CSE 276C Homework 5

Adrian L. Pavlak
U09830551

10. December 2023

Homework 5 - CSE 276 - Math for Robotics
Due: Sunday, 10 December 2023

The world model is shown in figure 1. The robot is a differential drive system with a square geometry of size 50x50.

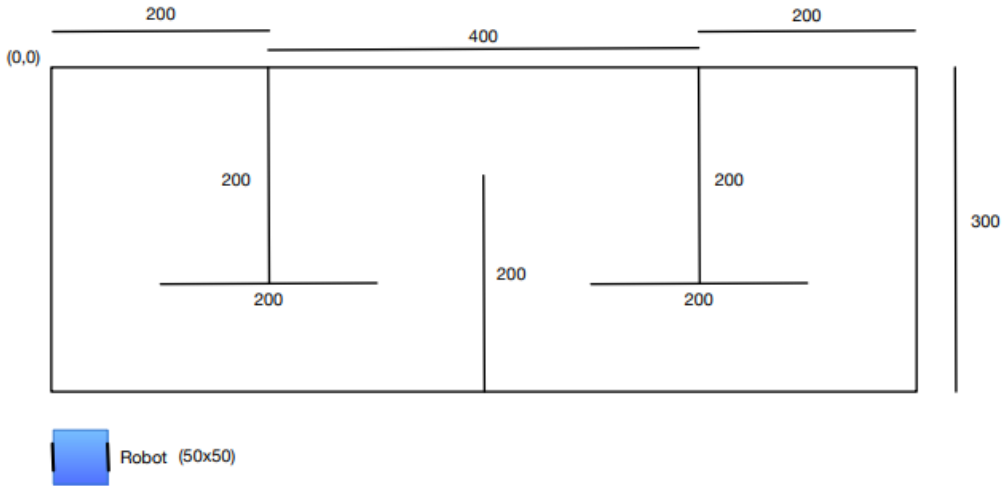


Figure 1: Homework 5 Problem formulation.

1 Problem 1

Generate the configuration space for the robot with a grid size of 2x2 and 5 deg in angular resolution. Generate an illustration of what the configuration space looks like with the robot at orientations 0, 45 and 90 deg.

Answer: In constructing the configuration space, I initiated by creating a 2D NumPy zeros array with dimensions width = 800 and height = 300. Subsequently, I implemented a function to incorporate walls and a border around the space. This function assigned values of 0 and 1 to represent open space and walls, respectively. When adding walls, I ensured a square thickness corresponding to the grid resolution of 2x2.

Following that, I introduced a robot of size 50x50 and placed it within the grid, along with defining start and end points. The resulting grid is visualized in Figure 2. Once the grid was established, I proceeded to generate the configuration spaces.

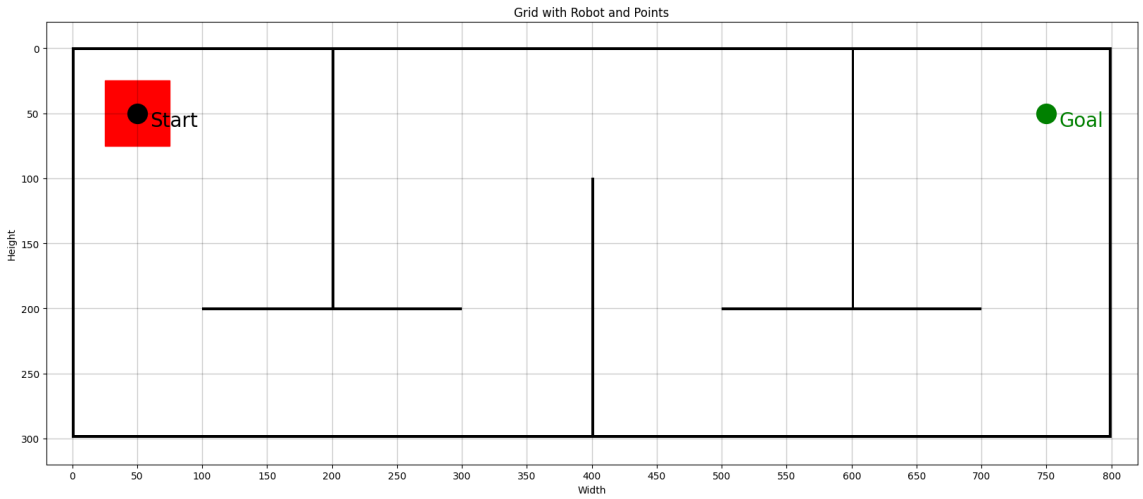


Figure 2: Overview of the grid.

