RBE 550 Motion Planning

Assignment 2

Aadesh Varude
avarude@wpi.edu

Q1 . Dikjstra's Explained

- In Dikjstra's I have initialised the cost map as the size of the grid and the obstacles as 1000 and the other paths as 1. Also, initialise the distance matrix with infinity as the distance value.
- Start Dikjstra's traversal from the first cell, i.e. given to us as a start, by setting the distance value at that point as 0.

```python
new_grid=np.array(grid).copy()
#Make the obstacels with higher values 10000
new_grid*=10000
# Make every other node as one
new_grid=new_grid+np.ones((len(grid),len(grid[0])))
# The distance map
dist=np.ones((len(grid),len(grid[0])))*np.Inf
```

- Initialize a boolean matrix to mark the visited cells of the matrix. Mark the cell as visited while traversing through the path.Initial a parent grid of the same size as the map in order to keep track of the path.

```python
parent=np.negative(np.ones((len(grid),len(grid[0])),dtype=object))
 visited=np.zeros((len(grid),len(grid[0])))
```

- Iterate in the while loop.
  - In the for loop check for the validity of the current position and explore the node as per the given sequence if the position is not visited then calculate the distance.
  - Update the distance if it is greater than the current calculated distance.

```python
           # direction priority for exploration
      r=[[0, 1], [1, 0], [0, -1], [-1, 0]]
              next_x=x+i[0]
              next_y=y+i[1]
              if(if_valid(next_x,next_y,len(grid),len(grid[0])) and grid[next_x][next_y]==0):
                  if(visited[next_x][next_y]==0):
                      distance=dist[x][y]+new_grid[next_x][next_y] # if not visited calculate the distance
                      if distance < dist[next_x][next_y]:
                          dist[next_x][next_y]=distance
                      if(parent[next_x][next_y]==-1):
                          parent[next_x][next_y]=(x,y)
          visited[x][y]=1
```

  - For every next node to be selected you need to find the minimum distance value and pick that node accordingly.

```python
     def get_next_node(a,visited,n):
          d=a.copy()
          d[np.where(visited)]=np.Inf
          ind=np.argmin(d)
          x=ind // n
          y=ind % n
          return x,y
```

- If you find the goal then backtrack the path using the parent matrix.

```python
    def path_finder(start,goal,parent):
path=[]
x=goal[0]
y=goal[1]
path.append(goal)
# print("goal pose",goal)
while parent[x][y]!=(start[0],start[1]):
    # print("values appending in path",[parent[x][y][0],parent[x][y][1]])
    path.append([parent[x][y][0],parent[x][y][1]])
    # print(parent)
    x1=parent[x][y][0]
    y=parent[x][y][1]
    x=x1

path.append(start)
path.reverse()
return path
```

2 . A star Explained

- In A star I have initialised the cost map as the size of the grid and the obstacles as 1000 and the other paths as 1. Also, initialise the f matrix with infinity and the g values as zeros of the size of the grid.
- Start A-star traversal from the first cell, i.e. given to us as a start, by setting the g and f value at that point as 0.

```
new_grid=np.array(grid).copy()
#Make the obstacels with higher values 10000
new_grid*=10000
# Make every other node as one
new_grid=new_grid+np.ones((len(grid),len(grid[0])))
# Initialising the g and the f map values
g=np.ones((len(grid),len(grid[0])))*0
f=np.ones((len(grid),len(grid[0])))*np.Inf
```

- Initialize a boolean matrix to mark the visited cells of the matrix. Mark the cell as visited while traversing through the path.Initial a parent grid of the same size as the map in order to keep track of the path.

```
parent=np.negative(np.ones((len(grid),len(grid[0])),dtype=object))
 visited=np.zeros((len(grid),len(grid[0])))
```

- Iterate in the while loop.
  - Within the for loop check for validity condition of traversal and obstacles-free nodes and visited nodes.
  - In the loop check for the validity of the current position and explore the node as per the given sequence if the position is not visited then calculate the f and g values.
  - Update the f_val if it is greater than the current calculated f_val.
  - The h_value manhattan distance between the current node and the goal node.

```
       # direction priority for exploration
  r=[[0, 1], [1, 0], [0, -1], [-1, 0]]
          next_x=x+i[0]
          next_y=y+i[1]
          if(if_valid(next_x,next_y,len(grid),len(grid[0])) and grid[next_x][next_y]==0):
              if(visited[next_x][next_y]==0 ):
                  g_val=g[x][y]+new_grid[next_x][next_y] # if not visited calculate the distance
                  h_val=abs(next_x-goal[0])+abs(next_y-goal[1])
                  f_val=g_val+h_val # calculate the f value
                  g[next_x][next_y]=g_val # update the g value
                  if f_val < f[next_x][next_y]:
                      f[next_x][next_y]=g_val+h_val # update the f value
                  if(parent[next_x][next_y]==-1):
                      parent[next_x][next_y]=(x,y) # adding the parent node
      visited[x][y]=1
```

  - For every next node to be selected you need to find the minimum f_value and pick that node accordingly.

```
     def get_next_node(a,visited,n):
          d=a.copy()
          d[np.where(visited)]=np.Inf
          ind=np.argmin(d)
          x=ind // n
          y=ind % n
          return x,y
```

- If you find the goal then backtrack the path using the parent matrix.

```
    def path_finder(start,goal,parent):
path=[]
x=goal[0]
y=goal[1]
path.append(goal)
# print("goal pose",goal)
while parent[x][y]!=(start[0],start[1]):
    # print("values appending in path",[parent[x][y][0],parent[x][y][1]])
    path.append([parent[x][y][0],parent[x][y][1]])
    # print(parent)
    x1=parent[x][y][0]
    y=parent[x][y][1]
    x=x1

path.append(start)
path.reverse()
return path
```

Difference and Similarities between A star and Dikjstra's.

