Data Science Nanodgree Program

Capstone Proposal – Robot Motion Planning

Pradeep Kumar
16 May 2020

Plot and Navigate a Virtual Maze

Domain Background

This project is inspired from an event 'Micromouse Competitions' where a small robot navigates a maze and learns the shortest path to the center of that maze. The competition originated in the 1970s and is a popular event held around the globe. The maze in the competition is typically made up of 16x16 grid cells with walls 50mm high. The robot is given multiple runs around the maze. The first run is for exploration where the robot tries to map out the entire maze, find the center, and learn the best path to the center. Then in the subsequent runs the robot is expected to reach the center in the fastest time possible, using what it has learned from the first run.

The first micro mouse competition was held in 1979 in New York where the robot mouse was intended to find its way out of a 10x10 maze and the winner was a wall follower robot. In recent years the performance of the robots has increased significantly and they are now among the highest performing autonomous robots.

Additionally, this research paper by Vishnu K. Nath and Stephen E. Levinson of University of Illinois presents how machine learning can enable a robot to solve an unknown 3D maze. In this research two approaches were discussed, one is graph traversal and other is reinforcement learning. The paper provides a clear conceptual overview of the maze solving problem and will be helpful to begin with this project. As for our project, we will be using a virtual 2-d maze and the mazes to be used in this project is provided on the Udacity Project Page in the form of text files along with some starter code files. The source of the files is provided through this link. These are the samples mazes provided by Udacity to test our robot.

My motivation for doing this project comes from my interest in the field of robotics which developed during my summer internship in which I studied basic robotics, embedded C, and built some simple arduino bots. And after working on the project 'train a smart cab' in this

nanodegree I have become more excited to learn about autonomous driving & reinforcement learning. Thus my interest lies in developing a good understanding of autonomous robotics and pursue higher studies in this field, and this project seems to be a good start for that.

Problem Statement

In simple words, the task is to plot an optimal path from the corner of a virtual maze to the center of the maze. To be precise, the goal of the project is to make an autonomous virtual robot which can gain information about the environment it is in and then learn to make decisions which can lead it to the goal in the fastest time possible. To accomplish that, the robot should be able to keep the track of its position in the maze, discover walls, dead ends, detect the goal, and map out the entire maze.

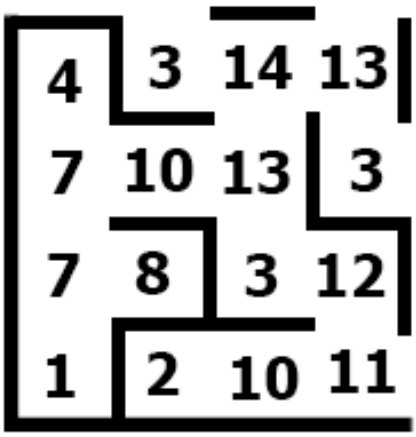
The maze is an *n* x *n* square grid where n is even. The minimum value for n is 12 and maximum is 16. There are walls in the maze which restrict all movements. The walls are along the entire parameter of the maze and along some of the connected edges of the squares in the maze. The robot will start from bottom left corner facing upwards, the square in the starting position will have walls on the left, right and back allowing a forward movement only. The goal will be at the center of the grid consisting of a 2 x 2 square.

In this project, the robot will run through 3 different mazes of dimensions 12 x 12, 14 x 14 & 16 x 16 one by one. The robot will make multiple runs in a maze. The first run is for exploration where the robot will gain information about the structure of the maze, learn all the possible routes to the goal, and find an optimal path to the goal. The robot must reach the goal in order to register a successful *run* around the maze. It should also continue to explore the maze after reaching the goal if it hasn't explored enough maze area. In the subsequent runs the robot will attempt to follow the optimal path & reach the goal in fastest time possible using what it has learned in the first run.

Datasets & Input

The mazes to be used in this project is provided on the Udacity Project Page in the form of text files along with some starter code files. The source of the files is provided through this link. These are samples mazes provided by Udacity to test our robot. In the first line of the text file there is a number at the very start which tells the no of squares on each dimension of the

maze. The following comma-delimited numbers in the file tells us the position of the walls in each square thus describing the open square edges for movement. The numbers are in range of 0 to 15. These numbers are four bit representations where the 1s represent the front/upward side, 2s represent the right side, 4s represent the back/downward side and 8s represent the left side. The bit value of 0 represents a wall and 1 represents an open edge. For example, number 9's four bit representation is 1001. Thus 9 number tells us that the front and the left side is open for movement while there is a wall at the back and right. The image below shows an example representation of a maze.



