Machine Learning Engineer Nanodegree Capstone Project
John Kinstler 03/26/17

Robot Motion Planning

Domain Background
The goal of robot motion planning is to find a path for a robot to follow from a starting point to some distant waypoint without colliding with obstacles in the world (also called the configuration space) the robot is moving through.

Motion planning has applications in autonomous and automation robotics, in video game artificial intelligence, architectural design, medicine and biology. It is further used in the nascent self-driving car industry.

My own particular interest in this area arose from deploying reinforcement learning to teach a Smart Cab to learn the rules of the road for a previous project. Here, rather, different algorithms will be explored as a potential solution.

Problem Statement
This project takes inspiration from [Micromouse](https://en.wikipedia.org/wiki/Micromouse) competitions, wherein a robot mouse is tasked with plotting a path from a corner of the maze to its center. 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 figure out the path 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.

In this project, I will create functions to control a virtual robot to navigate a virtual maze. A simplified model of the world is provided along with specifications for the maze and robot; the goal is to obtain the fastest times possible in a series of test mazes.

The problem here is to find a powerful, low overhead way to find the path to the center of the maze in the shortest time possible without mapping the entire maze, if possible.

There are a variety of motion planning algorithms which are detailed here. Some further discussion on this topic can be found here, wherein the author describes 3 algorithms most commonly used to solve motion planning problems which are A* grid search, Visibility Graphs, and Flow Fields. Alternatively, I could explore more typical machine learning algorithms such as Neural Networks or Reinforcement Learning.

This project will likely be best accomplished using the A* algorithm for which implementation guidance can be found here.

A* is an optimization of Dijkstra's algorithm for one path to a particular goal. Dijkstra's algorithm favors the lowest cost paths to search a space. A* favors the lowest cost path that moves toward a particular goal.

Datasets and Inputs

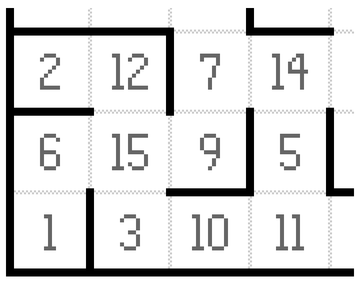
The central data/input for this problem is the maze. Additionally, the robot itself can see the configuration space via 3 sensors mounted front, left, and right. The actual inputs will be whether the robot sees walls or open spaces to the left, center, and right.

The Maze Specification is given in the MLND Capstone Project Description - Robot Motion Planning:

"Maze Specifications

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). 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."



The Robot Specification is given in the MLND Capstone Project Description - Robot Motion Planning:

"**Robot Specifications**

The robot can be considered to rest in the center of the square it is currently located in, and points in one of the cardinal directions of the maze. The robot has three obstacle sensors, mounted on the front of the robot, its right side, and its left side. Obstacle sensors detect the number of open squares in the direction of the sensor; for example, in its starting position, the robot's left and right sensors will state that there are no open squares in those directions and at least one square towards its front. On each time step of the simulation, the robot may choose to rotate clockwise or counterclockwise ninety degrees, then move forwards or backwards a distance of up to three units. It is assumed that the robot's turning and movement is perfect. If the robot tries to move into a wall, the robot stays where it is. After movement, one time step has passed, and the sensors return readings for the open squares in the robot's new location and/or orientation to start the next time unit.

More technically, at the start of a time step the robot will receive sensor readings as a list of three numbers indicating the number of open squares in front of the left, center, and right sensors (in that order) to its "next_move" function. The "next_move" function must then return two values indicating the robot's rotation and movement on that timestep. Rotation is expected to be an integer taking one of three values: -90, 90, or 0, indicating a counterclockwise, clockwise, or no rotation, respectively. Movement follows rotation, and is expected to be an integer in the range [-3, 3] inclusive. The robot will attempt to move that many squares forward (positive) or backwards (negative), stopping movement if it encounters a wall."

Solution Statement

As indicated in the problem statement there are several algorithms that are used to solve motion planning problems. However, this problem was specifically laid out to be a lattice grid search problem as can be determined from the Maze and Robot Specifications. This automatically removes Visibility Graphs as they are designed specifically to use graphs instead of grids.

Additionally, Flow Fields will likely be problematic since they are often not optimal nor complete and can get stuck in local minima.

According to the literature A* seems to be the preferred choice for solving motion planning problems. A* is optimal and complete, which means it can find the "best" path through it's grid space as well as find a solution if one exists. It's a fairly simple algorithm to understand and deploy and can be solved quickly.

A* works by balancing the cost of arriving at a particular location in the configuration (node) from the start and the cost of getting from that node to the goal. The cost of getting from any node to the goal is determined by the heuristic that is used.

In the case of a configuration space that takes the form of a lattice grid (which will be used for this project), the Manhattan distance would be a common and easy heuristic to apply. Additionally, other heuristics like Diagonal Distance and Euclidean Distance won't work since our robot can follow the grid lines since it can't make 45 degree turns.

The biggest problem for this project may come in the form of determined which route to select if multiple routes exist – how to break ties. This will be explored further if the problem is encountered.

Other approaches to this problem might be Neural Networks (function approximation) and Reinforcement Learning. The problem with NN is that it requires large training datasets to train the learner. However, I'd likely use A* to generate those datasets anyway, so why bother trying to approximate that which I already know.

Reinforcement Learning would be more useful for exploring more complex systems where multiple goals need to be achieved simultaneously. In the case of the SmartCab project I need to arrive at a goal within quickly (A*), while also not violating traffic laws nor getting into any accidents. This maze running project is far simpler than that so RL would likely be overkill.

Benchmark Model

For the benchmark model could use a Breadth First Search which explores the maze equally in all directions. However, since the maze is typically constrained to one only one or two choices of direction to take, this might approach might give some weird results.

Another approach could be to simply create a policy whereby the robot just turns left every time it encounters and obstacle. This gets problematic when it runs into a corner with the wall in front and to the left. A situation could arise where the robot is simply retracing its steps back and forth between that corner and the start.

So the policy could be that it first turns left. If there's an still an obstacle, it then turns right twice. If there's still an obstacle, it's at a dead end and would need to turn right again.

I'll start with Breadth First Search and modify it if it doesn't seem to be working.

Evaluation Metric

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.

In the review, it was asked that I state this more clearly. What exactly isn't clear here?

Project Design

The goal of this project is to create a robot that runs from the start to the center as quickly as possible.

To do this I must complete the robot.py script which contains the code for the algorithm necessary to find the center as well as the code for deciding which move to make next.

I will start by implementing the benchmark Bread First Search or policy implementation first to get some basic results and collect scores for thus runs.

I will then implement the A* algorithm and make several runs to start collecting scores for those runs.

I will use these scores to calculate a running percent improvement over the benchmark. Here improved scores would mean faster times so the graph could be simply a line that increases with percent improvements over the baseline.