- The main difference between the two algorithms is that A-star uses the addition heuristic value as the manhattan distance, whereas Dijkstra's just relies on the distance cost value.
- If we set the heuristic as h=0 then essentially the A stars turn into DIkjstra's algorithm.
- The major advantage of A star is that it takes less steps to find the optimal path than Dikjstra's.

Results

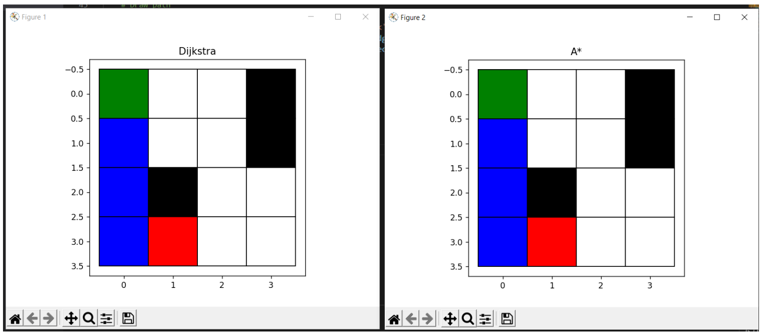

Figure 1: Images of A star takes 7 steps and Dikjstra's takes 10 steps on the given testmap csv file.

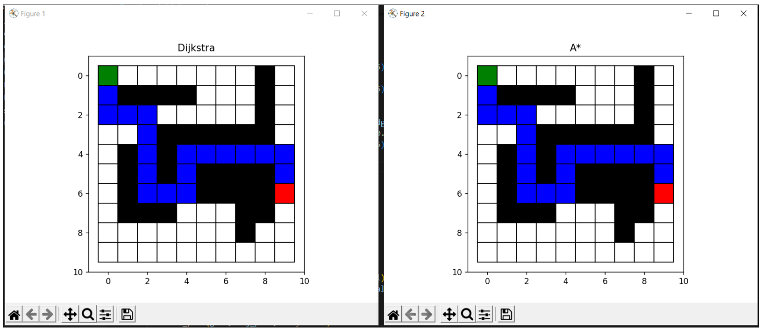

Figure 2: Images of A star takes 44 steps and Dikjstra's takes 64 steps on the given map csv file.

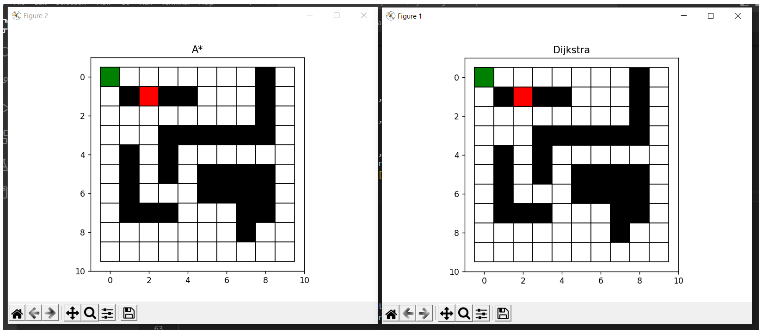

Figure 3: Images of A star and Dikjstra's on the given map when the obstacle is the goal.

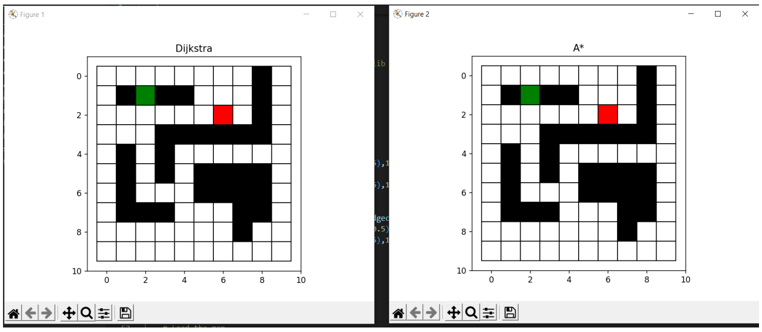

Figure 4: Images of A star and Dikjstra's on the given map when the obstacle is the start.

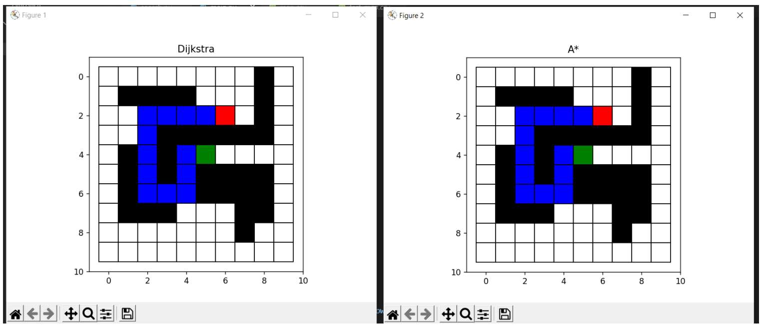

Figure 5: Images of A star takes 26 steps and Dikjstra's 58 steps on the given map with different start and goal point.

References

1. https://lunalux.io/dijkstras-algorithm-for-grids-in-python/
2. https://medium.com/@nicholas.w.swift/easy-a-star-pathfinding-7e6689c7f7b2