Machine Learning Engineer Nanodegree

Capstone Proposal - Robot Motion Planning

Edward Kish

August 18th, 2017

Proposal

Domain Background

In order for a robot to move around in its environment, it needs to do several things. It must:

- Track its own position.

- Translate sensor data into a map of its surroundings.

- Find a path to its objective.

Additionally, if the area is not already mapped out, the robot must explore the area and develop its own map. Each of these operations is critical for a robot to safely and effectively move through its environment. Thus, it is important to develop algorithms to accomplish these objectives. A good way to experiment and explore possible solutions to the problem of motion planning is by simulating a robot and its environment. This project is inspired by the Micromouse[1] competitions, where a robot must explore and solve a maze.

Problem Statement

The problem here is to implement an algorithm that will allow the simulated micromouse robot to solve the maze as quickly as possible. The robot gets two runs: one in which to explore the maze and determine an optimal route, and a second run in which to move from the starting location to the goal in the shortest number of steps. The score will be the number of steps required to reach the goal in the second run plus 1/30th of the number of steps used to explore the maze in the first run. The simulated robot should be able to solve any arbitrary maze presented to it. The mazes consist of discrete points, and the movements executed by the robot are

Datasets and Inputs

The input data for this project takes the form of a maze. In micromouse competitions, mazes are 16x16 squares in size. The test mazes provided here are 12x12 and 16x16 in size, but mazes of any arbitrary size are possible. It will be desirable to generate more testing mazes with specific features to test the performance of the simulated robot.

At each step, the robot receives sensory input. It can sense the distance to the nearest wall in three directions. From this, it is possible to determine which other squares can be accessed from the current square. After receiving the sensory input, the robot can execute a movement command consisting of a rotation and a movement distance. The micromouse can turn 90 degrees left or right, and move up to three spaces forward or backward. In this way, the mouse can move from one square to another. The robot has no way to measure its absolute position in the maze. Instead, we must rely on 'dead-reckoning' and keep track of the position based on previous movement commands. If a command is invalid, or the robot bumps into a wall, this could result in the robot losing track of its position.

One of the sample mazes is shown below. The robot starts in the bottom left, and must find its way to the goal areas in the center. After the first run is completed, the second run starts again from the same location at the bottom left corner.

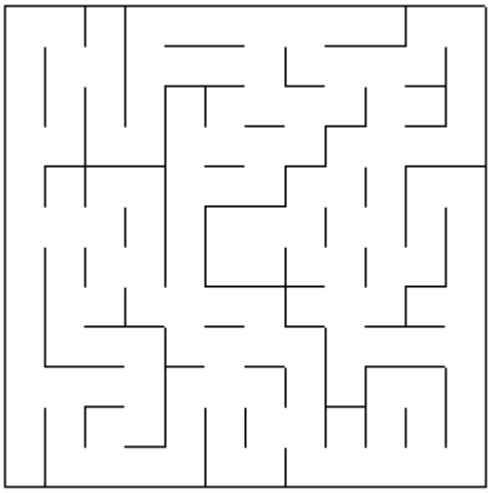

Fig 1: Sample maze defiend in 'test_maze_01.txt'

Solution Statement

The first run must be used to explore the maze and use the sensory inputs to build a map. This will require the implementation of a Simultaneous Localization and Mapping (SLAM) algorithm. At each step, the sensory inputs can be used to see which other squares can be accessed from the current square. The robot must choose new squares to move to in order to explore and map the maze. Once the maze has been mapped, finding the optimal path from the start to the goal is simple. Since the movement cost to move between one square and another is uniform (one time step), we can find the shortest path with a simple breadth first search. The mazes are on the order of 15x15, so the running time of BFS should not be a concern.