The location of the robot can be considered as the value of two point coordinates (X, Y) & it is considered to be at the center of a square. The robot has three sensors mounted at the front, right, and the left. These sensors detect the position of the walls and the open edges in the direction of the corresponding sensor in the form of a list of three numbers. For example in the image above at the starting position, the sensors will have a reading of 0 for left and right sensor and 1 for the front sensor. For any location in the maze, the robot must have the knowledge of its environment. It can chose to rotate 90 degrees clockwise or anticlockwise & move forward/backward up to three units/steps. The robot's rotation is assumed to be perfect and in case of moving into a wall the robots stays where it is.

Solution Statement

The robot will explore the maze in the first run where it will learn and map the structure of the maze, and find all possible paths to the goal. Then in the next run the robot will try to reach the goal following an optimal path in the fastest time or precisely in minimum number of steps.

The number of steps can determine whether it is the best path to reach the goal as the robot will have the knowledge of number of steps needed to reach the goal for all possible paths, and the path with the fewest steps is the fastest one. In order to test if our robot has learned the optimal policy we can perform multiple runs and for each run the path should remain same if it is indeed the best path to the goal.

Benchmark Model

For our benchmark model we will use Breadth-First Search Algorithm. This is the most common graph traversing algorithm which traverses a tree or a graph starting from the root node and exploring all the neighbor nodes first before moving to the next level neighbors.

Now, on each maze the robot will complete two runs. In the first run we will use random turn algorithm to explore and map the maze. A maximum of thousand time steps are allotted to the robot to complete a run in a maze. The robot is allowed to roam freely to navigate the maze and it must find the goal within a thousand time steps in the exploratory run. It is also allowed to explore and map the maze further after finding the goal. Now in the second run, we'll apply Breadth-First Search Algorithm to measure the robot's performance. Since the robot has gained information about the maze in the exploratory run, it should reach the goal in the second run in shortest time following an optimal path. The performance score for our benchmark model will be measured by the following metric:

Benchmark Score = (no. of steps in trial 2) + (no. of steps in trial 1) / 30

Evaluation Metrics

The robot's performance score will be calculated by adding 'no of time steps taken in the second run' & 'one-thirtieth the no of time steps taken in the first run'. Our goal is to minimize this score.

Score = (no. of steps in trial 2) + (no. of steps in trial 1) / 30

The score takes both exploration and optimization into consideration with optimization having more weight in the score. There is a trade-off between the exploration and optimization as the second run costs a complete step count for each step while the trial run costs $1/30^{th}$ of a step count for each step. For example, if in the first run the robot takes 600 steps to explore the maze and then 20 steps to reach the goal in the second run, then the score will be 20 +600/30 = 40. So our goal will be to explore the maze as much as we can in the first run in order to minimize the cost for the second run. Another thing to consider here is that while navigating

the maze in the first run the robot should avoid to make multiple visits to the cells it has visited already, so as to minimize the cost in the trial run too.

Project Design

The starter code for this project has been provided on Udacity's Project Page, so I'll begin with studying the starter code files: robot.py, maze.py, tester.py, & showmaze.py. After analyzing the code I'll start modifying the robot.py file & first define the essential variables and parameters. Then I will proceed to write the code for our benchmark model. First I'll test if robot is able to identify its location, sense the walls and make movements. Once the robot is able to do these basic functions, I'll write the code further and test if the robot can reach the goal, explore and create a map of the maze, & detect the dead ends and come out of them. Then I will test the benchmark model's performance and make any needed improvements. My aim would be to create the maze map and I'll try to accomplish this with fewer steps. Then I'll start applying other path planning and maze solving algorithms for optimization and select the one which performs the best. Now, for this project I'm considering the following algorithms:

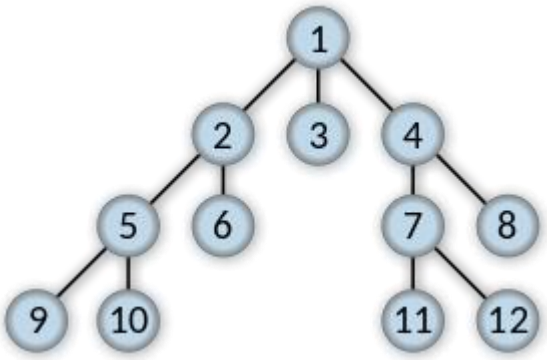
1. Random Mouse Algorithm/Random Turn:

This is a very trivial method implemented by a very unintelligent robot or agent. The robot will pick random directions to move at each step and will turn around at the dead ends. This method will always find a solution but it is extremely slow.

