Machine Learning Engineer Nanodegree

Capstone Project

Daiki Matsunaga

February 13, 2017

I. Definition

Project Overview

In this capstone project, I will tackle the difficult problem of robot mapping and motion planning. To put simply, this is the problem of programming robots that can map out its environment, plan an optimum path and take action in accordance with that plan. Robots have been a topic of interest for decades, notably in science fiction films. While robots were often depicted as autonomous creatures, which can become threats to human existence, the co-existence of sophisticated robots and humans has been merely a topic for thought experiments. However, with increased computing capabilities, greater amounts of data and improved algorithms, the field of robotics has suddenly emerged as a real tool in helping humans work and live more efficiently. For example, autonomous vehicles, drones, disaster relief robots and cleaning robots are some of the many applications of robotics, which are already starting to make an impact on today's world. In this project, a simpler version of the problem will be used to explore the world of robot mapping and motion planning. In particular, the robot will explore a maze, map out the surrounding environment and find the fastest path to the destination.

This seems simple at first glance but I believe the core essence of the problem is similar to more complicated problems that are being researched by large

corporations and universities. My goal here is to get a glimpse of the exciting world of robotics and hopefully use this as a stepping-stone to tackle large-scale problems in the future.

Problem Statement

This problem, in essence, is how to program a robot that can map out the environment and make rational decisions in accordance with the inputs to achieve a certain goal. In this case, the robot's goal is to understand the structure of the maze and make the correct decisions (go straight, go backwards) in order to reach its destination. The solution, then, would be to have the robot reach its destination in the shortest amount of time for different types of environments. Also, the time it takes for mapping is a part of the solution that needs to be considered. Therefore, it can be said that a solution to this problem is to build a robot that maps its environment quickly and reaches its destination quickly as well. This performance can be measured simply by how many steps it takes to map out its environment as well as the number of steps taken to reach a destination from the starting point.

Metrics

The benchmark metric for this project is initially provided as "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." This suggests that the time it takes to map out the environment in the first run is not as important as the time it takes from start to finish in the second run. This makes sense from a robotics or self-driving car perspective if we imagine a self-driving car, for example, that will be sold to customers in the future. The time it takes for this self-driving car to learn its surroundings, is not as important as how safe and reliable it is when the customer actually buys the vehicle.

II. Analysis

Data Exploration

The datasets used in this project are three text files used to create a maze. For example, Figure 1a shows an example of a text file, which has the first row show the number of rows in each dimension, and the subsequent numbers showing whether or not each side has a wall (on the top, right, left and bottom). The numbers represent four-bit numbers, which have 0 if there is a wall and 1 if it is open. The four numbers are 1, 2, 4, 8, for top, right, left and bottom, respectively. For example, a 10 would indicate that the right and bottom edges are open (1*0 + 2*1 + 4*0 + 8*1). The file ‘maze.py’ uses this text file to create a map, as shown in Figure 1b. The visual demonstration of the text file (produced by ‘showmaze.py’) can be shown in Figure 1b, as well as 2b and 3b.

Figure 1a

```
12
1,5,7,5,5,5,7,5,7,5,5,6
3,5,14,3,7,5,15,4,9,5,7,12
11,6,10,10,9,7,13,6,3,5,13,4
10,9,13,12,3,13,5,12,9,5,7,6
9,5,6,3,15,5,5,7,7,4,10,10
3,5,15,14,10,3,6,10,11,6,10,10
9,7,12,11,12,9,14,9,14,11,13,14
3,13,5,12,2,3,13,6,9,14,3,14
11,4,1,7,15,13,7,13,6,9,14,10
11,5,6,10,9,7,13,5,15,7,14,8
11,5,12,10,2,9,5,6,10,8,9,6
9,5,5,13,13,5,5,12,9,5,5,12
```