Benchmark Model

A simple benchmark robot would simply choose a random, valid direction to move. The number of steps it takes for a completely random walk to reach the goal should serve as a bare minimum standard

of performance. If a robot cannot beat a random walk through the maze, then the algorithm is very poor indeed. The goal will be to minimize the score calculated from the length of the two runs through the maze. Although the length of the second run is most heavily weighted by the scoring metric, it is the first run that needs to be most heavily optimized. Perhaps a better benchmark is to randomly explore the map on the first run, and then have the robot follow the shortest available path. Once the maze is mapped out, there is one optimal path length and that will be simple to find. The real challenge is to find a way to most efficiently explore the maze.

Evaluation Metrics

The scoring metric for this project is defined in the supplemental material[2] as follows: *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 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 are allotted to complete both runs for a single maze.*

This is a simple and objective measurement of the robot's performance. If the robot's behavior is at all random or stochastic, then it will be desirable to find an average score over many runs of the same maze to get a more accurate assessment. There are three test mazes provided, and these will be used to score the simulated micromouse robot. But it will also be possible, and desirable, to generate additional test mazes.

Project Design

This project can be broken down into to several parts. The goal is to simulate a micromouse robot that can efficiently solve a maze. This simulation will have to do several things.

- Take sensor input

- Convert sensor input into a map of the maze

- Decide how to explore the maze

- Build a complete map based on sensor data as it explores

- Convert a planned movement into a rotation and distance command

- Decide when to reset and begin the second run

- Plan and follow the shortest possible path to the goal

The final product will be a python class that implements all this functionality. In addition, it will be useful to visualize both the maze, and the path of the robot as it explores and then solves the maze.

These points describe the basic structure of the simulated robot. The real challenge of the project will be determining the algorithm that most efficiently explores and solves the maze. In the literature, this problem is referred to as 'Simultaneous Localization and Mapping' or SLAM. The robot has to build a map of its surroundings while keeping track of its own position. In this project, since the robot moves on a grid, we can represent a map of the surroundings by a hash table mapping one square to all other squares which are accessible (i.e., not blocked by walls and within 3 spaces). By carefully keeping track of the movement commands sent, the robot can keep track of its own location in this map.

At each step, the robot can use sensor data to extend its map, and choose where to move next based on that map. As mentioned above, the benchmark robot would randomly choose a nearby square to move to. To succeed in this challenge, the robot will need a better algorithm for exploration. Several simple algorithms could provide better than benchmark performance. By keeping track of which spaces it has visited before, the robot can prioritize visiting unexplored sites. Or, using some distance metric such as Manhattan distance (since the maze is on a grid), the robot could choose to explore spaces closer to the goal.

There is a rich literature discussing other maze exploring algorithms thanks to Micromouse competitions. More complicated algorithms like Potential Value[3] and Partition Central[4] that have proven successful and should also be considered and compared to simpler algorithms and the random benchmark.

There is also the question of when to stop exploring the maze and when to begin the second run. The first run needs to find the goal area, but it can continue afterwards. So there is some tradeoff between further exploring the maze and maybe allowing a shorter route, or cutting the exploration short and taking the shortest route based on the explored area. The optimal choice here may depend on the exploring algorithm or even the particular maze itself.

There are a number of different algorithms to consider, and other design tradeoffs to examine. Evaluating these different choices based on the scoring metric described above will be the focus of the project. It should be possible to write a Robot parent class that implements the mapping and movement functionality, and several child classes that implement different exploration algorithms. This will allow testing and evaluation of several different algorithms. Also, in addition to the provided test mazes, it is possible to generate additional mazes to examine how different algorithms perform when confronted with specific types of obstacles.

[1] https://en.wikipedia.org/wiki/Micromouse

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

[3] Wyard-Scott, L., and Q-HM Meng. "A potential maze solving algorithm for a micromouse robot." _Communications, Computers, and Signal Processing_, 1995. Proceedings., IEEE Pacific Rim Conference on. IEEE, 1995.

[4] J. Cai, X. Wan, M. Huo and J. Wu, "An Algorithm of Micromouse Maze Solving," 2010 10th IEEE International Conference on Computer and Information Technology, Bradford, 2010, pp. 1995-2000.