To find the configuration space, I performed convolution on the grid containing only walls and borders using different masks. The convolution utilized the `convolve2d` function with `mode=same` to maintain consistent dimensions. The masks, representing various orientations of the robot due to the 5-degree angular resolution, were generated for angles (0, 5, 15, ..., 180)°. Examples of these masks are illustrated in Figure 3.

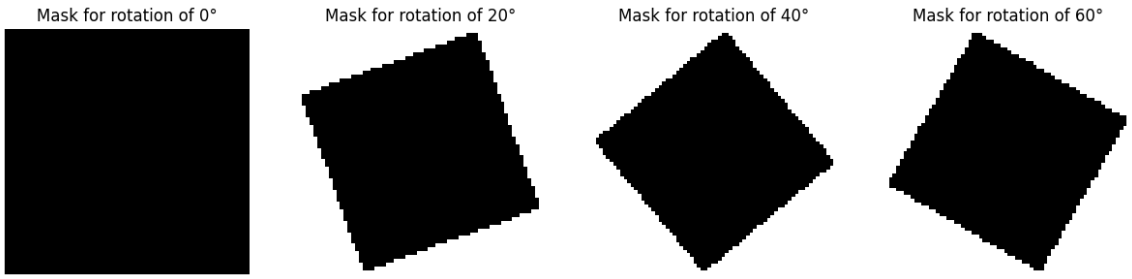


Figure 3: Different mask for different configurations spaces.

The resulting convolutions produced a new 2D NumPy array, where non-zero values indicated

areas where the mask/robot intersected with walls. Subsequently, I set all values larger than 0 to 1 and saved each new 2D configuration space into a 3D NumPy array of dimensions 800x400x36. Figures 4, 5, and 6 depict the configuration spaces for robot orientations of 0°, 45°, and 90°, respectively.

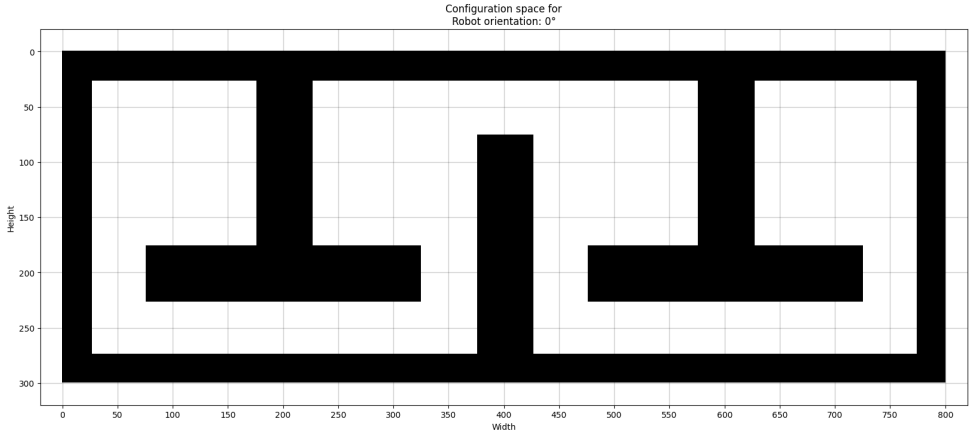


Figure 4: Configuration space for robot rotation 0 °.

