# Capstone Project

**Machine Learning Engineer Nanodegree**

Muhammad Zeeshan Karamat

July 13, 16

# Plot and Navigate a Virtual Maze

## Definition

### Project Overview:

A robotic mouse in an unknown virtual maze is programmed to firstly plot the unknown environment and then find the optimal path from start location to the center of the virtual maze. This video (YouTube) is an example of a Micro mouse competition.

### Problem Statement:

The robotic mouse has two runs for this project. In the first run, the robot will explore and map the unknown maze surface and store its maps in its memory with a certain time limit. It will continue exploring the space no matter it has reached its goal i.e. center of the maze. Then, in the second run, the robotic mouse will exploit what it has learned in the first run and come up with optimal strategy from the start location to the end location and will plan its route in an optimal way.

Some initial code is already provided along with specifications for the robotic mouse environment and testing. I have programmed the robotic mouse to the first plot and then optimally navigate through that virtual environment. For the first run, the robot will explore as much surface as it can in minimal steps and for then for the second run, it will use artificial intelligence search methods to come up with the optimal route to reach the center of the maze.

### Metrics:

For scoring the performance of the robotic mouse, combination number of steps taken by the robotic mouse in the first run plus number of steps for the second run. For adding more weight to the second run, the first run has been divided by thirty and then added to the second run steps to make the final score. Total number of steps taken by the robotic mouse is limited to 1000.

# Analysis

## Data Exploration:

The shape of virtual maze is n*n square with goal in the center of it, which is 2*2 grid. For each cell of the grid, it might have walls in the left, right, up or down side of the cell, which will prohibit the robot motion though them. Robot will always start from the bottom left corner of the virtual maze where it can only all walls closed except for the up one.

The robot has three distance sensor mounted on it, that will measure the robot distance from its current location to the walls on their respective sides. It is assumed that the sensor readings are accurate and free from any noise. Below are these three sensors,

- Left distance measurement sensor
- Front distance measurement sensor
- Right distance measurement sensor.

For the robot motion, it can rotate either clockwise or anticlockwise direction or move straight and then move forward or backward. It is assumed the robotic motion is perfect with no probability of moving into the wrong direction. If the robot hits the wall while moving, it will stay where it is after that time step and there is not penalty for this collision. Sensors values are available after each movement of the robot.

Rotation of the robot is expressed in angles with possible values -90(counter clockwise), 0 or 90(clockwise) degrees. And the movement is expresses as an integer with possible values form -3 to 3 inclusive, which indicates the number of steps taken in the respective direction.

Mazes are provided as text files with first lines describing the dimension of the square maze and the subsequent lines indicates the allowed location for each column delimited by commas. 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), as shown below.


These maze files are only used for testing the search approach independently and for testing the robot performance, but the robot will plot the unknown virtual environment by itself, rather than from these files.

For example for the first maze provided, its visualization will be covered in the next section. Some of the interesting points that can be exploited for obtaining better score are avoiding dead-ends, avoiding loops, using initial heuristics to guide the robot towards the center of maze, and preferring the front motion as compared to turning. They are further discussed in the next section.

## Exploratory Visualization:

For the first maze, it is shown below with optimal route from bottom left corner to the center of the maze output by the program is shown below, where the arrows indicate the robot moving direction.

```
[' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ']
[' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ']
[' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ']
[' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ']
[' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ']
[' ', ' ', ' ', ' ', ' ', ' ', ' ', '<', ' ', ' ', ' ', ' ']
[' ', ' ', ' ', ' ', ' ', ' ', ' ', '^', '<', ' ', ' ', ' ']
[' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '^', ' ', ' ', ' ']
[' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '^', '<', '<', '<']
['>', 'v', ' ', ' ', '>', '>', 'v', ' ', ' ', ' ', ' ', '^']
['^', 'v', ' ', ' ', '^', ' ', '>', 'v', ' ', ' ', ' ', '^']
['^', '>', '>', '>', '^', ' ', ' ', '>', '>', '>', '>', '^']
```

and the equivalent path plotted on the actual maze is shown below, with thick black rectangle on the bottom letf corner indicating the start position of the robot and violet color lines indicating the robot trajectory.

The red rectangular blocks indicating dead-end in the maze where if the robot can enter leaves no option for further motion until rotate 180 degrees or turn back. Yellow blocks indicate some potential loops that the robot must avoid during its exploration phase to move in those loops. As during the exploration phase the robot will try to maximize the coverage of the space along with finding the optimal route to its goal, so the robot will loose its number of steps while moving in the same loop again and again.

When it comes to the optimal path, although the length of the path is 30, but considering the fact that the robot can move 3 units per move, it can achieve the above drawn path in just 17 moves. So, in the subsequent algorithm, I have preferred forward motion as compared to turning.

## Algorithms and Techniques:

As there are two runs allowed for the robotic mouse with total number of steps limited to 1000. So, during the first run, I have compared performance following algorithms,

- Random movement,
- Random movement with weighted choice based on the count frequency,
- Weighted choice based on count frequency while avoiding dead ends

- Weighted choice based on count frequency while avoiding dead ends and heuristics guide

The random movement technique will work for small grid size and will reach goal, but its coverage is low as compared to other techniques. But as the grid size increases, its performance start to degrade substantially and will not suffice to reach goal for most of cases.

Now coming to the second approach with weighed choice for the next move, I have counted frequency for each of cell of the maze during its exploration phase. And the robot uses weighted random selection based on assigning low probability to the high frequency cells and coverage is high even for large size mazes. So, it results in better coverage as compared to the previous technique, but it waste a lot of time in dead ends.

For the third technique, I also captured information about the dead-ends, using the fact that all of the three sensors values will be zero for that that location. So, while making next movement, I used that dead-end information captured in the previous movements to avoid trapping in these dead-ends by turning 180 degree. I also removed random selection for this approach and used minimum of all count frequency for all of the possible choices for giving absolute 1 probability value to the location less explored.

