wiki/Breadth-first_search

Depth-First Search Algorithm

https://en.wikipedia.org/wiki/Depth-first_search

Dijkstra's Algorithm

https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm

A* Search Algorithm

https://en.wikipedia.org/wiki/A*_search_algorithm

Flood Fill Algorithm

https://en.wikipedia.org/wiki/Flood_fill