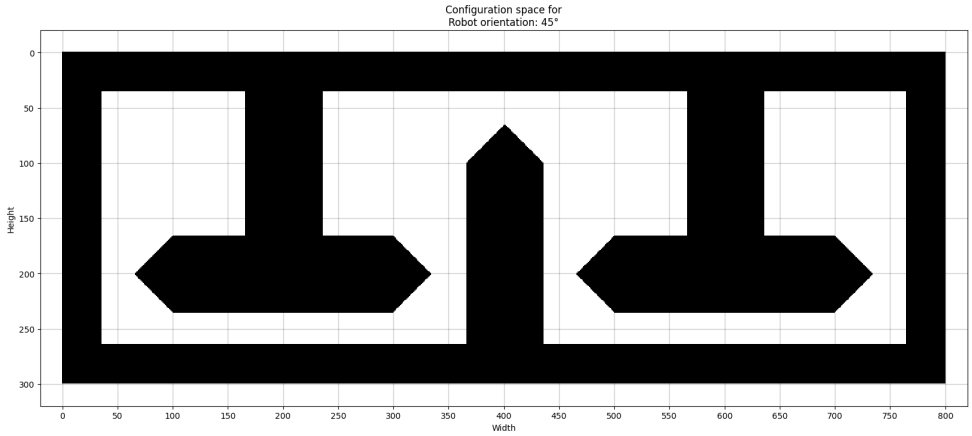


Figure 5: Configuration space for robot rotation 45 °.

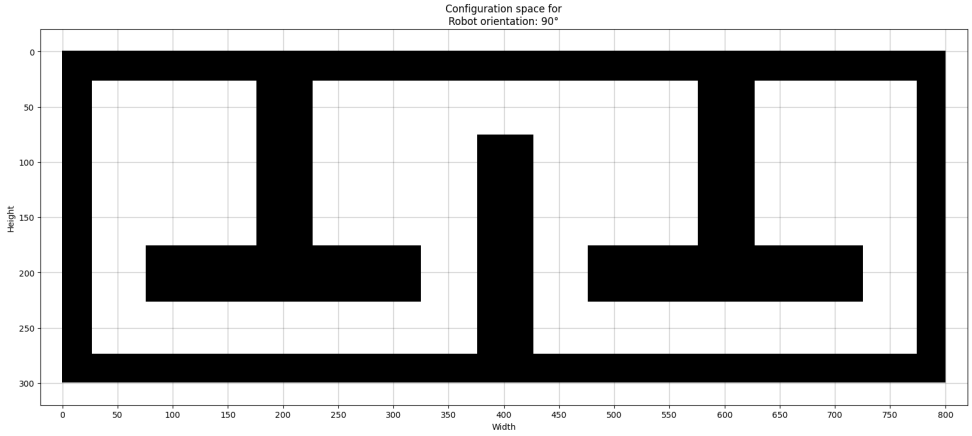


Figure 6: Configuration space for robot rotation 90 °.

2 Problem 2

Use greedy search to find the shortest path between start-point (50,50) and end-point (750,50). Illustrate the path and provide its length.

Answer: To find the shortest path between the start-point (50,50) and end-point (750,50) I employ a modified Greedy Dijkstra algorithm [1]. This adaptation involves assigning reduced weights to edges connecting nodes that are closer to the goal, enhancing the algorithm's inherent greediness by favoring nodes closer to the goal during the cost evaluation process.

The Djikstra alogirthm works like this:

1. Each grid cell corresponds to a node in the graph, allowing the robot to transition from one node to another. In our scenario, the graph comprises 800x300x36 nodes, reflecting the configuration space dimensions.

2. We assign weights to the edges connecting nodes based on their spatial relationship. In our case, a weight of 2 is assigned between free spaces in the same orientation, 1 for cells in the same position but with different orientations, and ∞ for edges between free space and an obstacle (denoted as 1 in the configuration space). Additionally, the weight is adjusted inversely based on the proximity of the node to the goal, with closer nodes having smaller weights and more distant nodes having larger weights. This makes the algorithm more greedy.

3. Initiating the algorithm involves setting the initial cost of the starting node to 0, while the cost to all other nodes is initialized to ∞. A priority queue (min-heap) is employed to organize nodes based on their current costs.

4. During each iteration, the algorithm selects the node with the minimum current cost from the priority queue. The costs of neighboring nodes are updated if a lower-cost path is identified. This iterative process continues until the goal node is reached or the entire graph is explored.

5. The algorithm terminates when the goal node is reached, and the shortest path is found.

I implemented the algorithm to equip each node with 10 neighbors—8 from its immediate surroundings and an additional 2 obtained by moving 5 degrees either clockwise or counterclockwise.

