Machine Learning Engineer Nanodegree

# Capstone Project-Robot Motion Planning: Plot and Navigate a Virtual Maze

Lincoln Stein

December 18, 2016

## I. Definition

### Project Overview

Autonomous navigation, route planning, and optimization are popular topics even to those not directly involved in machine learning. These concepts are the core to everyday occurrences such as using a GPS navigation app to make your way to work or a new destination as well as serving the basis for self-driving cars. There are several problems to address in order to implement a successful navigation model. This project approaches this domain in a simplified manner to explore the basic concepts that underpin the more advanced versions we see today.

The project is based on the micromouse competition. In the micromouse competition, teams compete to create a robot that can navigate a maze in the shortest amount of time. In this project, the goal is to create a program that will control a virtual robot navigating a virtual maze. The model of the robot and maze have been somewhat simplified. The virtual robot is given two runs to navigate the maze and obtain the fastest time possible. The first run is allocated 1000 moves to reach the center goal square as well as map out alternative paths through the maze. The robot is then given the second run to score the fastest time possible navigating from the start point to the goal in the center of the maze.

### Problem Statement

The problem of interest is successfully navigating to the goal of the maze, which is located in the center of the maze, in the least amount of time possible. The maze is considered discretely, consisting of a square grid of cells. There are several sub problems that must be solved in order to reach this primary goal. These sub problems are: localization within the maze, sensing the environment to avoid obstacles such as walls, mapping the maze to determine feasible paths to the goal, and moving the robot through the maze.

I plan to solve these problems by implementing a method for the robot to track its location and orientation in the maze given the initial condition being the coordinate [0,0] and oriented “up”. This is accomplished by tracking the moves that the robot makes within the maze. The ability to sense distances forward, left, and right has been provided to the robot already. The distance to the next wall is updated at the beginning of each time step. This will then be used to update the map that the robot is creating as it travels through the maze. The robot is given the dimensions of the maze at the start of the first run. It will use this information and the fact that the goal is located in the center to initially score every location in the maze to help it decide which way it should travel by comparing the scores of the cells near it.  Initially, the robot will travel towards the goal, using the score assigned to each location and the map it is building to navigate. Once the goal is reached, the robot will attempt to reach all the locations it has not already been to in order to get the most complete map.  Once it has done this, it will request to be reset and calculate the optimal path from the start location to the goal. At the start of the second run, it will follow this optimal path from the starting point to the goal.

The expected solution will move through  a given maze from the starting position in the bottom left corner to the goal square which is a 2x2 square in the center of the maze.  The robot is given 2 runs through the maze. The first run intended to provide the opportunity for mapping the maze but the robot must enter the goal square at some point. With the information gained from the first run, the robot should determine the optimal path to traverse from the start position to the goal. Then, on the second run, the robot should execute the fastest run possible from the start position to the goal.  The robot is allotted a maximum of 1000 steps combined for both runs.

## Metrics

The metric used to measure the robots performance is the score reported by the provided testing program. This score is based on the time steps the robot takes during the second run plus 1/30th of the time steps taken during the first run. This metric ensures the solution meets the goals of the project, given that the problem is to create a robot that can reach the goal in the fastest time possible.

## II. Analysis

### Data Exploration

The virtual robot is provided with the necessary ability to sense and move in the environment. The model of the robot is simplified: it is in the center of the current space, it is able to rotate ninety degrees left or right and move forward or backward up to three spaces without any errors, the sensors on its front, left, and right report perfect information on the number of open spaces in their respective direction. This sensor data is provided at the start of each time step in a list with the reading from the robots left, front, and right side respectively. For instance, the first reading the robot makes in test maze 1 is given as [0, 11, 0] indicating no open spaces to the left or right and 11 open spaces in front of the robot. If the robot moves forward one space, the reading is reported as [0, 10, 0]. This is the only way the robot is able to interact with the maze, illustrating the crucial need for the robot to remember and learn. The constraints of the maze inform the robot about the location of the goal in relation to its starting location. Using the sensor data and its knowledge of the goal coordinate, the robot can move through the maze towards the goal. Looking at the first test maze provided, there are several areas where the robot will have to move further from the goal in order to reach it. There are also a number of dead ends that could result in unnecessary moves for the robot. Implementing an agent that chooses randomly between left, straight, or right turn and moving between 1-3 moves forward, it reaches the 1000 step time limit without reaching the goal, illustrating the difficulty of navigating the maze.