Figure 1b

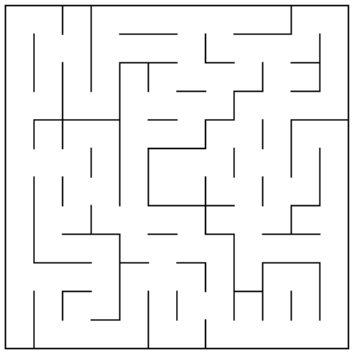


Figure 2a

```
14
1,5,5,7,7,5,5,6,3,6,3,5,5,6
3,5,6,10,9,5,5,15,14,11,14,3,7,14
11,6,11,14,1,7,6,10,10,10,11,12,8,10
10,9,12,10,3,12,11,14,11,14,10,3,5,14
11,5,6,8,11,7,12,8,10,9,12,9,7,12
11,7,13,7,14,11,5,5,13,5,4,3,13,6
8,9,5,14,9,12,3,7,6,3,6,11,6,10
3,5,5,14,3,6,9,12,11,12,10,10,10,10
10,3,5,13,14,10,3,5,13,7,14,8,9,14
9,14,3,6,11,14,9,5,6,10,10,3,6,10
3,13,14,11,14,11,4,3,13,15,13,14,10,10
10,3,15,12,9,12,3,13,5,14,3,12,11,14
11,12,11,7,5,6,10,1,5,15,13,7,12,10
9,5,12,9,5,13,13,5,5,12,1,13,5,12
```

Figure 2b

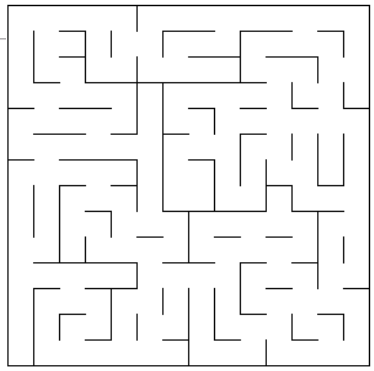
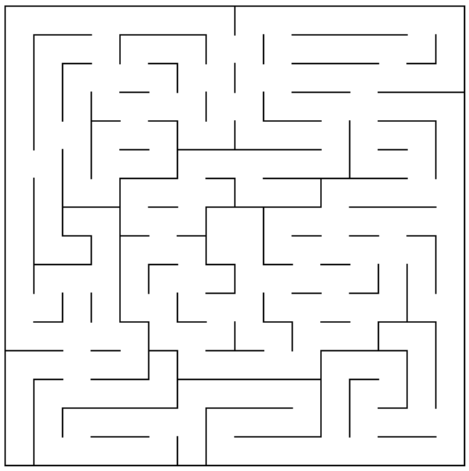


Figure 3a

```
16
1,5,5,6,3,7,5,5,5,5,7,5,5,5,5,6
3,5,6,10,10,9,6,3,5,5,13,7,5,5,6,10
11,6,11,15,15,5,14,8,2,3,5,13,5,6,10,10
10,10,10,10,11,5,13,5,12,9,7,6,3,15,13,14
10,10,10,9,12,3,5,6,3,6,10,11,14,11,6,10
9,14,9,4,3,13,6,11,14,10,9,12,11,12,10,10
1,13,6,3,14,3,15,12,9,15,6,3,13,7,12,10
3,6,10,10,9,14,8,3,6,8,10,9,7,13,7,12
10,10,10,10,3,13,7,13,12,3,14,3,13,7,13,6
10,10,10,11,12,3,14,3,6,10,10,10,3,15,7,14
10,9,12,9,7,14,11,14,10,8,10,10,10,10,10,10
11,5,5,6,10,11,14,11,15,6,9,13,14,10,10,10
11,7,6,10,9,14,9,14,10,10,3,7,15,14,10,10
10,10,9,12,2,9,5,15,14,10,10,10,10,11,14,10
10,11,5,5,12,3,5,12,10,11,13,12,10,10,9,14
9,13,5,5,5,13,5,5,13,13,5,5,12,9,5,12
```

Figure 3b



The main difference in these mazes is the dimension. Maze 1, 2, and 3, have dimensions 12, 14 and 16, respectively. Naturally, having larger dimensions means greater difficulty since the mazes are larger and thus takes longer to explore in finding the optimum path. Aside from that, however, the mazes seem to be similar in complexity and the difficulty of solving the mazes do not drastically differ.

In terms of the characteristics of the maze, it is noticeable that the first row from the starting location is very open, while the middle of the maze is more convoluted. This means that the robot will require many turns and rotations in the middle of the maze. Also, the path from the starting location to the goal is not a straight path but rather a path that is shaped like a U. This is because the area around the goal location is blocked off, almost like drawing a vertical line through the middle of the maze. Therefore, the robot

will have to go around the goal area and into the goal entry point. In this sense, it becomes increasingly important to make the correct right turn in the first row because it is hard to recover from a bad decision.