Then I ran the modified Djikstra and got the shortest path shown in figure 4 with a path length of 1132.

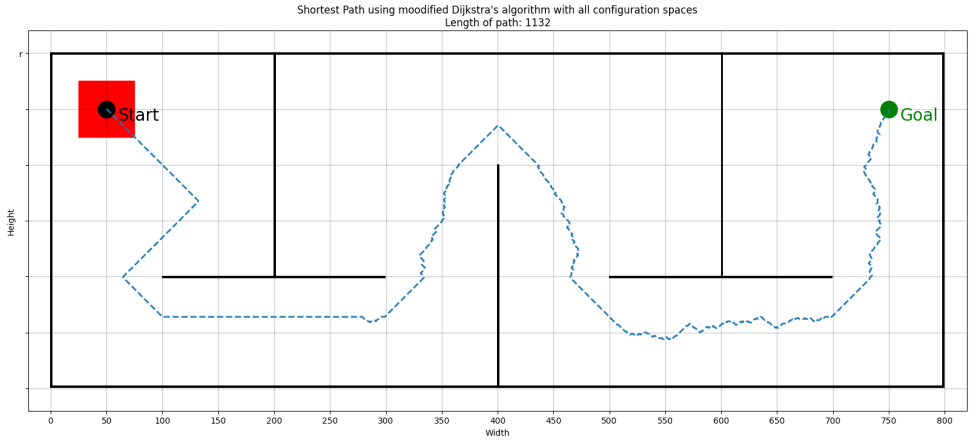


Figure 7: Shortest path using modified Djikstra. Pathlength: 1132.

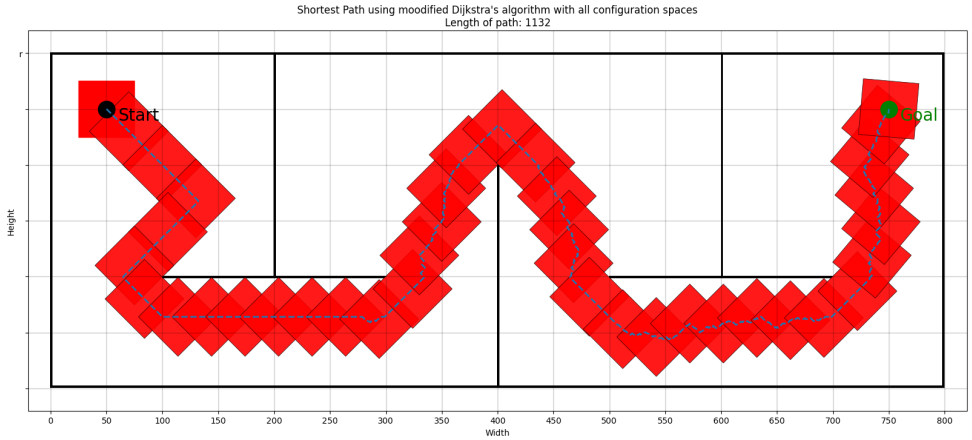


Figure 8: Shortest path using modified Djikstra showing the rotations of the robot. Pathlength: 1132.

You can see the result of using the greedy property as the path starts going to the target at the beginning and then finding it has to move left to get around the boundary. It also uses the ability to change orientation to more efficiently go around the walls.

3 Problem 3

Compute the safest path from start to finish (hint: medial axis transform/Voronoi). Illustrate the path and provide its length.

Answer: Because of the way I implemented the grid and configuration spaces, that is a 2D NumPy array of 0 is free space and 1 is a wall I could find the safest path with the Skeletonization/Medial Axis Transform using the skeletonized function from skimage.

Skeletonization, or the creation of the Medial Axis, involves transforming a complex shape into its essential core, representing the central axis of the structure. In the context of finding the safest path from A to B, skeletonization aids by identifying the most structurally significant and open regions, facilitating the determination of a path that minimizes potential obstacles and maximizes safety.

I used the Skeletonization on the configuration space for 0 °shown in figure 9 and then I used the Skeletonization on every configuration and plotted each skeleton on top of each other shown in figure 10.

