Machine Learning Engineer Nanodegree
Capstone Project Proposal:
Robot Motion Planning

September 9, 2017

1 Domain Background

This project belongs to Motion Planning domain. Motion Planning is a term used in robotics for the process of breaking down a desired movement task into discrete motions that satisfy movement constraints and possibly optimize some aspect of the movement[1]. Motion planning has applications in many important domains like: Self driving cars, Robot navigation, automation, robotic surgery etc.

Based on a research paper[2], it can be concluded that in last few decades, there has been steady increase in research efforts for driver-less cars. Robotic surgery is also proven to be helpful in certain complex cases of radio-surgery, neurosurgery and cardiosurgery. All these problems, directly or indirectly, belong to motion planning domain. Correct implementation of these technologies will be an revolution in field of AI. Due to these reasons, these problems should be solved.

Personally, I am fascinated by self driving cars and aspire to contribute something valuable in this field. Robot motion planning is initial step to fulfill my aspirations. That's why I have chosen this project.

2 Problem Statement

Primary purpose of this project is to program a robot which can find a path between two given locations which is-

- efficient, if not optimal,
- satisfies all surrounding constraints.

From above statment, it can be clearly seen that the above problem is a minimization problem under some constraints, hence it can be expressed in mathematical terms. Metric to measure complexity of problem and solution,

[1]Definition taken from Wikipedia.

[2]*A Survey of Motion Planning and Control Techniques for Self-driving Urban Vehicles* by *Brian Paden, Michal p , Sze Zheng Yong, Dmitry Yershov, and Emilio Frazzoli*

could be **number of steps**. Also, this problem can be reproduced by reproducing the environment described below.

This problem is inspired from Micromouse competitions.

3 Datasets and Inputs

Inputs will be:

- Maze
- Current position
- Destination

No data preprocessing will be required in this case, because data is already in a state from where it can be imported directly.

3.1 Maze Specifications [3]

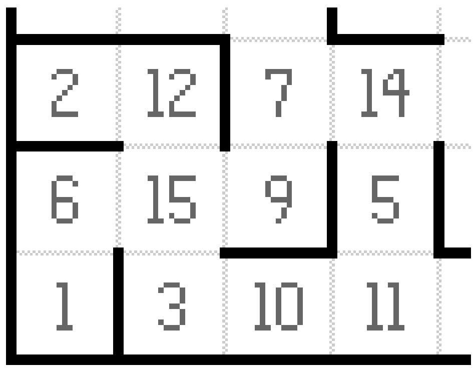
The maze exists on an n x n grid of squares, n even. The minimum value of n is twelve, the maximum sixteen. Along the outside perimeter of the grid, and on the edges connecting some of the internal squares, are walls that block all movement. The robot will start in the square in the bottom- left corner 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.

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

$$(0 * 1 + 1 * 2 + 0 * 4 + 1 * 8 = 10). \tag{1}$$

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.

[3]taken from *MLND Capstone Project Description - Robot Motion Planning.*



4 Solution Statement

In layman terms, the solution can be explained as follows. 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 [4].

This problem can be solved using search algorithms like grid search algorithm, interval based search, reward based search, sampling based algorithms and geometric search.

Some specific algorithms to solve this problems are: BFS, DFS, A-star search, Dijkstra or Uniform Cost Search Algorithm.

More details about algorithms are discussed in *Algorithms to be considered* subsection of *Project Description* section.

All of above mentioned algorithms have well defined mathematical structure and an upper bound on time complexity, hence they are quantifiable and measurable(in terms of time). Since, these algorithms are standard algorithms, solution can be reproduced easily. Hence, the solution is replicable.

5 Benchmark Model

Benchmark score can be weighted average of number of steps taken on optimal path to reach to the goal and number of steps required to traverse the whole maze in order to find optimal path. It will provide an upper bound and lower bound on performance of a model.