Algorithms and Techniques

To figure out what the best algorithms are for this project, it was important to understand the rules of the robot and the nature of the environment. The robot, in this environment, is allowed two runs, as long as the total number of steps is less than 1000. The first run has the robot roaming around the maze trying to map its environment out. Once the robot reaches its destination in this first run, it can go back to the starting point whenever the robot is ready. The second run, then, is the robot's real test where the robot starts in the left hand corner and goes for the destination, which is the 2x2 square in the middle of the maze. At each step, the robot senses three pieces of information, namely whether or not the wall on the left, center, and right of the robot is open. Using this information, the robot returns a turning angle (-90, 0, or 90) and the number of steps to take forward (0 for stationary, (1, 2, or 3) for forward and (-1, -2, -3) for backwards. It is worth noting here that the robot is not allowed more than three steps per move.

With the above in mind, I plan to evaluate A* as the algorithm used for the second run, and the first run will be a mapping step that fits with the information necessary for A* search in the second run. First of all, I chose A* search as opposed to Breadth First Search or Dijkstra's algorithm because it is a algorithm optimized for finding a single location or a goal. Breadth First Search, for example, explores in all directions and Dijkstra's algorithm only considers the cost of each movement. A* search, on the other hand takes the best of both worlds by considering the cost at each step as well as the distance from each location to the goal. For example, let's say I am at location A and I have a choice to go to location B or C. The distance from B to the goal is 3 and 4 from C to the goal, while the cost to going to B is 10 and the cost of going to C is 5. In this case, A* search prefers C because

the sum of the movement cost and location to goal is less for C (9 compared to 13). It seems that this is the preferred algorithm since the maze only has one goal location.

In order to implement this, it is important for the robot to make an accurate map in the first step so that at the end of the first step, the A* algorithm can find a path to the goal that is the quickest. This mapping process will be developed manually by using the wall information in each step to update wall information for other locations as much as possible. For example, if the sensor tells us that there are three open steps to the left at (3,3), we automatically know that there are two open steps to the left of (2,3), one open step to the left of (1,3) and a wall on the left of (0,3). Using this idea, the robot can roam around randomly (giving priority to unexplored locations) and keep updating the wall information. It is also worth noting that the wall updating method for mapping is possible because the robot knows the dimensions of the maze when it is initialized. This means that we simply have to make an array using the dimensions and fill in the wall information as the robot roams around.

Benchmark

The benchmark model can simply be one in which a human (me) solves each maze and finds the optimum solution to get to the destination. We can then compare this result to the robot's performance and set a threshold e.g. within +10% of time steps compared to the shortest time solved by a human. For example, it seems that the shortest amount of steps taken to solve the maze in Figure 1b is 18 steps (assuming it can move up to three coordinates at each step). If the robot solves it within 20 steps, it would have a very high performance. Also, as previously mentioned, the metrics used to evaluate the performance includes the time it takes in the first run. Of course, the emphasis will be on the number of steps it takes to complete the maze in the second run.

III. Methodology

Data Preprocessing

No preprocessing of data was necessary because the maze was initially provided. The three mazes provided are already very challenging with dimensions 12, 14, and 16 as its width and height. Any preprocessing to make the maze size larger or more complex would require more steps than the limit of 1000 steps set in this project.

Implementation

In my first attempt for this project, I was trying out different ways to map out the wall information for each location. However, moving randomly did not allow the robot to reach the goal in time, so I added an if-statement that forced the robot to go into the goal area if it is not explored and is able to go into the goal location in one step. Also, since the robot only had three sensors, I thought it was necessary to turn 90 degrees at each step and record the remaining wall information. However, this turned out to be too costly in terms of number of steps, and unnecessary since wall information for the neighbors are updated as well. To implement the ‘update wall’ step, I wrote a somewhat complicated function called ‘update_wall_info’. This function updates all of the neighbor wall information that can be deduced. Also, I attempted to call this function again in the second round to update any missing wall information. However, I was not fully understanding the way A* works. My initial idea was that A* can be used in the second round to find a path. However, A* is inherently an algorithm that explores different locations and finds an optimum path only at the end. This means that A* has to return the optimum path before the second round begins. With this realization, I created a new python file called a_star.py that does all of the A* exploration.