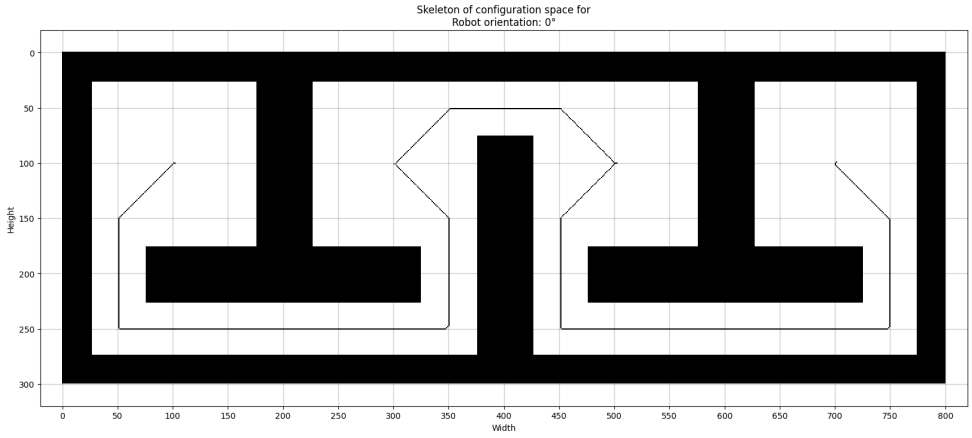


Figure 9: Skelethon for robot in orientation 0 °.

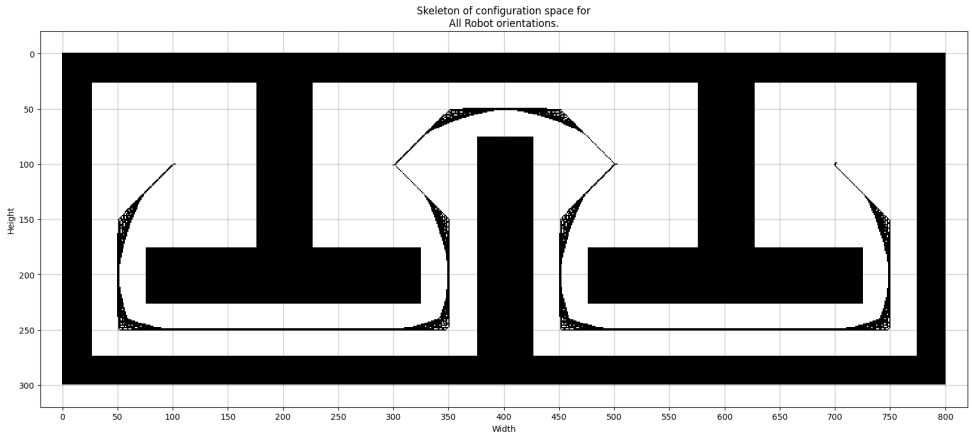


Figure 10: Skelethon for all robot orientations on top of eachother showing in orientation 0 °.

When I had the skeletons I used only the configuration 0°and found the shortest path from robot start to the skeleton, then found the shortest path from the goal to the skeleton and plotted the path. The path length was 1534 which is larger than the shortest path found with modified Djikstra and we can see in figure 11 that that path is the furthest from the walls as possible making it the safest.