Exploratory Visualization

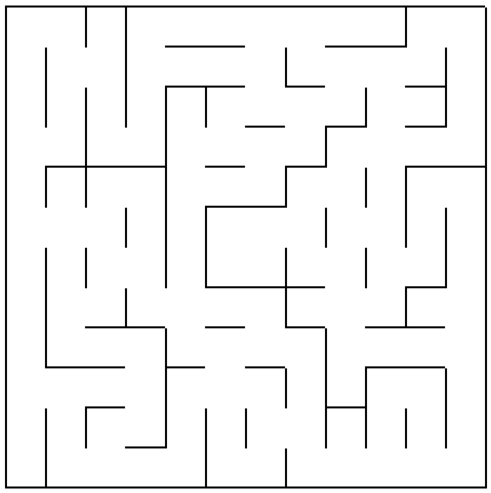

Fig.1: Test Maze 1

Figure 1 provides an overhead view of Test Maze 1. This view provides a helpful reference when examining what the robot “sees.”

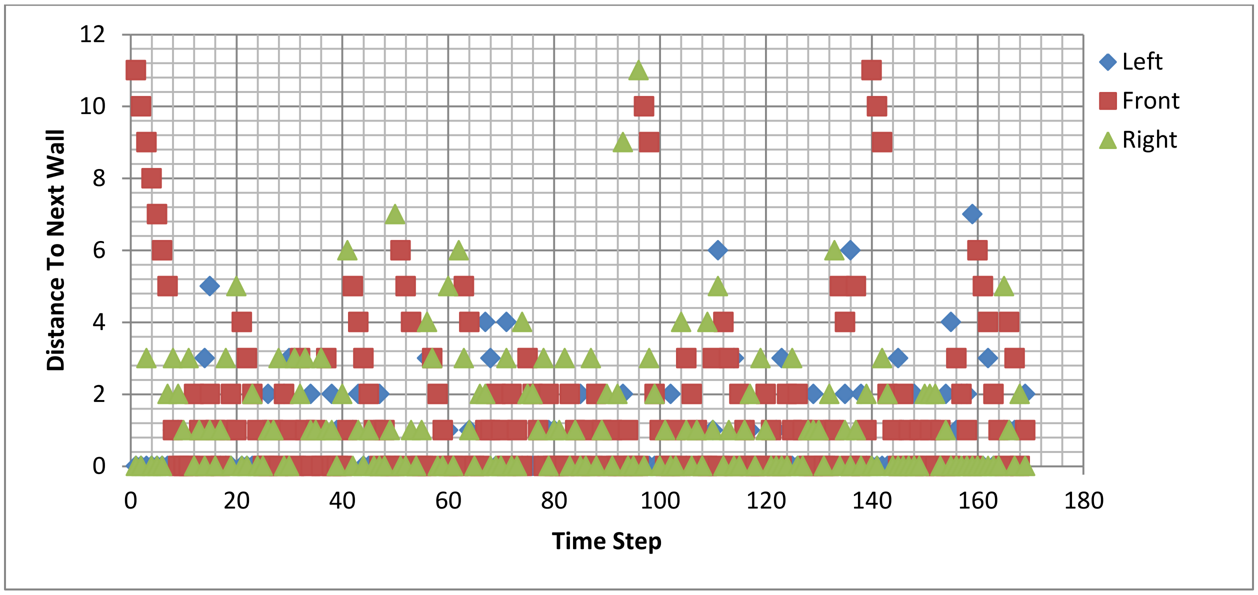

Fig. 2: Plot of Sensor Data

Figure 2 provides a sample plot of the readings the robot takes from each of its sensors as it navigates the maze. This is how the robot “sees” the maze. Comparing the two figures, it is easy to see the challenge in creating a robot that can effectively interpret the data in Figure 2 in order to successfully navigate what we see in Figure 1. While it may be possible to match sensor readings to possible locations if the robot is given the full map initially, the more general approach requires the robot to have an awareness of where it is in order to correlate the data with a location and build its own map of the world.

## Algorithms and Techniques

In order to solve the problem of plotting an efficient path from the start to the goal, I plan to implement a basic flood algorithm. Other algorithms, such as A* and Dijkstra are also commonly used in path planning. The Dijkstra algorithm starts at the current location and expands from this node, adding new nodes, in this case maze cells, to a queue to be analyzed. It continues expanding until it has found a path to the goal. The A* algorithm is similar, but it uses an additional heuristic when assigning the value for a node. This heuristic is an approximate value for the distance from that node to the goal node. This heuristic is used to limit which nodes are searched as part of the possible path to the goal, as each node is compared using the heuristic and those that are longer than already present paths are eliminated. The flood fill algorithm was chosen based primarily on its simplicity to implement and troubleshoot. This algorithm could represent a future area of improvement for the robot. Based on the default parameters provided, which include the dimensions of the maze and the starting location, which is the bottom left corner of the maze and considered the origin. The initial flood assigns a value to each cell based on the number of cells that have to be traversed to reach the goal from that cell. An example visualization of the flood fill algorithm is shown in Figure 3, demonstrating the goal as the center square marked with an X and the surrounding squares assigned a number based on the number of steps they are from this goal square. As the robot traverses the maze, it gathers sensor data which is used to update its map of the maze. Once this map is updated, the flood algorithm is run again, updating the values for each location based on new information on obstacles. The flood algorithm provides a simple to implement but effective method for determining the distance from any point in the maze to the goal location. The dataset, which describes the parameters of the maze, is interpreted indirectly by the robot. It is presented the information only as inputs to its sensors, so the robot is required to recreate the dataset by tracking its moves and sensor information in order to understand the data and complete the goal of reaching the maze.

