Documentation

I. Instructions to run the program:

1. The program is written in Python. To run the program, open a command terminal and run: *python RandOptPlanner.py*
2. The inputs to the program are world_state, robot_pose and goal_pose. These can be changed in the input section (lines 22 to 33) of the *RandOptPlanner.py* file.
3. On running the program, we get the following output:

```
a) Random Path
[0, 0] -> [0, 1] -> [0, 2] -> [0, 3] -> [1, 3] -> [2, 3] -> [3, 3] -> [4, 3] -> [5, 3] -> [5, 4] -> [5, 5] -> [4, 5] .

Grid =
['>', '>', '>', 'v', ' ', ' ']
[' ', ' ', ' ', 'v', ' ', ' ']
[' ', ' ', ' ', 'v', ' ', ' ']
[' ', ' ', ' ', 'v', ' ', ' ']
[' ', ' ', ' ', 'v', ' ', '*']
[' ', ' ', ' ', '>', '>', '^']

Iterations :  26

Random Search Pass

b) Optimum Path
[0, 0] -> [0, 1] -> [0, 2] -> [1, 2] -> [1, 3] -> [2, 3] -> [2, 4] -> [2, 5] -> [3, 5] -> [4, 5] .

Grid =
['>', '>', 'v', ' ', ' ', ' ']
[' ', ' ', '>', 'v', ' ', ' ']
[' ', ' ', ' ', '>', '>', 'v']
[' ', ' ', ' ', ' ', ' ', 'v']
[' ', ' ', ' ', ' ', ' ', '*']
[' ', ' ', ' ', ' ', ' ', ' ']

Iterations :  20

Optimal Search Pass
```

II. Algorithm:

a) Random Planner:

- It has been implemented in the *RandomSearch(world_state,robot_pose,goal_pose)* function.
- The algorithm starts from the source cell and opens its adjacent cells in a random pattern. Class *random.shuffle()* has been used to generate pseudo-random numbers between 0 and 3 to select one of the cell directions to move forward in.
- The opened cells are added to an open list *open[]*
- The cells that have been selected for moving further are added to the closed list *closed[]*.
- The maximum number of iterations is set at 1000.(line 65, max_step_number)
- The algorithm will not attempt to visit a cell that was visited in the last sqrt(max_step_number) steps except if this is the only available option because of the open[] and closed[] set conditions placed in the program.

b) Optimal Planner:

- It has been implemented in the *OptSearch(world_state,robot_pose,goal_pose)* function.
- An A* algorithm has been implemented to find the optimal path to the goal.
- The search starts from the source cell and expands the adjacent cells orthogonal to it.
- The opened cells are added to an open list *open[]*
- The cells are assigned two costs, the forward moving cost *g* and a heuristic cost *h.* The forward moving cost *g* is calculated as the sum of the total distance travelled from source till the current cell. The heuristic cost *h* is calculated as the Euclidean distance from the current cell to the goal. A total cost *f* is calculated as the sum *g+h*.
- From the open list the cell with the least cost function *f* is removed and is included in the path by adding it's location to the closed[] grid and *action[]* grid after satisfying the required conditions.
- A *policy[]* function is used to keep track the path.

III. Performance Comparison:

The number of iterations for each of the approach has been included in the output and the optimal planner has lesser iterations in most cases.

Since a similar approach has been maintained to traverse cells during expansion, the difference between random and optimal algorithm's performance comes mainly due to the cost function and heuristics included in optimal planner.

The time complexity of optimal planner is approx. O(E) , where E is the number of one-to-one connections between adjacent cells. The Random planner also has a similar time complexity, but is not as efficient as optimal planner because heuristic and forward cost is not considered.