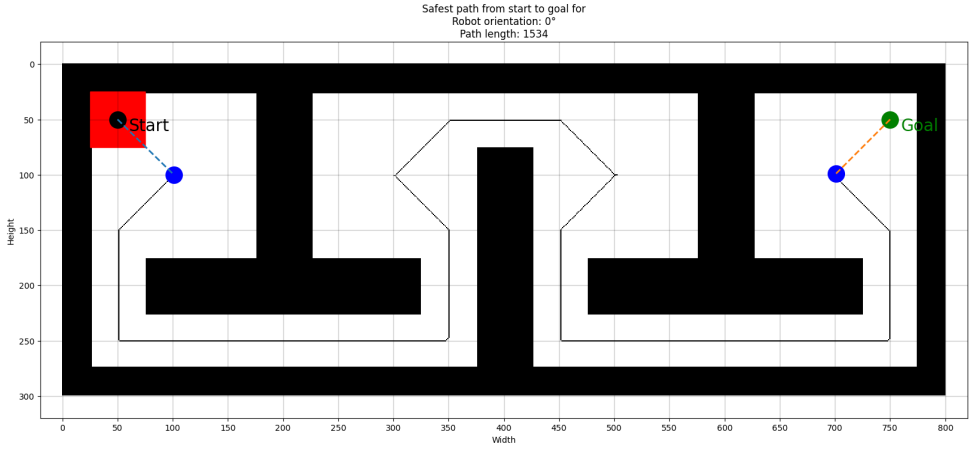


Figure 11: Safest path for Robot in orientation 0 °, path length = 1534

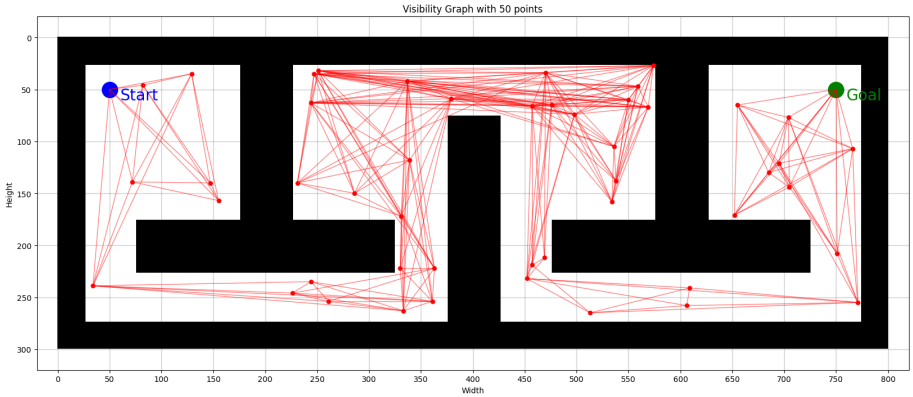
4 Problem 4

Use probabilistic roadmaps (PRM) to compute a path between startand end-points with 50, 100 and 500 sample points. What is the difference in path length? Illustrate each computed path.

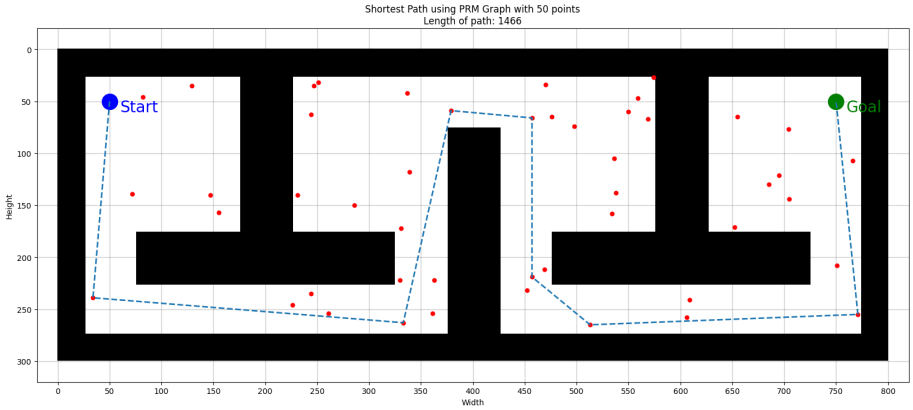
Answer: Probabilistic Roadmaps provide a method for pathfinding from the start to the goal. The algorithm is implemented as follows:

1. Generate N distinct points in the configuration space, ensuring they do not intersect with walls.
2. Establish edges between each point and all other points, ensuring the edge does not pass through a wall.
3. Utilize the 'networkx' library to construct a graph where the shortest path, based on distance, is determined. The weight on each edge is specified as the distance.