For the last technique, I make a initial heuristic function using information available like the center of the maze, by initializing the goal locations with zero heuristic value and then the value gradually increases in the subsequent goals. Sample initial heuristics is shown below,

```
[5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5]

[5, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 5]

[5, 4, 3, 3, 3, 3, 3, 3, 3, 3, 4, 5]

[5, 4, 3, 2, 2, 2, 2, 2, 2, 3, 4, 5]

[5, 4, 3, 2, 1, 1, 1, 1, 2, 3, 4, 5]

[5, 4, 3, 2, 1, 0, 0, 1, 2, 3, 4, 5]

[5, 4, 3, 2, 1, 0, 0, 1, 2, 3, 4, 5]

[5, 4, 3, 2, 1, 1, 1, 1, 2, 3, 4, 5]

[5, 4, 3, 2, 2, 2, 2, 2, 2, 3, 4, 5]

[5, 4, 3, 3, 3, 3, 3, 3, 3, 3, 4, 5]

[5, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 5]

[5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5]
```

So, in this technique, instead of just using previous captured frequency and dead end information, I also used this heuristics value to count for deciding next location. And the results of this approach are significantly better than as compared to the previous approaches in the first run.

For the second run, I used A STAR search method for using mapped information about the maze to output optimal route to the center of the maze using the heuristics value again compiled. Now heuristics values are compiled using the mapped information. Starting from the goal, I counted for each cell the minimum number of steps it takes to reach the goal using optimum policy algorithm. Then using that heuristics value, I have come up with optimal value for the route. This approach will outperform when the coverage of the maze is high. I also assigned more cost value to the left or right turn as compared to the forward motion while computing the G value for the A STAR search method. By that way, I will come up with more strait lines on which the robot can travel multiple units and save time later.

### Benchmark:

Comparing performance of different techniques using the criteria that 1/30 of the number of steps of first run plus the number of steps in the second run, I compared performance of all of them. The random method performance is worst among all of them, and I used that as my baseline method for further comparing performance of other advanced algorithms used for this project.

# Methodology

### Data Preprocessing:

Since the environment and robot specifications are given, so not data preprocessing steps are required for that project. Also sensors data is free from noise, so no noise removal or cleaning steps are required for this project.

## Implementation:

I implemented all of the required functionality inside the robot.py class. Firstly coming to tracking robot motion inside robot.py class for keeping track of its direction and taking respective actions, I used following constructs for defining its rotation,

```
forward = [[-1,  0], # go up
        [ 0, -1], # go left
        [ 1,  0], # go down
        [ 0,  1]] # go right

action = [1, 0, -1]

forward_name = ['up', 'right', 'down', 'left']
angle_val = [-90, 0,90]
```

This is used for taking robot to take various directions, like moving up, etc. Furthermore, forward_name list is used for defining current robot direction. And then the action list is used for mapping corresponding robot rotations to the forward list index. Like if robot want to turn right, then it will decrement its current orientation head by one as in the action list i.e. -1 and its corresponding forward list coordinates will be multiplied with its current coordinates to make its new location.

Then, initialized some variables that will be used for further logic implemented and are documented in the code. Some of them required description, and they are

```
self.count_grid  = [[0 for row in range(self.no_of_rows)]
            for col in range(self.no_of_cols)]

self.mapped_grid = [[self.init_val for row in range(self.no_of_rows)]
                    for col in range(self.no_of_cols)]

self.deadend_grid = [[0 for row in range(self.no_of_rows)]
                    for col in range(self.no_of_cols)]

self.heuristics = [[self.def_heu_val for row in range(self.no_of_rows)]
                   for col in range(self.no_of_cols)]

self.init_heuristics = [[self.def_heu_val for row in range(self.no_of_rows)]
                    for col in range(self.no_of_cols)]
self.route = []
```

**count_grid** is used for keeping frequency of visit of each cell as the robot explores the space. **mapped_grid** contain values of the mapped wall position for each cell, and they are computed as follow, 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).

**deadend_grid** is used for keeping track of dead-ends detected while exploring the mace, **heuristics** contains respective heuristics values developed after the maze is mapped and the robot will use it for calculating optimal path. **init_heuristics** is used for heuristic exploratory which is made at robot initialization with goal filled with zero values and its value is decremented for each cell after that. And **route** list contains optimal path values in the form of (rotation,movement) for the final optimal path used in the second run of robot.

Then, I used various helper method for updating and fetching values from and to the respective grids. And some other methods like

```
def is_all_space_explored(self):
    for i in range(len(self.count_grid)):
        for j in range(len(self.count_grid[0])):
            if self.count_grid[i][j] ==0:
                return False
    return True
```

which will be used for checking whether robot explores all of the maze or not by checking all elements of the count_grid values, and if any one of them is zero, then it returns true.

```
def move(self, steering, distance):

    # make a new copy
    curr_pos  = dir_names.index(self.heading)
    act_index = angle_val.index(steering)
    dir_index = (curr_pos+action[act_index])%len(dir_names)
    self.heading = dir_names[dir_index]

    delta     = forward[dir_index]
    self.location  = [ self.location[i]+delta[i]*distance for i in range(2) ]
```

and then the move method is used for updating self.**location** which is location of current position of robot and self.**heading** which contains robot current heading. These are updated by taking corresponding steering and movement by the robot.

And then there come other method like

```
def weighted_choice(self,choices):
    values, weights = zip(*choices)
    total = 0
    cum_weights = []
    for w in weights:
        total += w
        cum_weights.append(total)
    x = random() * total
    i = bisect(cum_weights, x)
    return values[i]
```

which is used for getting weighted probability of the choices based on the values of second factor. I used it for getting possible action based on the weighted probability of their respective visit frequencies.

And coming to the weighted_prob_exploration method, which is used to return weighted selection of the possible next location based on the frequency visit values. It assign very low probability to the deadend locations to avoid selecting them.

```
def weighted_prob_exploration(self,sensors_array):
    weighted_array = []
    for sensors_item in sensors_array:
        x,y = self.simulate_move(angle_val[sensors_item],1)

        if self.count_grid[x][y] == 0:
            sensor_weight = self.self_exploration_val
        else:
            if self.deadend_grid[x][y] == 1:
                sensor_weight = self.epsilon_val
            else:
                sensor_weight = self.self_exploration_val **(-self.count_grid[x][y])


        weighted_array.append((sensors_item,sensor_weight))
    return self.weighted_choice(weighted_array)
```

Similarly weighted_prob_exploration_heu method is incorporating self.init_heuristics for making next location decision based on the low value of the total of visit frequency and

self.self.init_heuristics value. And the other two remaining methods counting_exploration and counting_exploration_heuristic are used for the same purpose but they are not probabilistic but rather using minimum value of the total sum.

