Proposal for Robotic Motion Planning

Long Wan

Domain Background

This project was inspired from Micromouse competitions, which became popular in late 1970s. In this competition, there was a maze with 16 by 16 grid of cells, where a mice, actually a robot, tried to find the fastest way from its original place to the central point of the maze. The robot would collect data throughout a number of trials and attempt to reach the target with what it had learned. In this process, various searching algorithm might be used to train the robot. Previous algorithms used includes Bellman Flood-fill method, Dijkstra's algorithm and many others regarding tree traversal.

The Micromouse issue could be regarded as the miniature of self-driving system, to find the optimal path, to avoid collision and to learn based on data collected. So to try designing the method to help the mice find its way is of great importance to start a self-driving career. For me, I will strive to accumulate hands-on experience of designing algorithm and data processing, and further strengthen my understanding of machine learning and robotics.

Problem Statement

Robot is expected to start from the bottom-left corner of the maze, and is supposed to move to the center of the maze. We should find the best algorithm for the robot to learn and find the optimal way to reach the goal.

The maze designed in the project will be in an n*n dimension, where n is even, so that the center of the maze could be a 2*2 enclosed area with only one open exit. Walls in the maze can block the way the robot goes so the robot should change their direction to avoid collide with walls. The starting cell is enclosed by 3 walls, therefore, the robot can only move to one direction (upward here) at the beginning. The entire maze is fully enclosed and the robot will not go outside the maze. Mazes are provided via text file in the format of 4 digit binary number, representing the existence of walls from 4 directions, upwards, right, bottom and left side, respectively. For example, 0101, which equals to 5, means that there are walls on the right hand and left hand side, and ways are open upwards and downwards.

The robot is designed to perfectly take up only one cell. There are 3 sensors embedded inside the robot indicating the number of open cells in front of the left, right and front side of the robot. For each time, result of sensors will tell the robot opened ways, and then the robot will choose to turn left or right if there is a wall in front of it, or move forward if there are walls on the two sides. Each movement should be 1 cell and takes 1 unit time. The maximum consecutive movements is 3 between two sensor readings. Robot might turn exactly 90 degree clockwise or counterclockwise when walls block it. If the robot tries to go into the wall, it will just stay at the same location.

The robot will start by exploring the maze. It will travel to somewhere and collect information regarding the maze. After it reaches the goal, it will proceed to the next run again and again, until it finds the optimal path which minimize the total time spent.

So generally, the robot will receive 3 numbers from sensors and pass them along to the "next_move" function. The "next_move" function returns two values, rotation(-90, 0, 90 representing left, forward and right) and movements(-3 to 3, where negative values represent backward and positive values represent forward). Once the robot enters into the center area, it will be regarded as a successful run. We should design an algorithm to have it reach the goal in the shortest way. Potential algorithms have been covered in Domain Background above.

Datasets and Inputs

The nature of the project does not require any data inputs, but in the process of learning and optimization, some important data are collected and updated, like the location of the robot, walls information, action of robot, knowledge of the maze and so on.

Location of robot: normally an element of an array, like [0,0] representing the bottom-left cell.

Walls information: It has been covered in the previous part.

Actions: contains movement(-3 to 3) and rotation(-90, 0, 90)

Knowledge of the maze: center area location, dimension, walls distribution, how far away from a cell to the target.

Solution Statement

First of all, the robot starts from the corner of the maze, reading the sensors values and choosing the direction to move. As it move forward, the robot will gradually learn what the maze looks like, recording its structure, like walls distribution and consecutive open spaces at one direction. Also it will learn where the goal is and the way to enter into the center grid as it will stop only if it finally find the goal for the first time. In this process, learning the structure of the maze as much as possible even though time spent is long is a good idea since the penalty for the first run is lower. So taking A star algorithm is a good choice, which explores the maze according to the distance between current location and the goal. It will go backward if the new location has higher distance than other choices have. Time might be longer as it includes repeated location in its paths, but it is totally fine given that time is not an important restriction and evaluation criterion for the first run.

Next, after initially exploration, the robot should try to find an optimal path in a more efficient way in order to lower down penalty. Dijkstra's algorithm is probably a good option to find out the fastest route. Based on previous exploration, there should be a graph reflecting the maze structure, and Dijkstra's algorithm will follow the graph and pick up the best path as soon as possible.

Evaluation Metrics

There are two important steps for this project, exploration and optimization, and weights of them are different on the measurement score. Weight for exploration runs should be much lower than following optimization runs. So evaluation metric is the aggregate value of following scores,

1. The total number of steps taken in exploration process divided by 30. The exploration process might be run again if the robot did not find the target within the upper limits of steps, like 500, or 1000.

2. The total number of steps takin in optimization processes. This process will not stop until the robot thinks it finds the optimal path.

3. If the robot did not reach the goal within the step limits, there will be an extra penalty of 5 for exploration process and 5/30 for optimization process. It will then restart again.

The lower the score, the better.

Benchmark Model

The benchmark model I choose is the random model. The robot will avoid running into a wall according to sensors information but it chooses direction and movement steps randomly when needed.

The benchmark model can be evaluated by the metrics. Once the robot find the target for the first time, it will then move forward to optimization process. Steps can be recorded and extra penalty can be calculated. We can set an upper limit of attempts to let it stop.

Project Design

The project will be built under Python 2.7 with Numpy package. There are several starter codes provided here in an archive:

https://drive.google.com/file/d/0B9Yf01UaIbUgQ2tjRHhKZGlHSzQ/view

Starter codes include:

Robot.py: py script that establishes the robot class.

Maze.py: script that constructs the maze and checks for walls upon robot movement or sensing

Tester.py: script that test robot's performance on navigating mazes.

Showmaze.py: plot a map of the maze

Test_maze_##.txt: mazes of different dimensions to test robot.

My workflow mainly includes as followed,

- Modified robot.py and insert my own algorithms into it.
- Explore and discuss how the robot will explore the maze, and describe the structural characteristics.
- Run the algorithm with Benchmark model and record its performance.
- Run the algorithm with designed models and record its performance.
- Test the result with tester.py in three mazes of different dimensions given what the robot was trained.
- Visualize performances of the robot in different mazes within this algorithm.
- Further improve the algorithm, modified some arguments, and redo processes above, until it reaches the optimized point.
- Write down the whole process and complete the final report