The algorithm was executed for N values of 50, 100, and 500, as indicated in the problem. The results are visualized in the figures below:

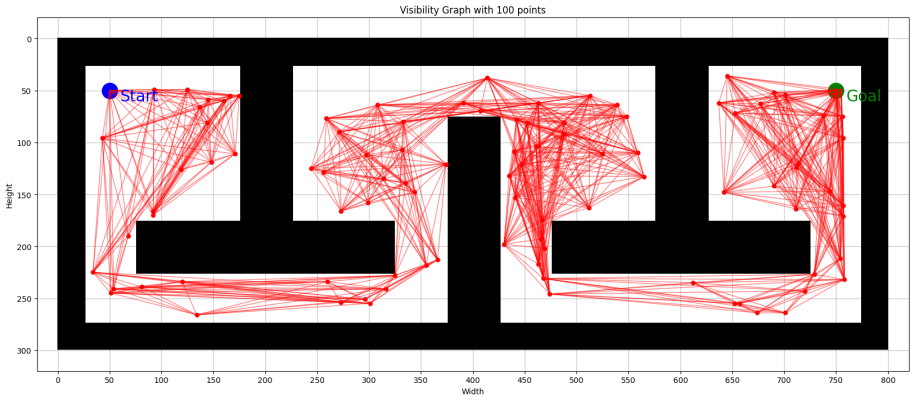


(a) PRM algorithm with 50 points.

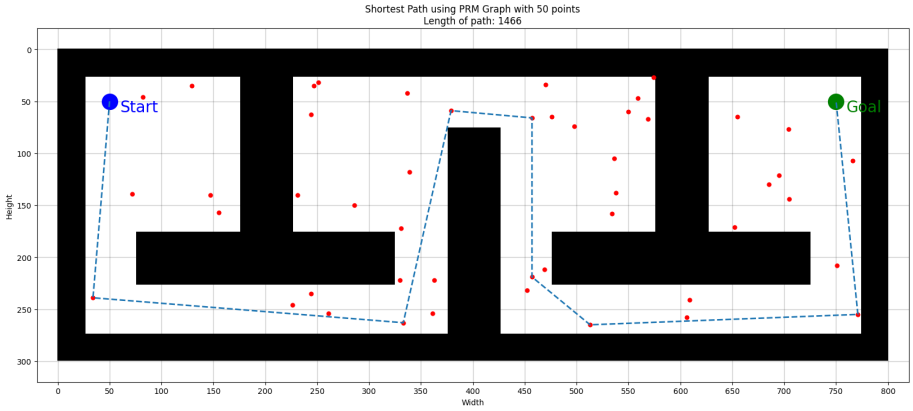


(b) PRM algorithm with 50 points and shortest path, path length = 1466

Figure 12: PRM Algorithm Results N=50



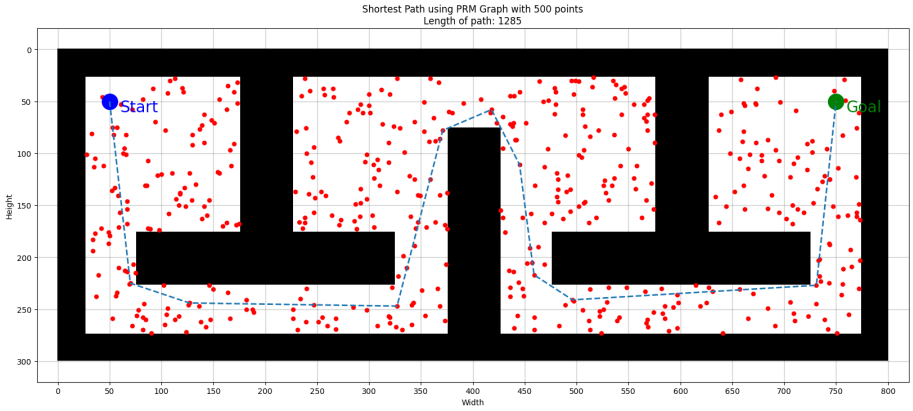
(a) PRM algorithm with 100 points.



(b) PRM algorithm with 100 points and shortest path, path length = 1305

Figure 13: PRM Algorithm Results N=100

(a) PRM algorithm with 500 points.



(b) PRM algorithm with 500 points and shortest path, path length = 1285

Figure 14: PRM Algorithm Results N=500

Method	Path Length
Mod. Djikstra Shortest Path	1132
Safest Path	1534
PRM N=50	1466
PRM N=100	1305
PRM N=500	1285

Table 1: Path Lengths for Different Methods

While the paths for N=50 occasionally failed to connect, N=100 experienced intermittent challenges, and N=500 consistently produced a path. The path lengths observed are intermediary between the shortest and safest paths. As N increases, a trend of decreasing path length is evident, indicating a finer-grained exploration of the grid space.