Refinement

My initial results were very mixed since it usually ran out of trials or got stuck in one location for a long time. Even if it reached the goal, the results were very low with a score of about 240. This was mainly because the first round did not gain enough wall information and thus couldn't feed an accurate map of the maze to the A* algorithm before the second round. Therefore, I included the if-statement to turn 90 degrees without moving again (if there is an unknown edge) and initialized 0 values for the perimeter of the maze (0 for the left edge of 0, 3 for example). To elaborate, 'self.walls' is the numpy array that is size 12 x 12 x 4 (in the case of a 12 x 12 maze) and stores how many steps are open in the left, top, right, and bottom locations. If, for example, the self.walls for a particular location showed [0, 8, -1, 3], there is an unexplored edge (-1) on the right side. Therefore, we turn 90 degrees, record that unknown wall information and move on. Also, when the maze is initialized as 12 x 12 for example, we know the wall information for the perimeter of the maze, so I included that as well. With this change, the results were decent for Maze 1, with a score of 49.133, 50.833, 48.2, 46.967, 49.1, and 46.4. Each of these trials had a second round step count of 19, which means that after the second round started, it took 19 steps for the robot to reach the goal from the starting point. This was a pretty decent score and the step count of 19 was definitely similar, if not faster, than the benchmark score. However, this model unfortunately did not work for Maze 2 and Maze 3, which are size 14 and 16, respectively. This was mainly due to the fact that the mapping process in round 1 did not finish in time, leaving none to very little time for round 2. Even if the robot was able to get to round 2, the results were very poor or incomplete since the maze that was fed into A* was poor. Also, I had a problem in which the A* algorithm kept producing an impossible path such as 0,13 from the start location on Maze 3. This is impossible since there is a wall at 0, 3.

Two changes I made solved the above problems and completely altered the outcome.

1. Previously, the robot randomly chose the next location whether or not the next locations were unexplored. However, I changed this system to two different possible actions. First, if there are possible next locations that are unexplored, choose randomly from the choices. Second, if all possible next locations are already explored, choose the location that is least explored.
2. As for the A* algorithm not giving optimum path choices, or providing impossible first steps, it was a simple bug that calculated cost based only on the g-values. This was mostly solved after taking into account the h-values as well. Also, there seemed to be confusion in the algorithm when a list was fed into the queue at each step. After making it all consistent with tuples instead of lists, the problem was completely resolved and the A* algorithm produced an achievable path for all mazes.

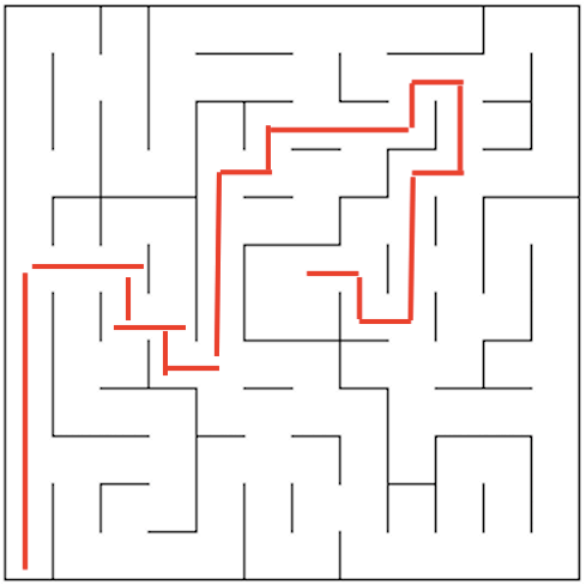
IV. Results

Model Evaluation and Validation

The final results for the final model are as follows:

1. *Maze 1 (12 x 12)*

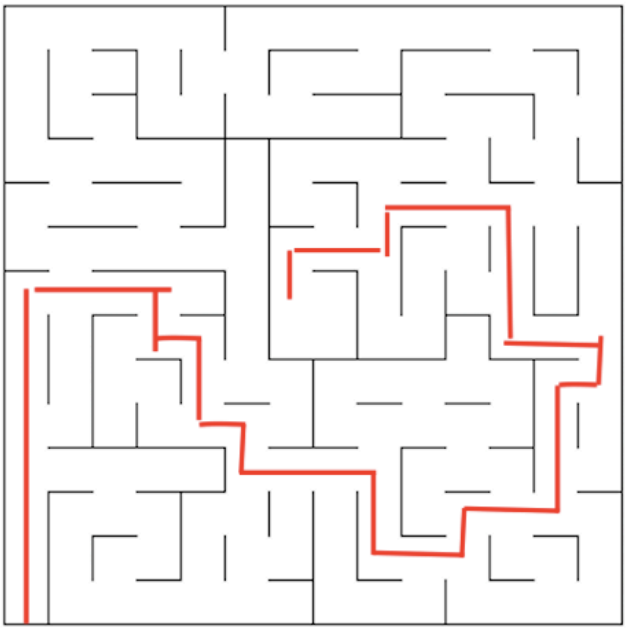
```
12
1,5,7,5,5,5,7,5,7,5,5,6
3,5,14,3,7,5,15,4,9,5,7,12
11,6,10,10,9,7,13,6,3,5,13,4
10,9,13,12,3,13,5,12,9,5,7,6
9,5,6,3,15,5,5,7,7,4,10,10
3,5,15,14,10,3,6,10,11,6,10,10
9,7,12,11,12,9,14,9,14,11,13,14
3,13,5,12,2,3,13,6,9,14,3,14
11,4,1,7,15,13,7,13,6,9,14,10
11,5,6,10,9,7,13,5,15,7,14,8
11,5,12,10,2,9,5,6,10,8,9,6
9,5,5,13,13,5,5,12,9,5,5,12
```



