Capstone Project

Machine Learning Engineer Nanodegree

Yangfeng Lian
August 20th, 2016

Plot and Navigate a Virtual Maze

I. Definition

Project Overview

I programmed in this project is a robot mouse finding a optimal way to the destination in a virtual maze.

Problem Statement

The robot mouse can run twice in the virtual maze. The virtual maze is a square with even number such as 12,14,16 side length. The destination is a 2*2 square in the center of the virtual maze. In each run, the robot mouse starts at the left down corner of the virtual maze, it tries to map out the maze to not only find the center, but also figure out the best paths to the center. In second run, the robot mouse attempts to reach the center in the fastest time possible, using what it has previously learned. In this project, I will create functions to control a virtual robot to navigate a virtual maze. My goal is to obtain the fastest times possible in a series of test mazes.

Metrics

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. A maximum of one thousand time steps is allotted to complete both runs for s single maze.

II. Analysis

Data Exploration And Visualization

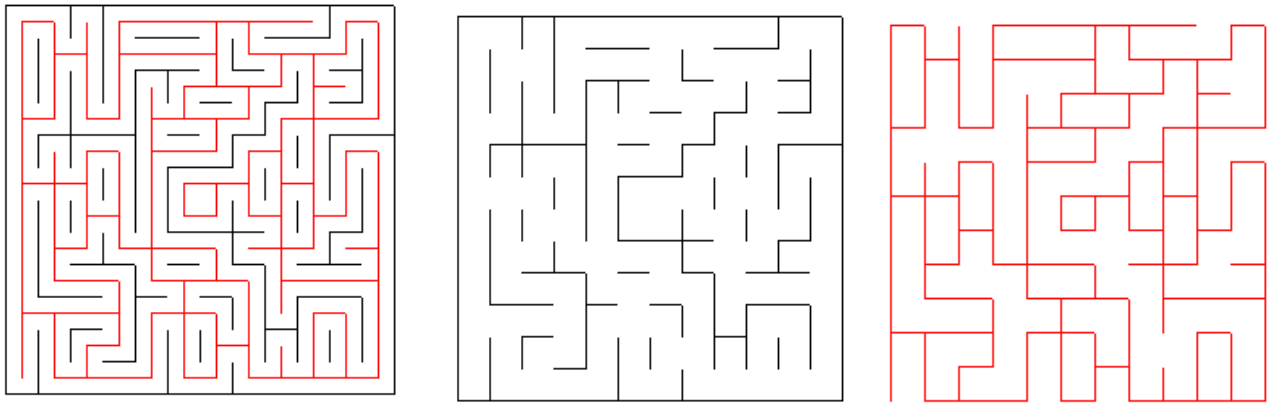
The data of this program is a *n* x *n* number array, each number represents one cell of the maze, Each number represents a four-bit number that has a bit value of 0 if an edge is closed (walled) and 1 if an edge is open (no wall); the 1s register corresponds with the upwards-facing side, the 2s register 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. For example the data of first maze is:

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

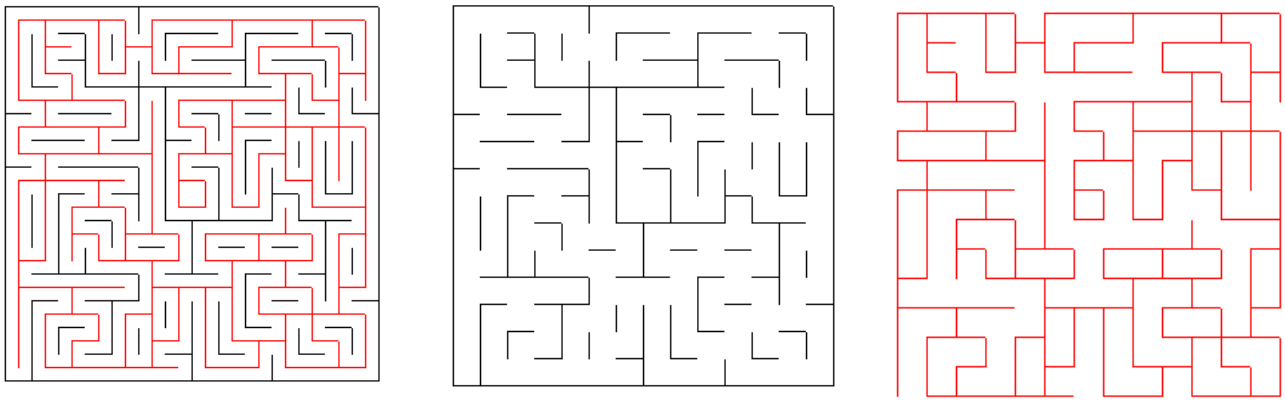
10,9,13,12,3,13,5,12,9,5,7,6
9,5,6,3,15,5,5,7,7,4,10,10
3,5,15,14,10,3,6,10,11,6,10,10
9,7,12,11,12,9,14,9,14,11,13,14
3,13,5,12,2,3,13,6,9,14,3,14
11,4,1,7,15,13,7,13,6,9,14,10
11,5,6,10,9,7,13,5,15,7,14,8
11,5,12,10,2,9,5,6,10,8,9,6
9,5,5,13,13,5,5,12,9,5,5,12

We can see that the number array is not intuitionistic. We are hard to image the maze looking at the data. So I modify showmaze.py and use it to draw the maze. I use black line to draw walls of the maze and use red line to draw the route of the maze.

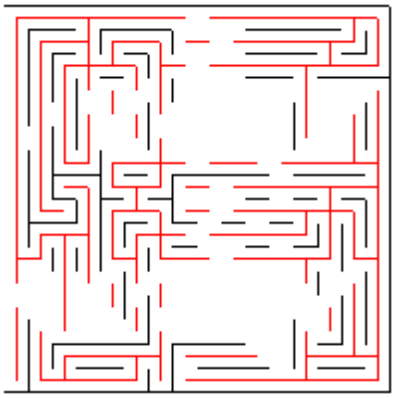
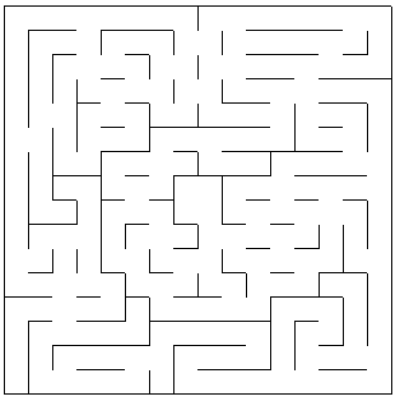
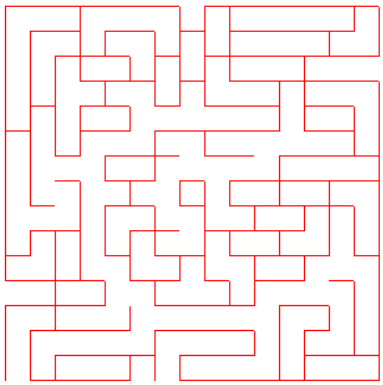
Maze01:



Maze02:

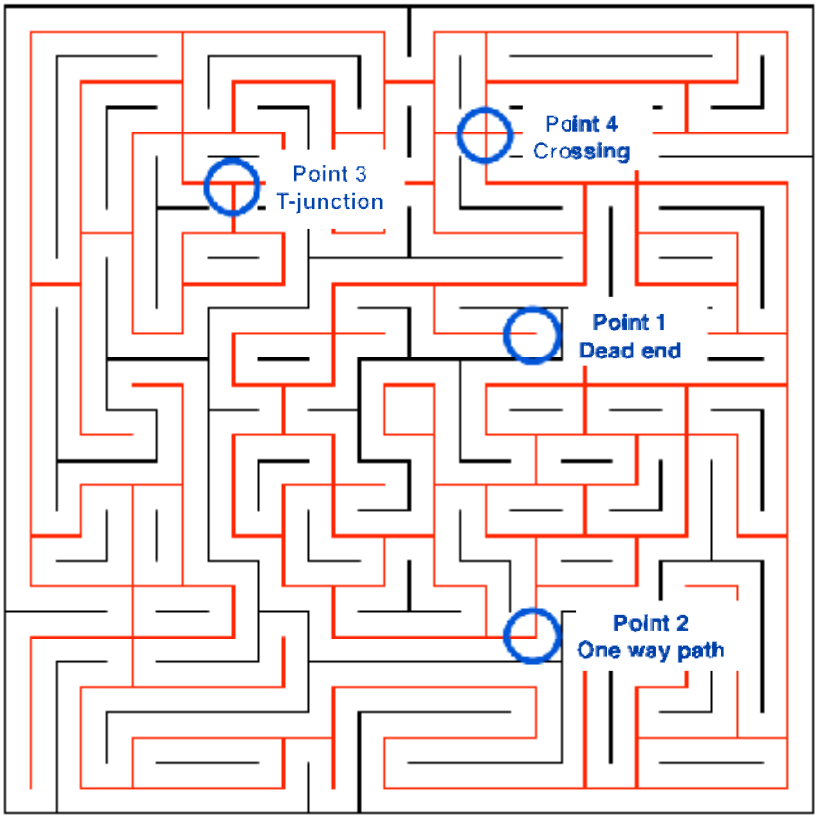


Maze03:

The robot has no eyes, it has three obstacle sensors at its left, front and right side. At each step, the three obstacle sensors will tell the robot how far it's from the corresponding side wall. According to the sensors data, the robot should decide its steering, both rotation and movement. The rotation has three options, turning left, going straightly and turning right. The movement has seven values, {-3,-2,-1,0,1,2,3}, the negative value means moving back, the positive value means moving forward, the number means how many steps it moves.

According to the right maze image, we can see that all the points in the maze can be classified into four classes corresponding to four values of total sides count of facing wall directly.



For example,Point 1, if the robot is in a dead end point, it will receive a sensors data of {0,0,0}, the values of total sides count of facing wall directly is three.

Point 2, if the robot is in a path with only one way to go excluding moving back, it will receive a sensors data like {0,0,positive number} or {0,positive number,0} or {positive number,0,0}, the values of total sides of facing wall directly is two.

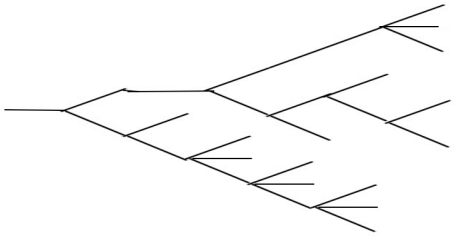
Point 3, if the robot is in a T-junction, it will receive a sensors data like {0,positive number,positive number} or {positive number,0,positive number} or {positive number,positive number,0}, the values of total sides count of facing wall directly is one.

Point 4, if the robot is in a crossing, it will receive a sensors data like {positive number,positive number,positive number}, the values of total sides count of facing wall directly is zero.

Algorithms and Techniques

If the robot explores the entire maze,we can use A* algorithm to fine the shortest path from the start location to the goal area. Therefore,in the first run,the robot should try exploring the maze as much as possible and expand

the mapping area so that, in the second run, the robot can apply the A*search algorithm to find the optimal path and moves. On the other hand, the robot should finish exploring the whole maze as fast as possible in the first run as it affects the score value and if too much time is spent in the first run, it may not be able to leave enough time for the second run.



Suppose that, if the map is of tree structure like the right image rather than maze structure. The robot start exploring at the leftmost point, it has a simple strategy to explore the whole map, it just need simply go along the right side or left side wall all the time until return to the start point. For example, if it go along the right side wall, it will keep the last value of sensor data zero. At any point of the map, if need deciding witch path to go, choose the rightmost path. If reach a dead end, turn left twice. Else situation, go along the path.

In this program,I classified the all points of the map into four classes, dead end, one way point, T_junction and crossing. At each class point, I make a strategy to choose the path, then the map will be transformed from maze structure to tree structure and explored by right wall strategy like the tree structure.

