MLND Capstone Project Proposal

Robot Motion Planning in a Maze

Qingyuan Ji

November 8th, 2017

Domain Background

The micromouse competition is an annual event in the robotics industry where a small robot mice is asked to navigate through a small sized maze (Misha et al, 2008). The maze is usually a space consisting of 16 by 16 cells, where each cell is about 180 mm square. Walls exist between cells so that a cell is only allowed to be entered and leave in limited directions. Figure 1 show an example of a maze.

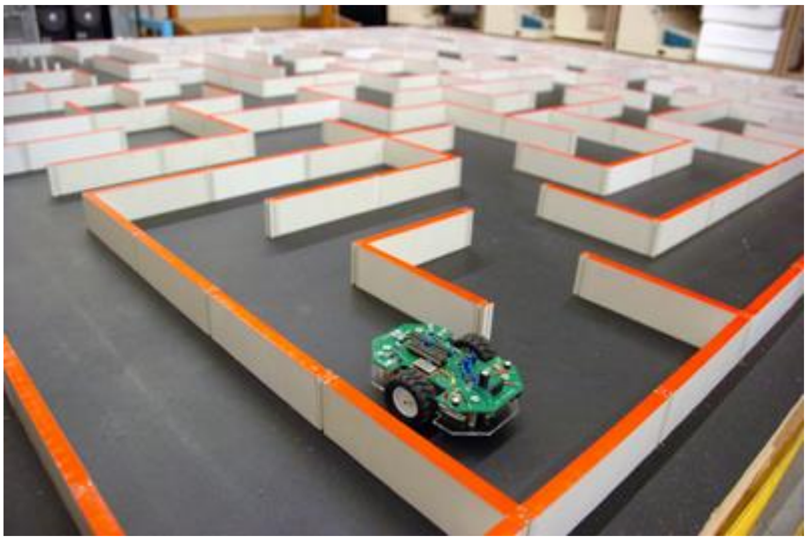

Figure 1. A maze example for the micromouse competition (source: Google Images).

A robot mouse should be completely autonomous and heuristic to be able to find the correct path from the predetermined starting point to the goal (normally the central of the maze) without external help or command. The key task for the robot mouse is to roam around the maze with no prior knowledge, detect the location of walls blocking its way, and keep track of where it has explored and where the goal is. Finally, the robot should be able to somehow map the maze and find the optimal path leading to the goal, from the starting point.

This competition starts from 1970s, and is still being hold every year nowadays. Despite being around 40 years old, its importance in the field of robotics is still unparalleled, as it requires complete analysis of unknown space and proper path planning and decision making. Different algorithm such as Dijkstra (Gupta and Segel, 2014) and A* search (Ambeskar et al, 2014) haven been extensively applied and modified to tackle this issue. This project is inspired by the micromouse competition, where I plan to implement the simplified mazes and robot mouse digitally to find a model allows the robot mouse to reach the goal in the fastest way.

Problem Statement

The problem I want to solve is that, given mazes of different layouts and different models, let the robot mouse beat the mazes as fast as possible. For example, figure 2 shows a specific layout of maze that will be used in the project.

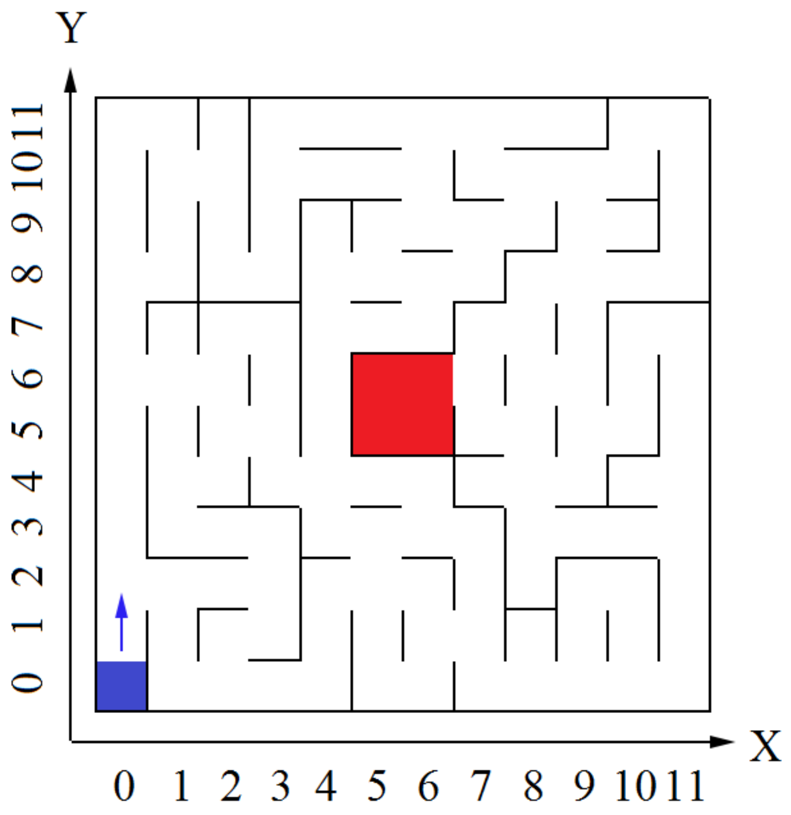

Figure 2. A sample maze (12×12) that will be used in the project.