Score	Part 1 Steps	Part 2 Steps
36.633	499	19
34.667	439	19
33.4	401	19
30.7	290	19
32.867	386	19

2. Maze 2 *(14 x 14)*

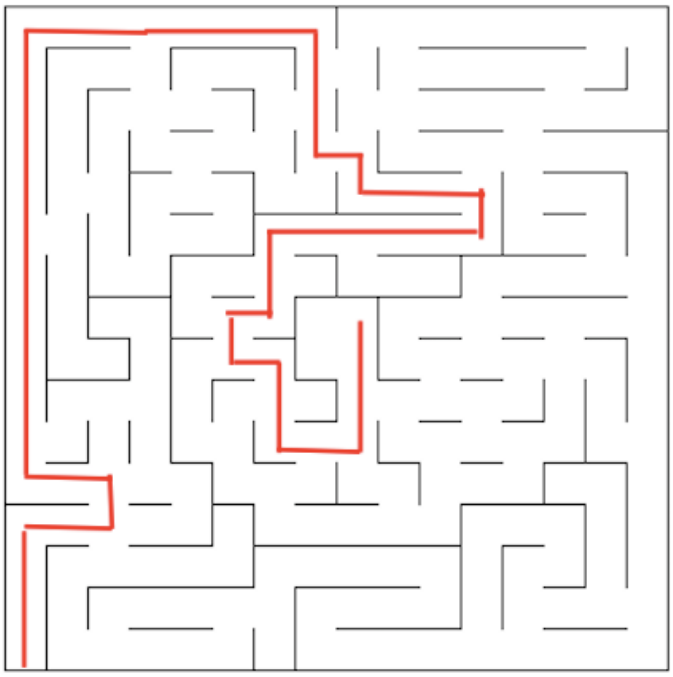
```
14
1,5,5,7,7,5,5,6,3,6,3,5,5,6
3,5,6,10,9,5,5,15,14,11,14,3,7,14
11,6,11,14,1,7,6,10,10,10,11,12,8,10
10,9,12,10,3,12,11,14,11,14,10,3,5,14
11,5,6,8,11,7,12,8,10,9,12,9,7,12
11,7,13,7,14,11,5,5,13,5,4,3,13,6
8,9,5,14,9,12,3,7,6,3,6,11,6,10
3,5,5,14,3,6,9,12,11,12,10,10,10,10
10,3,5,13,14,10,3,5,13,7,14,8,9,14
9,14,3,6,11,14,9,5,6,10,10,3,6,10
3,13,14,11,14,11,4,3,13,15,13,14,10,10
10,3,15,12,9,12,3,13,5,14,3,12,11,14
11,12,11,7,5,6,10,1,5,15,13,7,12,10
9,5,12,9,5,13,13,5,5,12,1,13,5,12
```



Score	Part 1 Steps	Part 2 Steps
44.967	658	22
42.833	594	22
40.6	527	22
47.133	723	22
52.4	881	22

3. Maze 3 *(16 x 16)*

```
16
1,5,5,6,3,7,5,5,5,5,7,5,5,5,5,6
3,5,6,10,10,9,6,3,5,5,13,7,5,5,6,10
11,6,11,15,15,5,14,8,2,3,5,13,5,6,10,10
10,10,10,10,11,5,13,5,12,9,7,6,3,15,13,14
10,10,10,9,12,3,5,6,3,6,10,11,14,11,6,10
9,14,9,4,3,13,6,11,14,10,9,12,11,12,10,10
1,13,6,3,14,3,15,12,9,15,6,3,13,7,12,10
3,6,10,10,9,14,8,3,6,8,10,9,7,13,7,12
10,10,10,10,3,13,7,13,12,3,14,3,13,7,13,6
10,10,10,11,12,3,14,3,6,10,10,10,3,15,7,14
10,9,12,9,7,14,11,14,10,8,10,10,10,10,10,10
11,5,5,6,10,11,14,11,15,6,9,13,14,10,10,10
11,7,6,10,9,14,9,14,10,10,3,7,15,14,10,10
10,10,9,12,2,9,5,15,14,10,10,10,10,11,14,10
10,11,5,5,12,3,5,12,10,11,13,12,10,10,9,14
9,13,5,5,5,13,5,5,13,13,5,5,12,9,5,12
```