My dead end and one way point strategy of maze structure is same as the tree structure. My T-junction and crossing strategy of maze is different from the tree structure. I will specify details in the next chapter.

After exploring the whole map, I use the A* algorithm to find a optimal way. I read A* algorithm from wiki, the link is https://en.wikipedia.org/wiki/A*_search_algorithm. My A* algorithm has some differences from the wiki one, I will specify details in the next chapter.

Benchmark

I'm allowed one thousand time steps for exploring (run 1) and testing (run 2),and given the metric defined in the project. I think a good performance is that explored the whole maze as quickly as possible in run 1 and move through a fastest route in run 2.

III. Methodology

Data Preprocessing

Because the maze specification and robot's sensor data is provided and 100% accurate. Therefore, there is no data preprocessing needed in this project (the sensor specification and environment designs are provided).

Implementation

In the robot.py, first, I define seven static dicts to get a quick enquire. The dir_sensors is used to enquire the orientations of three sensors corresponding orientation of robot's heading. The dir_move is use to translate the orientation string to axis value list. The dir_reverse is used to get the reverse orientation. The dir_compare is used to get the rotation. The dir_value is used to get the value of every point of maze data corresponding to the orientation. The dir_heading is used to translate axis values to orientation string. The dir_rotation is used to translate rotation string to values.

I add some attributes and methods to the Robot class. I define Junction_3 as T-junction and junction_4 as crossing.

Junction_3 pseudocode:

```
If not in junctions list:
    Create junction_3
    Go to right branch
Else:
    If oncoming is right branch:
        If left branch has been visited :
            Go back to the path where it first reach the junction from
        Else:
            Go to the left branch
    Else if oncoming is left branch:
        If left branch has been visited :
            Go back to the path where it first reach the junction from
        Else:
            Turn back
            Set the left branch visited value to True.
```

Junction_4:

The crossing is more complex than T-junction. The crossing has two modes. Mode 1 is that after creating the crossing object, the following time the robot reach the crossing again is by going back from the right branch. Mode 2 is that after creating the crossing object, the following time the robot reach the crossing again is by the left branch or forward branch. Below is the pseudocode:

```
If not in junctions list:
    Create junction_4
    Go to right branch
Else:
    If oncoming is right branch:
        If mode value has not been set :
            Set mode 1
            Go to forward branch
            Set forward branch visited value to True
        Else if mode is 2:
            Go back to the path where it first reach the crossing from
    If oncoming is forward branch:
        If mode is not set:
            Set mode to be 2
```

Go to left branch

Set left branch visited value True.

Set forward branch visited value True

Else if mode is 1:

If left branch is visited :

Go back to the path where it first reach the crossing from

Else:

Go to the left branch

Set the left branch visited true.

Else if mode is 2

Go to left branch.

If oncoming is left branch:

If mode is not set:

Set mode 2

Go to forward branch

Set left branch visited true

Set forward branch visited true

Else if mode is 1:

If left branch has not been visited:

Turn back

Set left branch visited true

Else if mode is 2:

Go to forward branch.

During the first run exploring, I use the to_point_value method to get the maze data and save them in cell_points array attribute. At the second run, I use the cell_points array attribute and A* algorithm to find the optimal way.

My A* algorithm has some differences from the wiki's A* algorithm. First, the goal in this program is an area rather than a point, so I define a manhattan_dist_to_goal function to calculate the H score. Second, the calculation of G score is more complex than the wiki's A* algorithm. I define two function to calculate the G score. The straight_step_num function is used to count how many more straight steps has been taken, because the maximum value of movement is three. The is_straight function is used to check if the neighbor point is a continuous straight movement.

Refinement

After running the A* algorithm for several times, I found that the algorithm can't get the optimal path almost. I think the reason is that the algorithm will be halted if the robot reach the goal area, but at that moment there may be some more optimal paths has not been searched, they are still in the open set. So I define the repeat_a_star function to repeat the A* algorithm for several times and return the fastest path.

IV. Results

Model Evaluation and Validation

The optimal moves are measured by using the A* search.