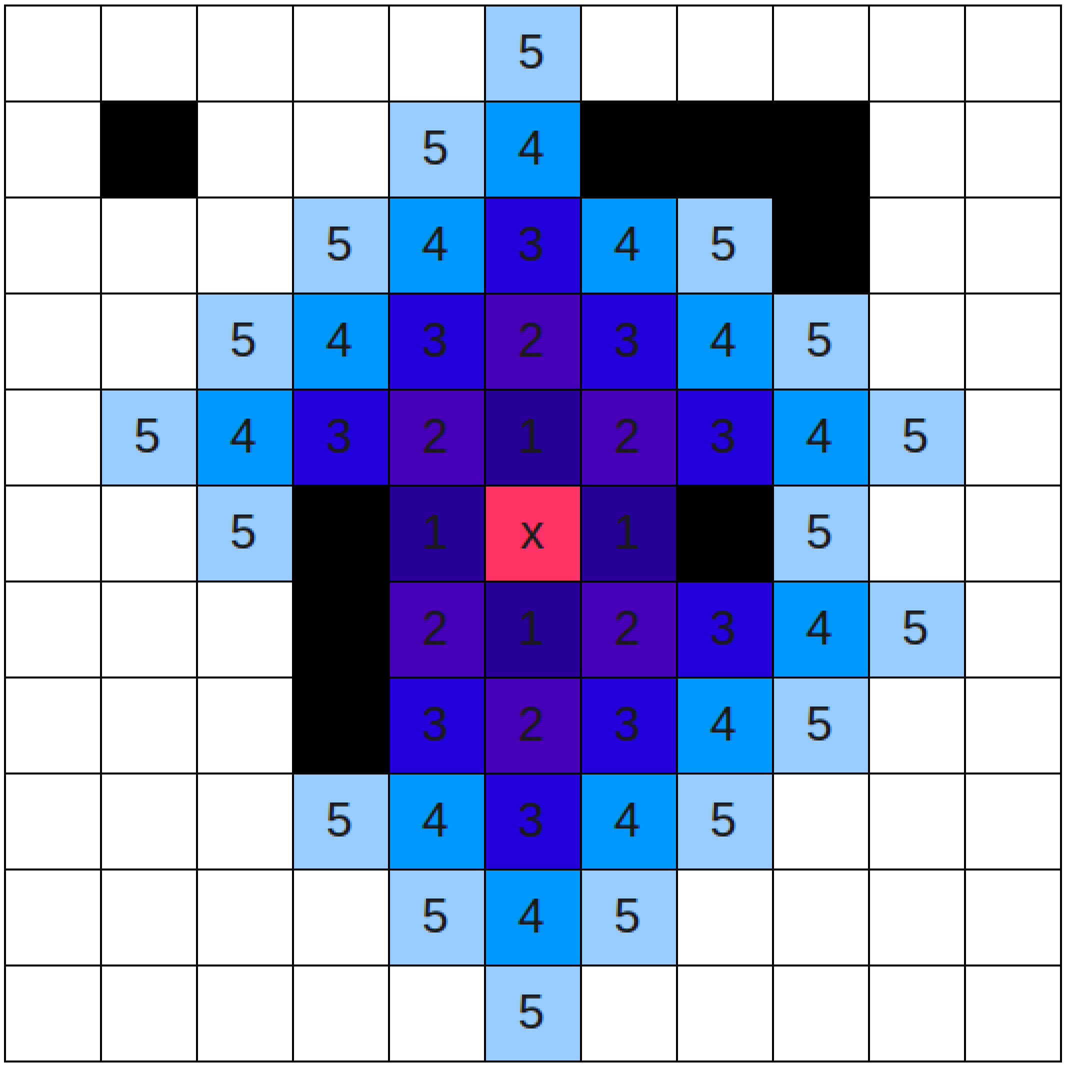

Fig. 3: Flood Fill Visualization from http://stackoverflow.com/questions/8120179/how-many-moves-to-reach-a-destination-efficient-flood-filling

## Benchmark

As defined in earlier discussion, the robot is limited to 1000 time steps combined for both exploratory and testing runs and the reported score is the number of time steps taken on the second run plus 1/30th the time steps taken on the first run. The maximum size of the maze is a 16 x 16 cell grid, which results in 256 unique locations. This is a guideline for how many steps will be added for the exploratory run for the scoring if we want our robot to explore the maze thoroughly. However, looking at the example maze provided, there are several areas where the robot will likely have to backtrack if it visits every cell, which would significantly increase the amount of time taken. In the sample maze discussed above, the robot took approximately 34 moves to find the goal. Although the number of timesteps will vary based on maze design and complexity, this is what I used in calculating the initial baseline. So, if the scoring takes 50 time steps adding in 1/30th of 300 time steps for exploration, we come to a benchmark value of 60.

## III. Methodology

### Data Preprocessing

The data required no specific preprocessing steps as the specifications for the sensor readings as well as the design of the maze was already provided. The data is able to be used as it is provided, there is no scaling or refactoring needed in order for the robot to utilize it effectively.