Two important methods for checking when to stop first run of the robot are

```
def all_space_explorered(self):
    return self.is_all_space_explored() or self.count_steps >=900

def goal_first_explorere(self):
    return self.isGoal(self.location) or self.count_steps >=900
```

all_space_explorered stops the first run, when all of the maze is explored and the second method stops the first run when the robot reached goal. One important thing about that is they both will terminate first run if the maximum number of steps taken by the robot in the first run exceeded from 900. As this will provide enough steps for robot in the second run for reaching goal.

In the **robot_exploration** method is used for exploring the maze in the first run. It can use any of the two criteria either goal_first_acheived or all_space_explored for stopping the first run. And it can also use any of the four next location decision functions for making next location decision. Those four methods are,

- weighted_prob_exploration
- weighted_prob_exploration_heu
- counting_exploration
- counting_exploration_heuristic

It also updates the self.mapped_grid for each of the cell using the sensor values based on the sensor values. In the end of this method it updates robot current location and robot rotation based on the action it has taken. When the robot meets criteria for the first run to terminate, it sets is_changed_explorat flat to true for moving onto the second run of the robot.

Now coming to the second run, after the robot has mapped maze based on the criteria, it further build new heuristic values based on the mapped grid of maze. This logic is in **build_heuristics** function. It will count the number of steps taken from each cell to reach the goal or center of maze in minimum number of steps. It will only count for the mapped cells in the first phase. It will build complete optimal heuristic function, if all of the space is explored in the first phase of exploration. Starting from center of maze, it will expand the heuristic values based on the minimum cost calculated based on rotation of robot. I assigned more cost for taking right or left turn, it will minimized the robot steps for reaching the goal, as the robot can take up to 3 units per move. Following is the heuristic grid for the first maze,

```
[39, 37, 34, 22, 21, 20, 19, 17, 19, 20, 16, 18]
[41, 35, 33, 25, 24, 23, 21, 15, 14, 12, 14, 14]
[42, 37, 32, 27, 26, 20, 19, 18, 16, 11, 13, 13]
[40, 38, 30, 28, 24, 23, 22, 20, 7, 9, 10, 11]
[44, 43, 39, 37, 27, 26, 24, 3, 5, 11, 16, 18]
[42, 41, 39, 36, 29, 0, 0, 1, 7, 9, 15, 20]
[44, 43, 37, 35, 30, 0, 0, 3, 5, 11, 13, 15]
[45, 41, 39, 33, 31, 33, 34, 9, 7, 9, 16, 14]
[46, 43, 45, 46, 32, 34, 27, 25, 8, 10, 11, 12]
[47, 40, 50, 48, 30, 29, 27, 24, 9, 23, 21, 14]
[48, 39, 38, 40, 32, 31, 25, 23, 22, 21, 20, 15]
[49, 37, 36, 35, 33, 29, 27, 21, 20, 19, 18, 16]
```

When the whole space is explored in the first run, using the all_space_explorered criteria and the number of steps taken in the second run are just 17.

After heuristic is made, I used A STAR search method for coming up with optimal route for the robot. It used heuristic value and build in the last step in the build_heuristic function and start from the start location of the maze. It will choose minimum g value next node to calculate the f value which will used as criteria for next state decision. Following is the output of the optimal path printed by the robot.py for the first maze, taking each step form start of self.route list.

```
[0, 2]
[90, 1]
[90, 2]
[-90, 3]
[-90, 2]
[90, 2]
[90, 1]
[-90, 1]
[90, 1]
[-90, 3]
[0, 1]
[-90, 3]
[-90, 3]
[90, 2]
[-90, 1]
[90, 1]
[-90, 1]
```

Based on these methods, the robot finishes all of the three maze samples provided in time threshold.

## Refinement:

Starting from crude random exploration of the robot used for exploring the maze in the first run of the project, its performance started to deteriorate as the maze size grows. So, I implemented other algorithms for improving the performance of total score. Following implemented methods are,

- Random movement,
- Random movement with weighted choice based on the count frequency,
- Weighted choice based on count frequency while avoiding dead ends
- Weighted choice based on count frequency while avoiding dead ends and heuristics guide

After random exploration, I keep track of visit frequency of each cell of the grid and used it to make next state decision. I used weighted choice probability selection method for giving more weight to the less visited locations of the maze. As a result, the coverage of the maze improved but the robot stills waste a lot of steps in the dead-ends. So, in the next refinement, I also keep track of the dead-ends faced by the robot and avoiding selecting them when the robot encounters while making decision. As a result, the robot coverage remain almost the same, but the robot will meets criteria for finishing the first run in less number of steps.

In the next phase, I used heuristic function developed at initialization for incorporating it into the decision making process. I make it such that its center is all filled with zeros and as the distance from center location increases its heuristic value also increases. And the result of this approach is significant. The robot accomplishes the first run in significantly less number of steps as compared to previous approaches.

Now coming to the criteria for stopping the first run of robot, I implemented two methods,

- all_space_explorered
- goal_first_explorered

As, when using all space explored function, the robot will come up with the optimal route for the problem but it will take more steps as the robot has to do more explorations as compared to the other approach, i.e. goal first explored. In the goal first explored, it will stop the first run as soon as the robot reaches the goal and then it will look for optimal route. Although route output from the latter approach will be sub-optimal as the robot has not done complete exploration of the maze and it might have overlooked some of the important locations which will yields optimal solution in the second run of the project. So, its basically trade-off for all_space_exploration compared to overall score i.e. 1/30 of the first run count plus the number of steps taken in the second run.