Mathematically, it can be expressed as

$$upper\_bound\_score = \frac{a * steps\_in\_optimal\_path + 2 * b * steps\_in\_optimal\_maze\_exploration}{a+b} \tag{2}$$

$$lower\_bound\_score = \frac{a * steps\_in\_optimal\_path + b * steps\_in\_optimal\_maze\_exploration}{a+b} \tag{3}$$

[4]Whole paragraph taken from *MLND Capstone Project Description - Robot Motion Planning.*

where a = 20 and b = 1 [5]

Assuming that dimension of maze is n X n, number of steps in optimal maze exploration will be n^2 which is same as dimension of the maze.

5.1 Explanation

Any implementation should explore whole maze by visiting each cell atleast one time and atmost two times. Exactly two times corresponds to upper bound case and exactly one time corresponds to lower bound case. If number of steps for exploration is not in these bounds, then there must be something wrong with our model.

6 Evaluation Metrics

As mentioned in Benchmark section, formula for evaluation can be

$$score = \frac{a * steps\_in\_path + b * steps\_in\_maze\_exploration}{a + b} \tag{4}$$

where a = 20 and b = 1. [6]

I have chosen this metric because it will always give a score between lower benchmark and upper benchmark model if exploration constraint mentioned in explanation subsection of benchmark model section is followed.

This metric is quantifiable as well as measurable.

7 Project Design

7.1 Algorithm to solve this problem

- Preprocess the data, if required.
- Maze exploration using some exploration algorithms, discussed in *Algorithms to be considered* subsection.
- Once maze is explored, storing them in efficient manner.
- Applying shortest path algorithm to stored maze. Algorithms are discussed in *Algorithms to be considered* subsection.

7.2 Data preprocessing

No data preprocessing is required in this case. We have maze specifications in a text file and starting and final destination. This data can be imported directly.

[5]Values are assigned intuitively

[6]Values are assigned intuitively

7.3 Algorithms to be considered

As discussed in solution statement section, following algorithms can be used to solve this problem:

For exploration, sensor information combined with some search algorithm can be used. Using sensor information and some search algorithm (say,DFS) to verify connectivity, a graph can be constructed which will serve as map for applying Dijkstra or A* algorithm. Kalman filters can be used to enhance this step to a great extent. For improving performance at this step, Kalman filter can be used. It applies Bayesian Inference to noisy measurements and develops a joint probability distribution for observed variables within a time frame. Some other techniques like Particle filters, PID controller and SLAM can be used for map exploration using information about local environment.

For finding shortest path, following algorithms can be used.

- **Dijkstra Algorithm**: Given a graph, this algorithm is used to find shortest path between two given nodes. For jumping from current node to next node, it uses priority queue data structure, where priority is decided by edge weight. Node having highest priority is selected as next destination node. This process stops when current node and destination node are same.

  Once exploration phase is completed and robot has full information about it's surroundings, it can use Dijkstra algorithm to find out shortest distance between source and destination, which can be interpreted as source node and destination node.

- **A* search Algorithm**: This algorithm is almost similar to Dijkstra algorithm. It also uses priority queue as data structure, however, now priority is decided by sum of edge weight and a heuristic function. Heuristic function is based on some heuristic, whose primary motive is to: given the current node, determine the next immediate node which minimizes the distance between source and destination. The node which minimizes the most is given higher priority than other nodes. Other than a different function for priority, working of A* search is similar to Dijkstra Algorithm. Basic idea behind using A* search is that is exploits domain knowledge of problem.

Other algorithms like BFS and DFS also exists. These two algorithms are popular and simple than above two algorithms, but there time complexity is more as compared to them. Hence, I will prefer to use A* or Dijkstra Algorithm for finding shortest path once the map is known.

7.4 Graph storage

Graph will be stored in 2D matrix. Indexes of matrix will serve as (**X**, **Y**) co-ordinates.

For improving performance, instead of storing 2D matrix, we can store 2D lists, which in turn improves performance of Dijkstra Algorithm. However, storing graphs and applying Dijkstra Algorithm is relatively easier in matrix, so I will be using matrix, unless performance boost is required.

7.5 Finding optimal path

Once the graph is stored, then Dijkstra or A* algorithm can be used to find optimal path.