### Implementation

As mentioned before, the robot interacts indirectly with the dataset which defines the maze through the information it receives through its sensors. At the beginning of each time step, the robot reads the information from its sensors. This information, which is simplified to whether there is a wall in the left, forward, or right direction with respect to the robots heading is then added to the map for the cell the robot is currently in. The robot then examines each cell in relation to its neighbors within the current map and updates the position of any walls by propagating them to the appropriate adjacent squares. This is necessary as the robot can only consider each cell individually and if walls are not propagated between cells, it would appear to the flooding algorithm that, for instance, cell [0, 0] could be accessed from cell [0,1] when there is a wall between them but the robot has only seen it from cell [0,0]. After updating the map, the robot executes the flooding algorithm. Starting at the goal square which gets a value of zero, the robot checks which cells are reachable based on the current map. Then, it iterates through each square, increasing the number assigned to each cell based on the number of steps it is away from the center square, ensuring it does not visit a square more than once.  The robot then decides its next move based on which available moves will take it closer to the goal cell using the distance map updated by the flooding algorithm. This flooding algorithm is used to guide the robot to a series of waypoints around the maze during the first run before going to the goal location. These waypoints are designed to get the robot to explore more of the maze and have a more accurate map.  After finally reaching the goal square, the robot requests to be reset and executes the best path it has discovered.

In coding for this project, the implementation of the mapping, flooding, and updating algorithms are quite simple. They are primarily built on relatively simple methods to iterate over a 2 dimensional list and comparing adjacent values. Below is an example of how the robot updates its map of the maze based on the information from sensors.

```
#Update mapping for all cells based on current data
        for i in range(len(self.mapping[0])):
            for j in range(len(self.mapping[1])):
                #print "Updating map based on cell: ", [i, j]
                #print "Cell value: ", self.mapping[i][j]
                #Determine adjacent cells
                adjacent = [[i, j-1], [i-1,j], [i,j+1], [i+1,j]]
                #print "Adjacent cells are: ", adjacent
                for cell in adjacent:
                    #print "Updating cell: ", cell
                    if (0 <= cell[0] < self.maze_dim) and (0 <= cell[1] <
self.maze_dim):#verify cell within maze
                        #print "Cell within maze"
                        #if cell has wall in direction adjacent, update
adjacent cell to show wall as well
                        if self.mapping[i][j][adjacent.index(cell)] == 0:#if
current cell has wall between adjacent cell
                            #print "Wall adjacent to: ", cell
                            #print "Current cell value: ",
self.mapping[cell[0]][cell[1]]

self.mapping[cell[0]][cell[1]][(adjacent.index(cell)+2)%4] = 0
                            #print "Updated cell value: ",
self.mapping[cell[0]][cell[1]]
```

Fig. 4: Code Excerpt Showing Mapping Update Routine

Refinement

The final program for the robot was developed in a series of refinements. Initially, a basic distance map that gave a value to each cell based on its distance from the center goal was used for the robot to decide its next move. This initial idea was refined to include the flooding algorithm which accounted for walls in determining distance to enable to robot to navigate around intervening obstacles. This allowed the robot to successfully discover a path to the center of the maze, which for test maze 1 was 46 time steps. However, the robot had no information on a significant portion of the maze which means it wasn't able to compare different paths. In order to improve the robots knowledge of the maze, I decided to implement a method to force the robot to travel through and gain information on a larger portion of the maze by traveling to several different waypoints prior to locating the goal. When the robot is first initialized, it calculates a series of waypoints spread throughout the maze based on the size of the maze. These are then set as the intermediate goals and once the robot reaches a waypoint it moves on to the next one. In this way, the robot travels through more of the maze and gains more accurate information and a more complete map of the maze. This more informed map improves the robots chances of finding the optimal path from the start to the goal location. The waypoints initially were the four corners of the maze. Using this approach, the robot achieved scores of 39, 67, and 84 time steps on test mazes 1-3, respectively. Using the midpoints of the outer walls and the start position as intermediate waypoints gave a score of 34, 55, and 57 for test mazes 1-3, respectively. This proved to be the most robust method on the test mazes and I feel provides a balance between effectiveness and implementing a more complicated algorithm to travel the maze more fully, such as ensuring the robot is visiting every cell.