# Results

## Model Evaluation and Validation:

Since, I have applied various methods for enhancing final score, so the optimal and stable score that I have achieved using weighted heuristic explorator is listed in the below table for all of the three mazes for the goal first explored first phase stop criteria,

| Maze number | Steps in the first run | Steps in the second run | Final score |
|---|---|---|---|
| 1 | 140 | 20 | 24.664 |
| 2 | 136 | 39 | 43.533 |
| 3 | 144 | 44 | 48.8 |

When using explore all as search criteria, then the outcomes are,

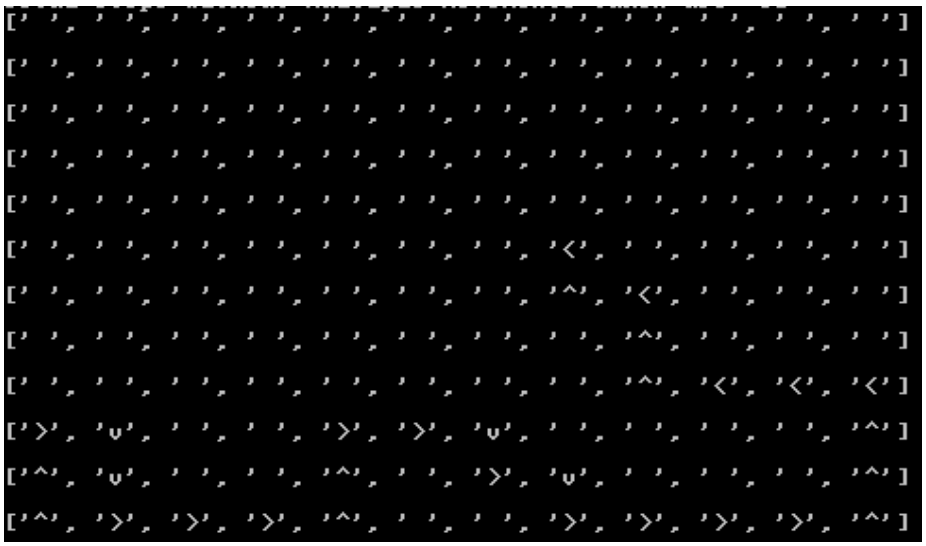
| Maze number | Steps in the first run | Steps in the second run | Final score |
|---|---|---|---|
| 1 | 289 | 17 | 26.33 |
| 2 | 593 | 24 | 43.767 |
| 3 | 894 | 28 | 57.800 |

So, when using the first approach, it is clear that robot will not come up with the optimal solution for the second run, but it uses subsequently less number of steps in the first step for exploring the space in the first run. But for the second case when we let the robot wander in the maze, by ignoring that it has reached the goal, its coverage will be 100% with a large number of steps in the first phase, but it will come up with a better strategy for the second run as depicted above. Furthermore, below is the route for the test mazes, The above diagram is printed output map of maze without walls with the following arrows showing robot directions,

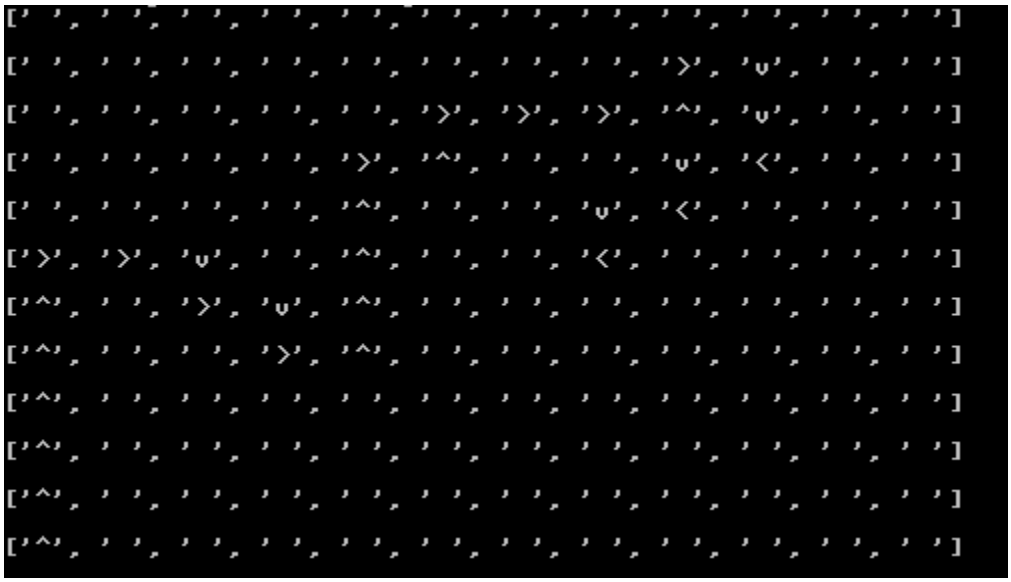
```
forward_name = ['^',        #Indicating robot moving upwards
                '>',        #Indicating robot moving left side
                'v',        #indicating robot moving downwards
                '<']        #Indicating robot moving right side
```

**Maze 1**

For the first maze, the optimal route will be as follows in the second run when the robot explores all of the space in the first run to come up with optimal solution for the second run, where starting point is on the bottom right side.