In figure 2 is a 12×12 maze, which means this maze consists of 144 grid cells. Since the maze is a 2D space, it can be described by a 2D coordinate system, and each cell has its own coordinate, for example, coordinate of cell on the top right is (11, 11). Walls exist in the maze to block the entry into a cell from a specific position. For example, it is impossible to travel from the cell (1, 0) to (0, 0), because there is a wall between them.

The rule for the robot mouse is as follows

The robot mouse is given two runs totally to solve a specific maze. At the start of each runs, it will always be placed at the bottom left cell (the blue cell) whose coordinate is (0, 0). Moreover, the mouse will initially face north (the blue arrow). The mouse is equipped with sensors on the forward, left, and right side to know if currently there are wall on these sides. For example, at the starting point, the mouse detects walls on the left and right sides, but no wall on the forward side.

In the 1st run, the mouse knows nothing about the maze and needs to start exploring it. A mouse is allowed to take a step from its previous position each time. A step consists of two sequential actions: **rotation and movement**. A rotation can be clockwise 90 degrees, counter-clockwise 90 degrees or no rotation. A movement can be moving forward or backward for 0, 1, 2 or 3 units, where one unit is the distance to move into the adjacent cell. **Note this simplified maze world is discrete, and rotation cannot be things like 45 degrees clockwise or counter-clock**

wise, and the movement cannot be 1.5 or 2.5 units. For example, in figure 2, from its starting location, a valid step for this mouse is to not rotate at all and then move forward for 2 units, where it will end in the cell (0, 2). The goal is depicted in the red areas (consisting of 4 cells). In the 1st run, the mouse is given a limited steps (1000) to move from the start location, to explore the maze and try to reach the goal. If the mouse reaches the goal and still has remaining steps, it can continue explore the maze freely. In the 2nd run, the mouse still starts from the same cell, but this time, it will try to reach the goal as fast as possible, based on knowledge it receives in the first run.

The performance of both runs (the steps used in both runs) will be used to judge how fast a mouse solves a maze.

Datasets and Inputs

The datasets and inputs come from the curated Udacity Machine Learning Nanodegree project - Robot Motion Planning Capstone Project, where three mazes of pre-defined layout are given to be used in this project. The size of the mazes are 12×12, 14×14, and 16×16, which are shown in figure 3. In each of the mazes, the goal area (consists of 4 cells) is depicted in red, and the starting position for the robot mouse is the blue cell.