Test maze	Path Length	Required Moves	My A* moves
01	30	17	17
02	43	23	23
03	49	25	25

My scores are that:

Maze	01	02	03
Score	28.933	39.367	47.067

Justification

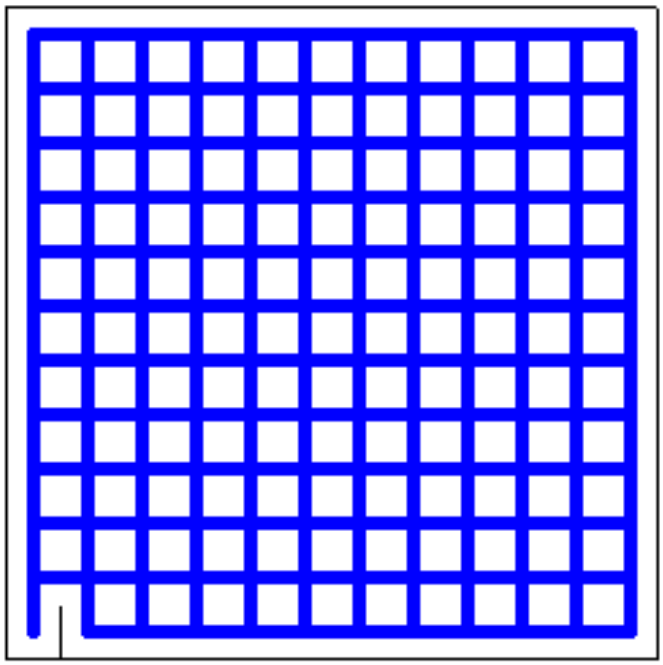
My program has found the optimal path in each maze and performed a good score although the score is lower than the article of https://github.com/udacity/machine-learning/blob/master/projects/capstone/report-example-3.pdf.

V. Conclusion

Free-Form Visualization

I run the program in another maze(maze 4) almost without any walls, the robot make a success at exploring the whole maze and find a optimal way in the second run. The maze data is following:

1,7,7,7,7,7,7,7,7,7,6
3,15,15,15,15,15,15,15,15,15,14
11,15,15,15,15,15,15,15,15,15,14
11,15,15,15,15,15,15,15,15,15,14
11,15,15,15,15,15,15,15,15,15,14
11,15,15,15,15,15,15,15,15,15,14
11,15,15,15,15,15,15,15,15,15,14
11,15,15,15,15,15,15,15,15,15,14
11,15,15,15,15,15,15,15,15,15,14
11,15,15,15,15,15,15,15,15,15,14
11,15,15,15,15,15,15,15,15,15,14
9,13,13,13,13,13,13,13,13,13,12



Reflection

This program is a true challenge for me. At the beginning, I barely think of the random movement. But the random movement get a bad performance, it even sometimes can’t reach the goal in the first run. So I keep searching how the robot can explore the maze in order for several days. Luckily, I got the solution.

Another challenge is A* algorithm, I have never seen A* before. I browsed many articles about it, but still can’t understand. Until thinking about it for a long time, I understand it and master it. According to the quality of the program, I modify the A* algorithm and get the optimal path.

Improvement

In this program, everything (time, location, move and turn) is in discrete domain. In the real micro mouse competition, everything is in a continuous domain.

For example, the distance from the robot to the wall is measured in continuous value with some sensor errors. The robot movement itself would have some randomness. Therefore, the robot would need to perform SLAM(simultaneous localization and mapping) to explore the maze. Moreover, the robot needs to use PID control to continuously adjust the direction and turns so that it can wander around in the maze without colliding with the walls. The speed needs to be controlled rather than just number of steps. Turns will be continuous rotations. Moreover, the robot may be able to move diagonally rather than zigzag which is not allowed in the discrete domain.

Talking about the real micro mouse competition, the fact that the robots are physical adds many more complexity. The path finding logic is probably one of the easiest part of the whole robot construction. There are many aspects to take care in physical robots; what sensors to use, what kind of motors and how heavy it can be, how much memory size available to use, etc. Maybe I could have a sensor rotating on top of the robot mapping neighboring areas simultaneously just like a google car. The possibilities are endless.