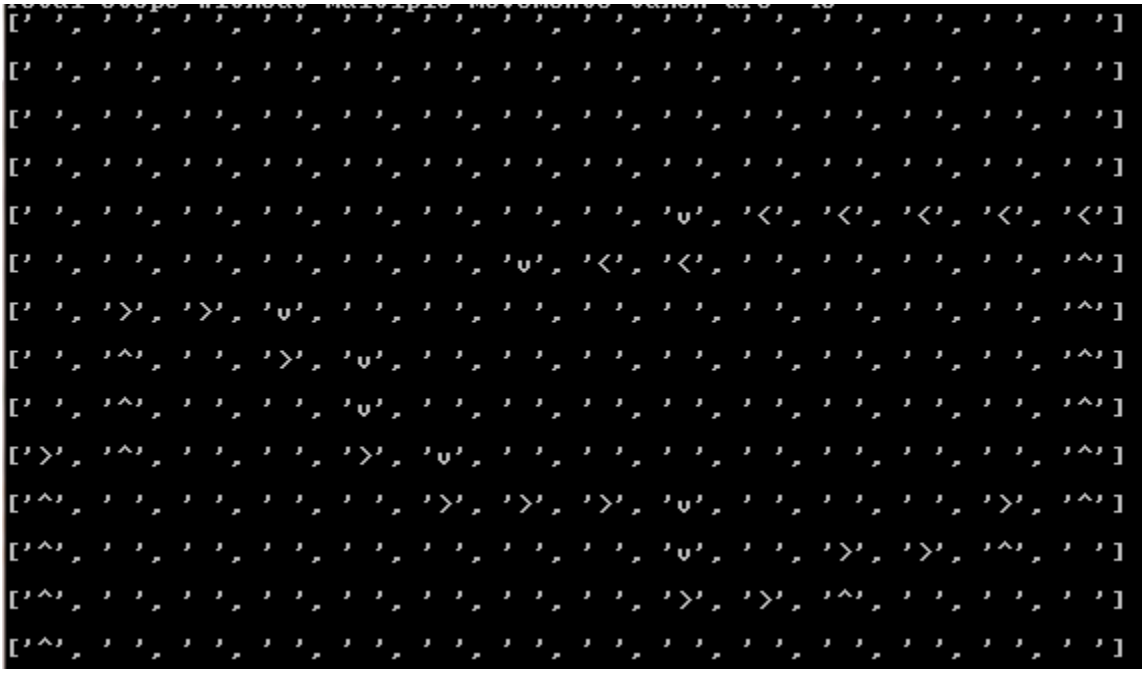


And below is the route when using goal first as exploration strategy in the first run and then coming with suboptimal path for the second run.

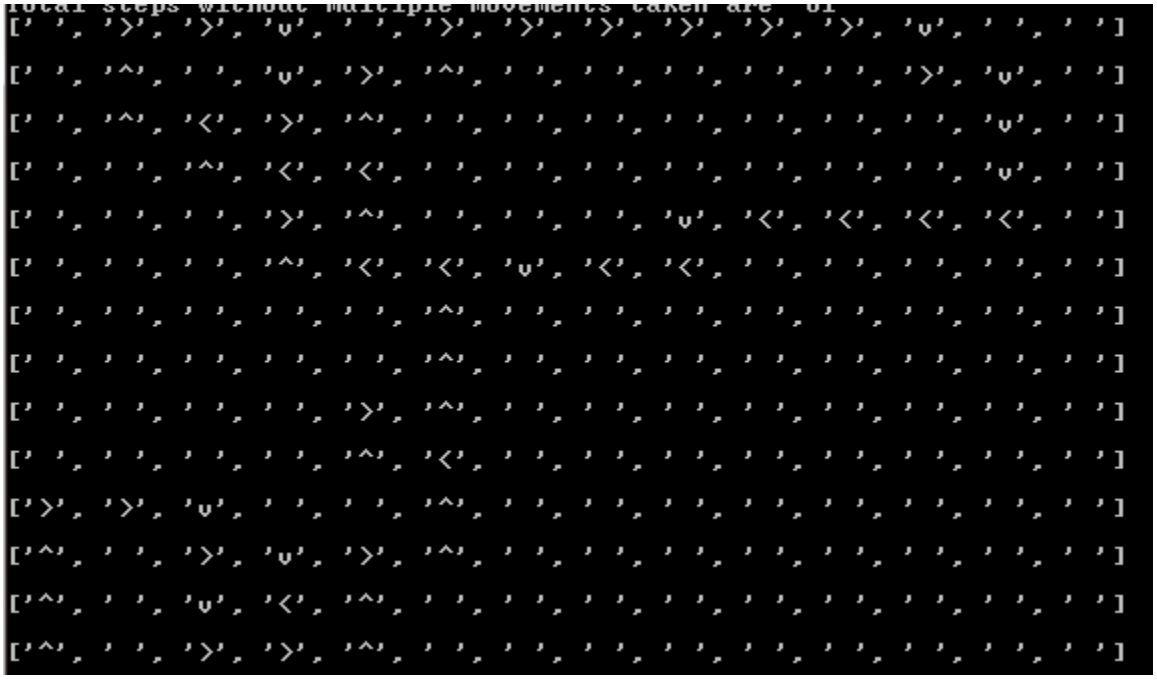


**Maze 2:**

For the second maze, the optimal route will be as follows in the second run when the robot explores all of the space in the first run to come up with optimal solution for the second run, where starting point is on the bottom right side.

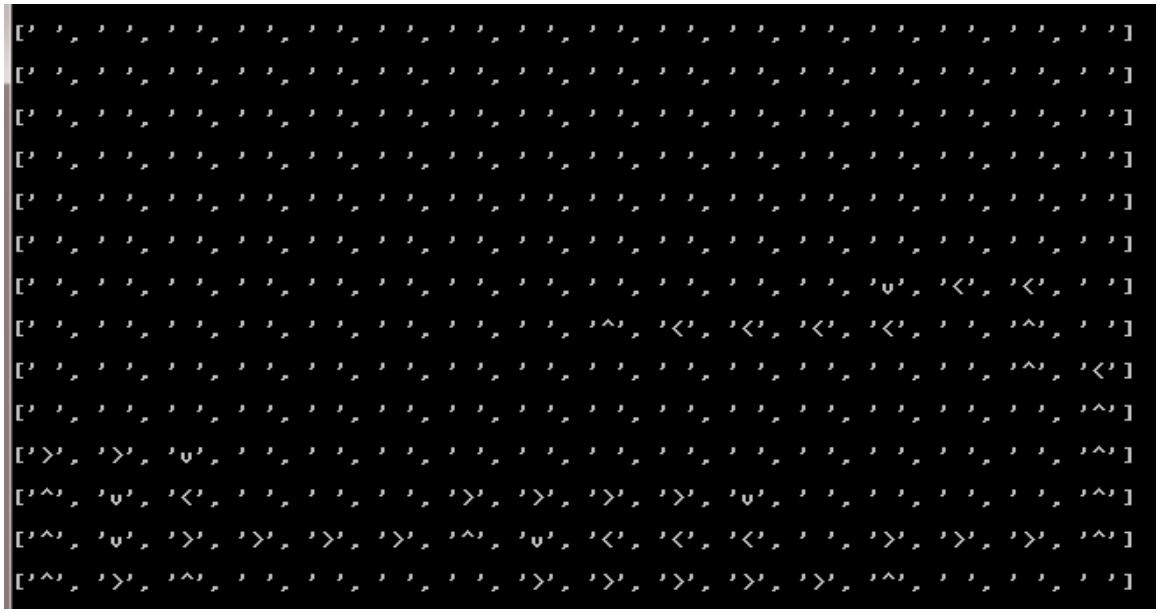


And below is the route when using goal first as exploration strategy in the first run and then coming with suboptimal path for the second run.