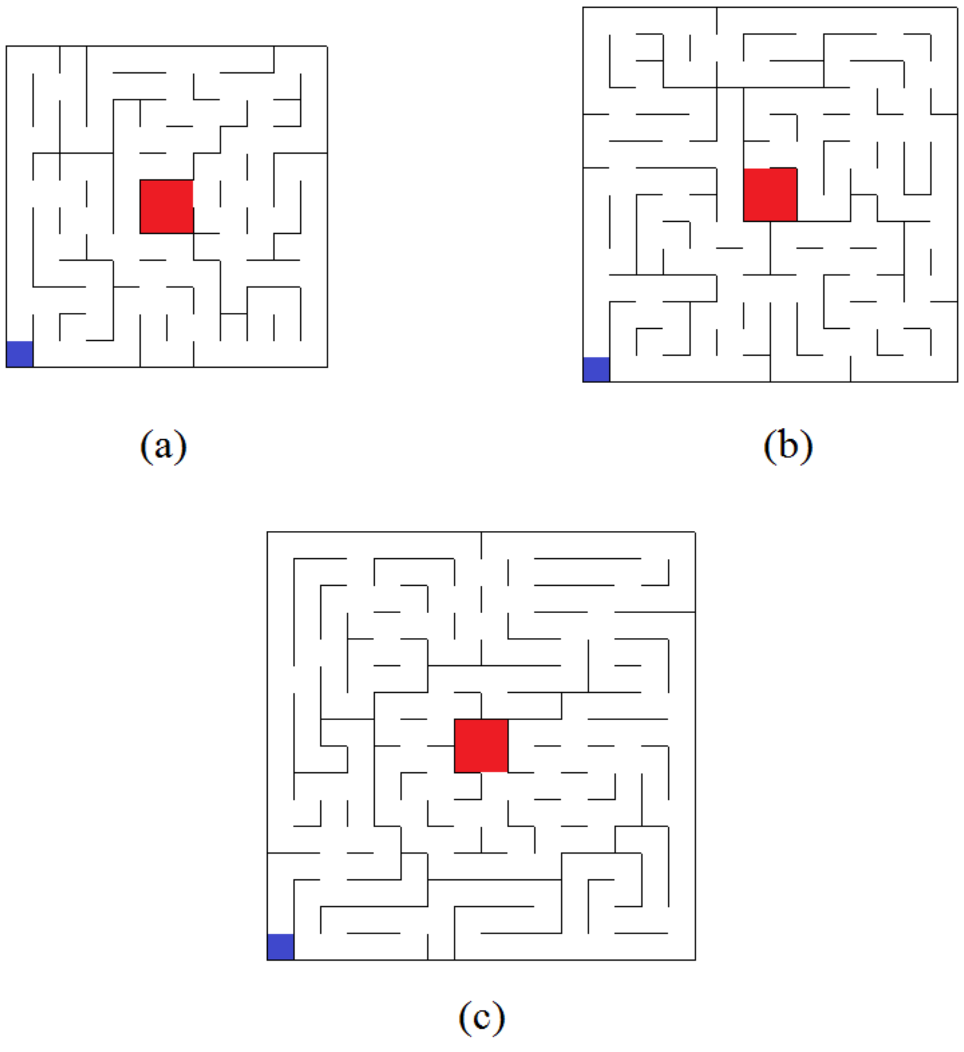

Figure 3. Specific Mazes used in this project, which are 12×12 (a), 14×14 (b), and 16×16 (c).

Solution Statement

As mentioned in the last section, 1st run and 2nd run needs to be dealt with separately, since they are different problems. In the 1st run, the robot mouse tries to find the goal and uncover the maze as much as possible. Then in the 2nd run, it tries to map the maze, and find a shortest path.

For the 1st run, I would like to implement a random movement controller. That is given the current position, and its sensor data, the robot mouse randomly picks up a valid movement (in other words, not bashing against the wall). Moreover, that controller has a preference that if there are multiple valid movements each time, then it picks up a movement to go into a cell that has never been visited or least visited. Hopefully, using this strategy, our robot mouse will be able uncover the maze as much as possible and find the goal area.

For the 2nd run, it will be a little easier. Let's suppose our robot mouse has gained information of the maze in the 1st run, then it would be able to map the maze. Here, inspired by the Udacity Artificial Intelligence for Robotics course, I would like to use the dynamic programming (Ni et al, 2013) here . That's is, once the maze is able to be mapped, I create a path grid of the maze to represent how distant each cell is to the goal area. And then I create a policy grid to tell the optimal movement for the cell, it the robot mouse wants to reach the goal as soon as possible. Finally, I just need to apply this policy grid to find the shortest path from the starting point.

Evaluation Metrics

The metrics for this project is quite simple, which is the one thirtieth (1/30) of the 1st run steps plus the 2nd run steps. A good model should have this result as small as possible. The reason for doing this is that we want to use steps in both runs, and we actually add a weight here. That is, if the robot mouse spends quite a long time to explore the maze in the 1st run, but is really fast in the 2nd run, it is still considered to be a good model. As long as long time exploration pays off, it is worthwhile.

Benchmark Model

The robot mouse are given two runs to solve a maze. Therefore, in the 1st run, it should try to discover the maze as much as possible even after it has reach the goal and still has remaining steps. Once it has discovered the entire maze or a majority of it (say, around 70% to 90%), it should be able to get the optimal (or the almost optimal) path from the starting location to the goal. Therefore I would argue that a benchmark model could be that the robot spends about 700-900 steps to explore the maze in the 1st run (depending on the maze size as well), and then spends around 20-30 steps as the short path in the 2nd run (depending on the maze size as well). Therefore, using our metrics defined earlier, the benchmark score would be between 43.3 and 60.

Project Design

First, the robot mouse is given three different mazes (12×12, 14×14, and 16×16), which are showed in figure 3. Then I used the random movement controller and dynamic programming to command the robot mouse in the 1st run and 2nd run. Due to the randomness of random controller in the 1st run, for each specific maze, I need to let mouse solve it multiple times (say, 10 times) and use the average score to judge its performance. In the end, I will construct a customized but more difficult maze to test if our algorithms works also fine.

Reference

Ambeskar, A., Turkar, V., Bondre, A., & Gosavi, H. (2016, August). Path finding robot using image processing. In *Inventive Computation Technologies (ICICT), International Conference on*(Vol. 3, pp. 1-6). IEEE.

Gupta, B., & Sehgal, S. (2014, September). Survey on Techniques used in Autonomous Maze Solving Robot. In *Confluence The Next Generation Information Technology Summit (Confluence), 2014 5th International Conference*- (pp. 323-328). IEEE.

Mishra, S., & Bande, P. (2008, November). Maze solving algorithms for micro mouse. In *Signal Image Technology and Internet Based Systems, 2008. SITIS'08. IEEE International Conference on* (pp. 86-93). IEEE.

Ni, Z., He, H., Wen, J., & Xu, X. (2013). Goal representation heuristic dynamic programming on maze navigation. *IEEE transactions on neural networks and learning systems*, *24*(12), 2038-2050.