2. Breadth-First Search:

This is the most common graph traversing algorithm. This algorithm traverses a tree or a graph starting from the root node and exploring all the neighbor nodes first before moving to the next level neighbors. This algorithm traverse the graph breadthwise as follows:

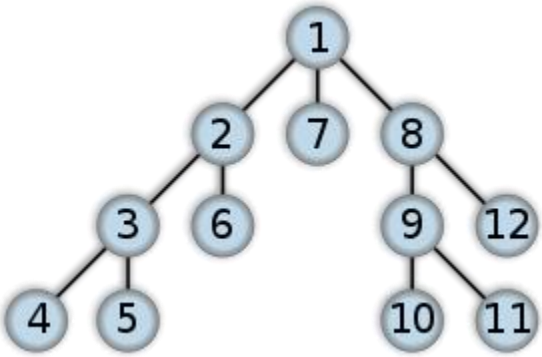
1. First move horizontally and visit all the nodes of the current layer
2. Move to the next layer



(Image source: https://en.wikipedia.org/wiki/Breadth-first_search)

3. Depth First Search:

This algorithm is a recursive algorithm which uses backtracking, i.e. moving forward until there are no more nodes along the current path and then go backwards and find a new node to traverse.



(Image source: https://en.wikipedia.org/wiki/Depth-first_search)

4. Dijkstra Algorithm:

Dijkstra algorithm determines shortest paths between nodes in a graph. This algorithm continuously calculates the shortest distances and excludes longer distances. The Dijkstra algorithm has many variants. The most common variant fixes a single node as the source node and finds shortest paths from that source node to all other nodes in the graph, producing a

shortest-path tree. It starts from the source node and marks all direct neighbors of the source node with the cost to get there. Then it proceeds from the node with the lowest cost to all of its adjacent nodes and again marks the node with the lowest cost.

5. Uniform Cost Search:

Uniform Cost-Search is just Dijkstra Algorithm with single source and single goal node. Although it can be considered as a subset of Dijkstra, it is significant enough to have its own name for which it considers the time and space complexity for a large search space. While the Dijkstra algorithm finds the shortest path from the root node to every node in the graph, the UCS algorithm finds the shortest path from the root node to the goal node. So, UCS is focused on finding the shortest path between the source and the goal rather than shortest path to every node. As soon as the path is found the UCS stops.

6. A* Algorithm:

A* is an informed search algorithm or a best-first search algorithm, meaning that it solves problems by searching among all possible paths to the goal for the one which has the minimum cost. At each iteration of its main loop, A* determines which of its partial paths to expand into one or more longer paths. It is done on an estimate of the cost denoted by f(n) and selects the path that minimizes

$$f(n) = g(n) + h(n)$$

where n = the last node on the path,
g(n) = cost of the path from the start node to nth node
h(n) = a heuristic that estimates the cost of the cheapest path from nth node to the goal

7. Flood Fill Algorithm:

Flood fill algorithm determines the area connected to a given node in a multidimensional array. The same algorithm is used in bucket fill tool of paint programs which fills the connected and similarly-colored pixels or area with a new same color. The flood-fill algorithm takes three parameters: a start node, a target color, and a replacement color. The algorithm looks for all nodes in the array that are connected to the start node by a path of the target color and changes them to the replacement color. There are many ways in which the flood-fill algorithm can be structured, but they all make use of a queue or stack data structure, explicitly or implicitly.

Resources

Udacity Project Files Link

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

Micromouse Online

http://www.micromouseonline.com/

Wikipedia

https://en.wikipedia.org/wiki/Micromouse

APEC Micromouse Contest Rules

http://micromouseusa.com/wp-content/uploads/2013/10/APEC-Rules.pdf

Algorithms - Hackerearth

https://www.hackerearth.com/practice/algorithms/graphs

Path Planning – Correl Lab

http://correll.cs.colorado.edu/?p=965

Maze Solving Algorithm – Wikipedia

https://en.wikipedia.org/wiki/Maze_solving_algorithm

Programming Theory – Solve a maze

https://stackoverflow.com/questions/3097556/programming-theory-solve-a-maze

Breadth-First Search Algorithm

https://en.wikipedia.org/wiki/Breadth-first_search

Depth-First Search Algorithm

https://en.wikipedia.org/wiki/Depth-first_search

Dijkstra Algorithm

https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm

Uniform-Cost Search Algorithm

https://en.wikipedia.org/wiki/Talk%3AUniform-cost_search

https://stackoverflow.com/questions/12806452/whats-the-difference-between-uniform-cost-search-and-dijkstras-algorithm

A* Search Algorithm

https://www.geeksforgeeks.org/a-search-algorithm/

https://en.wikipedia.org/wiki/A*_search_algorithm

Flood Fill Algorithm

https://en.wikipedia.org/wiki/Flood_fill