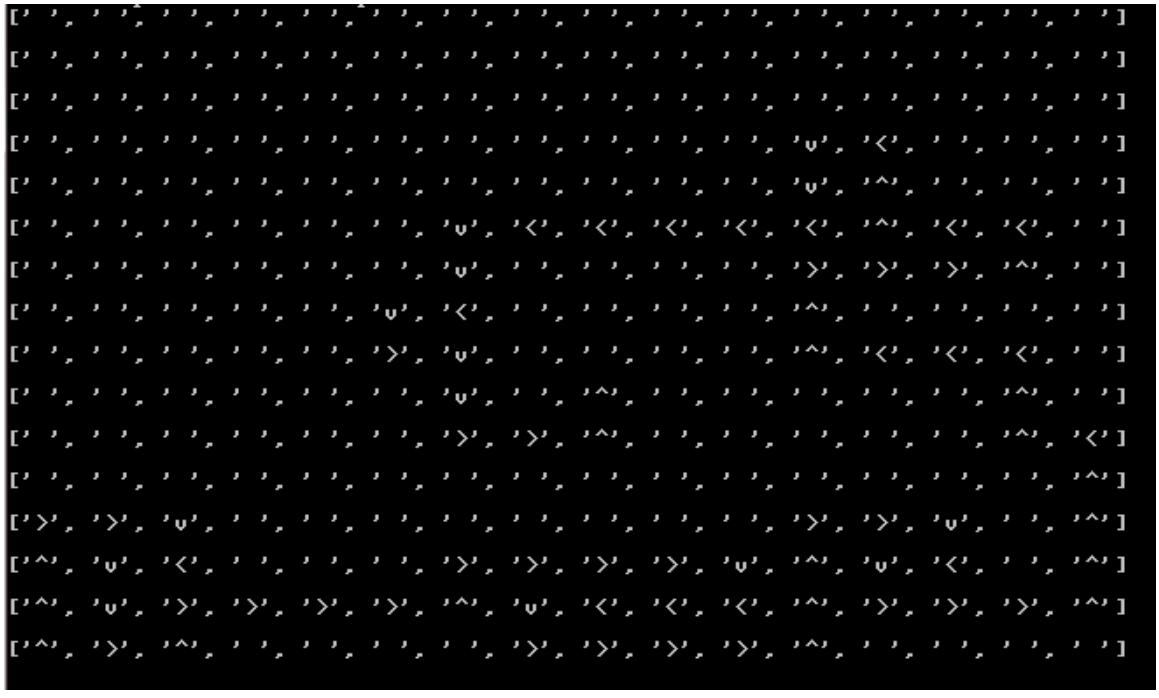


**Maze 3:**

For the third maze, the optimal route will be as follows in the second run when the robot explores all of the space in the first run to come up with optimal solution for the second run, where starting point is on the bottom right side.



And below is the route when using goal first as exploration strategy in the first run and then coming with suboptimal path for the second run.



So, considering the dead-ends, loops, and optimal path search, the robot effectively avoid the dead-ends, one it detects them, it will prohibit moving robot into them. As a result, the robot will spend considerable time as compared to the just using random decision making. And when it comes to avoiding loops in the grid, using the visit frequency based weighted choice decisions, the robot will coverage increases substantially as compared to the previous two exploratories and is vivid in the final outcome of the optimal path by using all search method that it uses less time. Further details of previous controllers are provided below,

When using just **random explorator** for the first run and using A STAR for the second run, and goal first as stopping criteria, following are the results. Since, it uses random choices, so following are three outputs for each of the testing maze,

**For maze 1:**

| Maze number | Steps in the first run | Steps in the second run | Final score |
|---|---|---|---|
| 1 | 496 | 18 | 34.533 |
| 1 | 406 | 21 | 34.533 |
| 1 | 668 | 17 | 39.267 |

**For maze 2:**

| Maze number | Steps in the first run | Steps in the second run | Final score |
|---|---|---|---|
| 2 | 664 | 23 | 45.133 |
| 2 | 732 | 26 | 50.4 |
| 2 | 758 | 31 | 53.267 |

**For maze 3:**

| Maze number | Steps in the first run | Steps in the second run | Final score |
|---|---|---|---|
| 3 | 557 | 33 | 51.567 |
| 3 | 764 | 27 | 52.4 |
| 3 | Not reached goal | Not reached goal | Not reached goal |

Now coming to using **weighted random explorator** with dead-end avoiding, and goal first as stopping criteria, following are the results in the below tables for each of the maze,

**For maze 1:**

| Maze number | Steps in the first run | Steps in the second run | Final score |
|---|---|---|---|
| 1 | 347 | 17 | 28.564 |

| | | | |
|---|---|---|---|
| 1 | 352 | 17 | 28.733 |
| 1 | 239 | 20 | 27.967 |

**For maze 2:**

| Maze number | Steps in the first run | Steps in the second run | Final score |
|---|---|---|---|
| 2 | 398 | 26 | 39.267 |
| 2 | 235 | 30 | 37.833 |
| 2 | 306 | 12 | 33.200 |

**For maze 3:**

| Maze number | Steps in the first run | Steps in the second run | Final score |
|---|---|---|---|
| 3 | 535 | 28 | 45.833 |
| 3 | 312 | 27 | 37.4 |
| 3 | 476 | 34 | 49.876 |

As, using weighted average of the visit frequency and dead-end avoidance, the final score and coverage improves, and make the robot more robust towards dead-ends and loops. Thus using efficient utilization of the time.