Score	Part 1 Steps	Part 2 Steps
56.567	916	25
57.667	949	25
58.267	967	25
58.333	970	25
55.033	870	25

From the above results, it was evident that the A* algorithm did a very fine job in calculating the optimum path and executing it in Part 2. Since this part was consistent, the final score only depended on how quickly the algorithm could both explore all of the locations and find the goal as well (line 209 of robot.py). If these two conditions were not met, the first round continued until 970 steps, which was only necessary once in Maze 3. The reason why the number of steps taken for Part 1 varies is in the random turns it takes when there are a number of unexplored next locations. This suggests that the beginning of the exploration step, where most of the maze is

unexplored, makes the difference in the final score. For example, ideally we'd like the robot to roam around the maze in the shape of a W, going back and forth until the end is reached. However, since it is taking random turns, the robot could've went on to the right side of the maze while leaving some locations left unexplored. In this case, the robot would then have to go back towards the starting location just to mark the one or two locations as explored.

Justification

Compared to the benchmark model, it was surprising to see that the robot did either equally well or even better compared to the benchmark model I defined.

The fastest path I came up with was 20, 22, 25 for Maze 1, 2, and 3, respectively, and the robot got 19, 22, 25. This result may slightly change since I did not go through every possible solution to solve the maze myself. However, it is evident that the robot is well within the threshold, and sometimes even better and certainly more efficient than humans in solving the maze. In this sense, it is safe to say that the problem stated in the beginning of this project is solved.

V. Conclusion

Reflection

To summarize this project, the mission was to create a robot that can map out a maze and solve the maze from the map it created. Initially, I was concentrating on the second round since it seemed to be more important to have a robot that solves a maze efficiently compared to exploring a maze and mapping it out. However, it turned out that the mapping and solving process goes hand in hand, especially since there was a time limit of 1000 trials. Indeed, even applying this to the real world, it is very inefficient to have a self-driving car that takes an extremely long time mapping out its environment before the vehicle can be sold to customers. Even if some of the decisions the robot made were random in the first round, one change such as making the robot go to a location that is least explored, had large implications on the overall score. This worked out wonderfully in the end, but until then, it was very difficult to design the robot to solve this maze on a consistent basis. Also, the difficulty in this project was to understand the true meaning of the A* algorithm and when it can be used. My first attempt was to map out the walls in the first round, and use the A* algorithm in the second run. However, it was wrong to assume that the algorithm finds the optimum path at each step. In reality, the algorithm explores many locations and the optimum path can only be found at the end of the algorithm. This realization forced me to change the whole design of the project but it allowed me to learn the essence of this algorithm as well as when and how to implement this.

Although this project is limited in its possible applications, the idea can be applied to building products, which require robotics. For example, this idea can be used to build a cleaning robot that roams around and maps out the environment in the first round. From the second round onwards, it can simply go through the most efficient way to clean the entire room and come back. This also has applications

to self-driving cars, although it is more complex in nature. For example, the car can use A* or similar algorithms to get to the destination. In the mapping process, the roads can be similar to having no walls in this project, and there can be various ‘walls’ such as streetlights, pedestrians, real walls, trees, etc.

Improvement

Within the scope of this project, I believe that the performance of this project is very high and any improvements are incremental. More importantly, it seems that second round always found the optimum path, and any improvements would be an incremental one for Part 1, the mapping step. However, if I were to experiment further, I would perhaps go through other algorithms both for mapping (such as SLAM) and finding the optimum path. I would also explore other environments such as greater maze dimensions, uncertain movements, uncertain sensors, and other variables that may make this project more difficult and closer to what a self-driving car has to achieve. Overall, this project is limited in direct applicability to many of the problems in the world today. However, in terms of solving a maze, the robot has done an exceptional job in efficiently finding the best solution, even doing this better and certainly faster than most humans could.