## IV. Results

### Model Evaluation and Validation

Examining the final implementation of the robot, it performs well compared to the initially established benchmark of 60 time steps, coming in below that on all three test mazes. The robot is within all required parameters of the problem statement. The robot has been tested over several mazes and exhibits an appropriate ability to generalize and adapt to new datasets. Given the robot falls within necessary parameters and exhibits the ability to deal with new data, I feel the results are valid and accurately represent the implementation.

## Justification

The results for the three test mazes respectively are 34.6, 55.9, and 57.4. These results all fall under the established benchmark of 60 timesteps discussed earlier. This leads to concluding that the robot is a significant solution to the problem of navigating a maze with the given parameters.

## V. Conclusion

### Free-Form Visualization

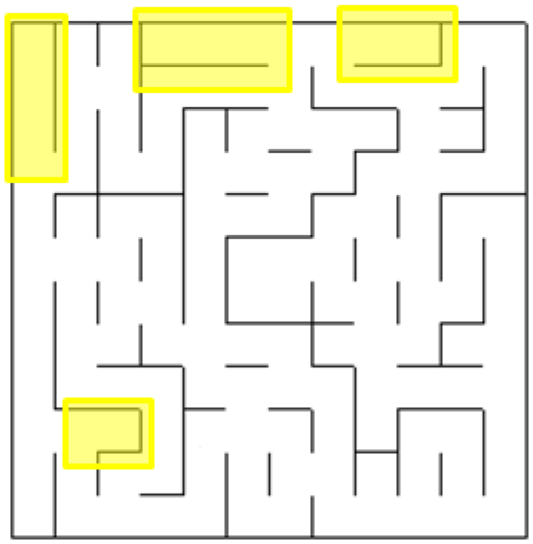

Fig. 2: New Maze

Alterations to the first test maze were made to increase the number of long lead dead ends, which are highlighted in yellow. This is an area where my implementation has not been optimized to handle very efficiently. One method of increasing efficiency would be to add a routine to determine what cells have been fully characterized without the robot having to actually visit that cell. This illustrates the flaw that my robot may not find the optimal path as it does not explore the entire maze, but rather takes an approximate approach.

## Reflection

The challenges inherent to creating a robot that is able to localize and navigate a maze is a interesting problem to solve given its use as a stepping stone to solve more complex problems involved in implementing robotics in the real world. I began the project by just getting the robot to move in the maze and read its sensors. The next step involved basic path planning based on the distance to the goal and creating a way for the robot to map the maze as it travelled. Once this was implemented, I moved on to implementing a flooding algorithm to discover an appropriate path and propagating the information from sensors across the map.  Finally, a improved method for exploration and testing run were implemented to complete  the robot. The most interesting aspects of the project for me were the implementation of the mapping and path planning logic. The process of translating how a person would solve a maze into a method for a robot to follow was a enjoyable challenge. This opened my eyes to the key machine learning tenet of basically translating how we perceive things into numbers and data constructs along with appropriate decision logic to create an “intelligent” behavior.

I think the most difficult aspect was getting the first few steps done and determining a method for mapping and planning. Just getting enough working to be able to test ideas was the key difficulty I had to overcome. Looking beyond the constraints of the micromouse inspired project, the basic algorithms and methodologies could be applied to more general navigation problems. While it performs well within the context of the problem, I know it would need significant work to be more generally applicable.

## Improvement

One area that could be improved within the current project scope would be the robots use of the exploration run. While the current implementation is effective, it is not as generally applicable as the other algorithms that have been implemented. Other improvements could be made in the choice of algorithm that may be more efficient in terms of computation time needed. This would become more significant as the complexity of the maze grew or the robot was placed in a continuous environment.   If the robot was put into a continuous domain, the measurements that the sensors make would need to be further processed to gain the same data. The robots movements would also need to be tracked more precisely, but the general process would remain very similar. Given accurate and precise sensor information, most continuous domain maze problems could be discretized to an suitable degree of precision that discrete techniques could be used.