Now coming to using **weighted random exploratory with heuristic values incorporated** with dead-end avoiding, and goal first as stopping criteria, following are the results in the below tables for each of the maze,

**For maze 1:**

| Maze number | Steps in the first run | Steps in the second run | Final score |
|---|---|---|---|
| 1 | 104 | 24 | 27.467 |
| 1 | 332 | 17 | 28.067 |
| 1 | 138 | 18 | 22.600 |

**For maze 2:**

| Maze number | Steps in the first run | Steps in the second run | Final score |
|---|---|---|---|
| 2 | 233 | 27 | 34.676 |
| 2 | 101 | 28 | 31.367 |
| 2 | 146 | 42 | 46.861 |

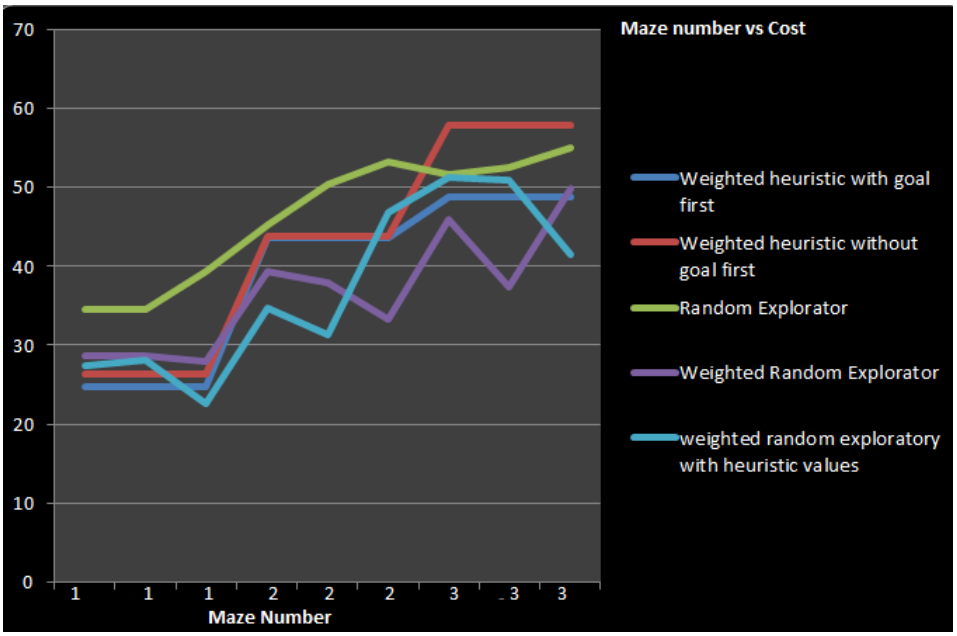
**For maze 3:**

| Maze number | Steps in the first run | Steps in the second run | Final score |
|---|---|---|---|
| 3 | 636 | 30 | 51.20 |
| 3 | 388 | 38 | 50.933 |
| 3 | 463 | 26 | 41.433 |

Now come to evaluate its performance, now the robot uses very few steps in the first run for most of the cases (since it is probabilistic) and the final score is good as compared to the previous explorators.

## Justification:

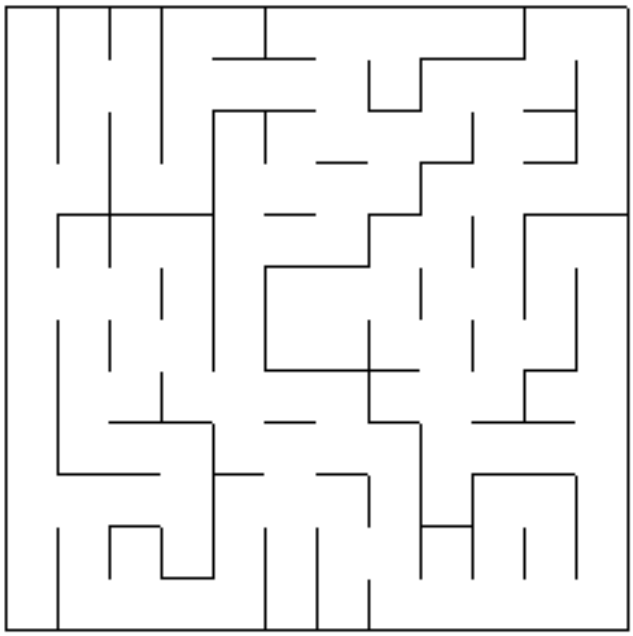
Based on the above results, the best performance is obtained in the final controller in which I used a combination of minimum frequency visit, dead-ends and heuristics values for exploring the maze in the first run with using is a goal achieved stopping criteria for the first run. And then using A-STAR approach for the second run to come up with the optimal solution. And based on the below graph comparing various approaches, then it is clear the weighted heuristic with the goal first outperforms and gave the stable values for the output route. As compared to other approaches, like weighted random exploratory with heuristics values, it gave good performance in many cases, but it will perform badly in some cases as it is probabilistic.



# Conclusion

## Free-Form Visualization:

Since my controller is robust in avoiding loops and dead-ends, so I modified maze 1 with more loops and dead-ends. But it will not impact the performance of exploration and optimal path output of the robot. Below is the new maze,



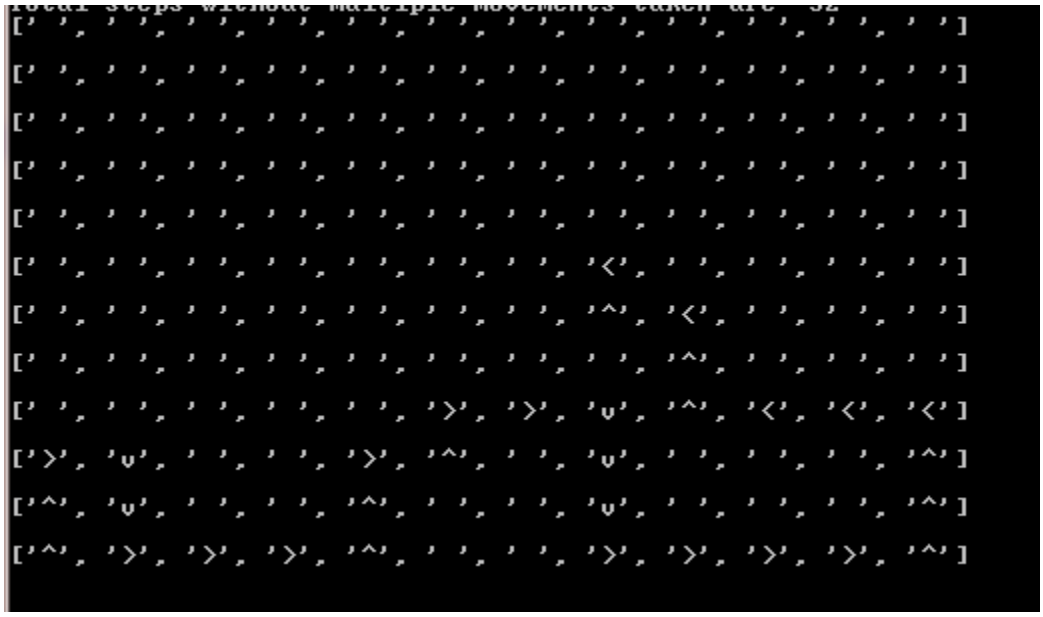
And following are the results for this maze for the **weighted heuristic explorator**

| Maze number | Steps in the first run | Steps in the second run | Final score |
|---|---|---|---|
| 4 | 58 | 23 | 24.933 |

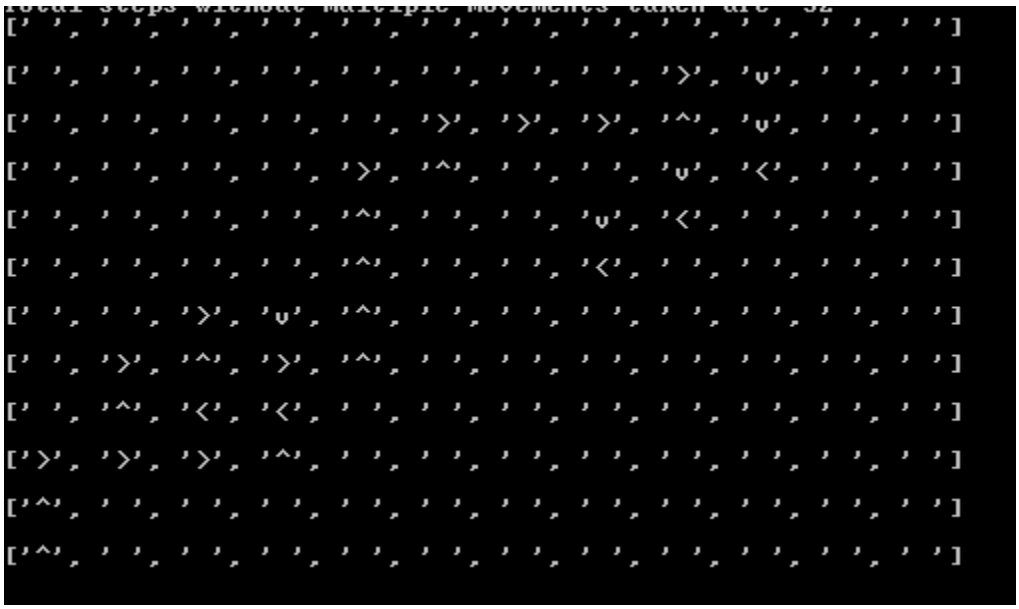
And applying other explorators for this maze are as follows, for **weighted probabilistic using heuristic** values**,**

| Maze number | Steps in the first run | Steps in the second run | Final score |
|---|---|---|---|
| 4 | 276 | 17 | 26.20 |
| 4 | 122 | 17 | 21.067 |
| 4 | 131 | 17 | 21.367 |

And the optimal path output for that maze using **weighted using heuristic** values is shown below,



And the optimal path output for that maze using **weighted probabilistic using heuristic** values is shown below,



As for the first approach, although the robot spent less time for the first run and explores less space, therefore come up with less optimal path as compared to the first approach in which the robot explores the maze using probabilistic model and counting more number of steps for the first phase and come up with optimal route towards the goal.

## Reflection:

While starting this project, I came up with different approaches for using in that project for coming up with the optimal solution for this project. At start, most of my time was consumed with working on setting various grid parameters and selecting methods for using in that case. After setting some initial settings used for robot exploration, I explored the maze using random search, in which I faced various problems like the robot was spending more time on dead-ends. So, after debugging that issue, I came up with tracking the dead-ends for the robots and avoiding them maximally. So, in the next iteration, I got problems like making next decision which will be useful in such a way that the robot explores maximum of the space and also move in the direction towards the goal. So, I came up with the idea of also keeping track of visit frequency for each of the cells and while making the decision for the next state of the robot, I also incorporated it for avoiding the potential loops that avoid robot to explore that loops less time and spend more time on remaining cells. And for moving the robot towards the goal of exploring more area around the goal, I came up with an initial heuristic grid for giving more values to the next location cell that moves robot more towards the goal. And after that, the results improves significantly and the robot makes efficient utilization of exploration time in the first run.

When coming up with second run challenges, I have various ideas for the second run like using A STAR, optimum policy function, or first search for using a maximum of the mapped grid of the first step. So, I redeveloped the heuristic function for using optimum policy to come up with a minimum number of steps required for each of the cells to reaching the goal. Then, I used that heuristic function for the A STAR search method for coming up with an optimal path towards the goal. Then I exploited the use of taking up to three steps at a given time, so I gave more cost to the left or right turns as compared to forward motion.

## Improvement:

In reality, everything is in the continuous domain, while for this project, it is assumed to be in the discrete domain. So for that you also need other algorithms like PID, smoothing, SLAM(for both updating location and mapping at the same time) etc. Also being an electrical engineer, it will make insanely difficult when it comes to making it physically. Like there come sensors noises, erroneous readings, sensor calibrations, no accurate rotations, and movements, and another robot physical design constraints. So, they can be addressed and can be further explored for improvements.

Overall for this project, I explored various online courses like Udacity Artificial intelligence for robotics and control system course and applied various new methods for coming up with an optimal solution. I came up with new exploration ideas for the first run, like cutting the robot position into diagonal based on the robot location into three areas and then counting the average of each of each area for utilizing that in making

further decision and ultimately coming with up better approach, but testing it will not improve robot performance as compared to heuristic function and prove technically challenging. Also for the sensor data available, I can make use of it maximally like for above locations too, as the sensor measures distance for the next possible location, so I can use that value for mapping for the other cells too, which robot has not visited yet. Also, I can make use of backward motion for coming up with the optimal path, but it does not prove pretty useful.