5 Problem 5

Do the same with Rapid exploring random trees (RRT). What are the main differences in performance between PRM and RRT? Illustrate each path.

Answer: The Rapid exploring random can be used to find a path from start to goal. I implemented the algorithm like this:

1. **Initialize Graph:**
   - Create an empty graph to represent the roadmap.
2. **Add Start Node:**
   - Include the start node in the graph.
3. **Generate Random Point:**
   - Generate a random point within the configuration space.
4. **Find Closest Node:**
   - Identify the closest existing node in the graph to the newly generated random point.
5. **Draw Connecting Line:**
   - Draw a line between the closest node and the random point.
6. **Extend Line to New Point:**
   - Extend the line from the closest node to a point at a distance of MAXDISTANCE in the direction of the random point.
7. **Check Path Clearance:**
   - If there is an unobstructed path from the closest node to the newly extended point, add the new point to the graph. Otherwise, explore the second closest node.
8. **Termination Condition:**
   - Repeat steps 3-7 until the newly added node is within a specified distance of the goal.

The algorithm was executed for N values of 50, 100, and 500, as indicated in the problem, that MAXDISTANCE was 50. The results are visualized in the figures below:

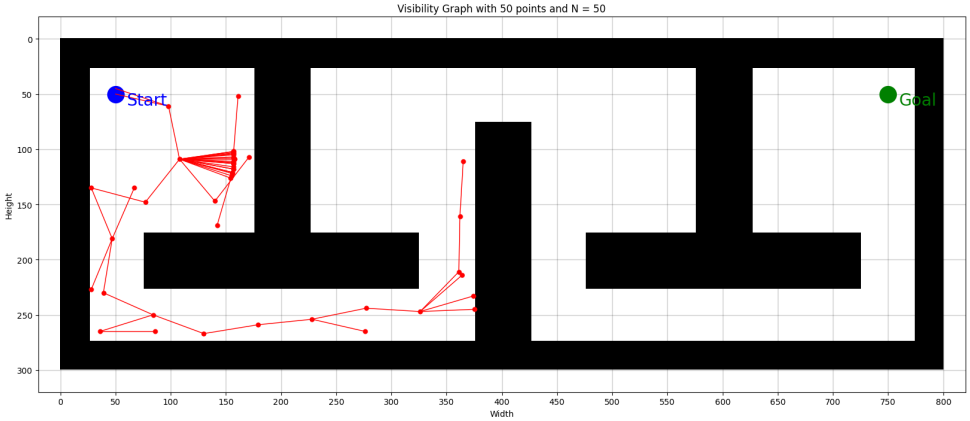


Figure 15: RRT for N = 50.

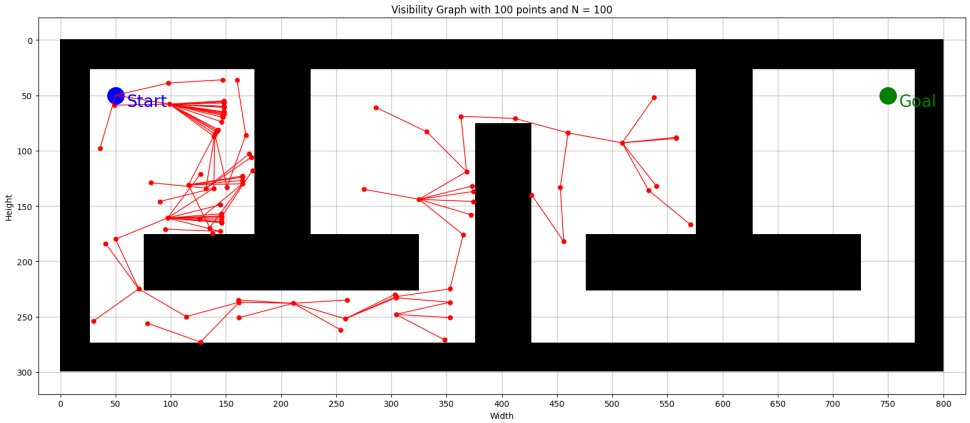
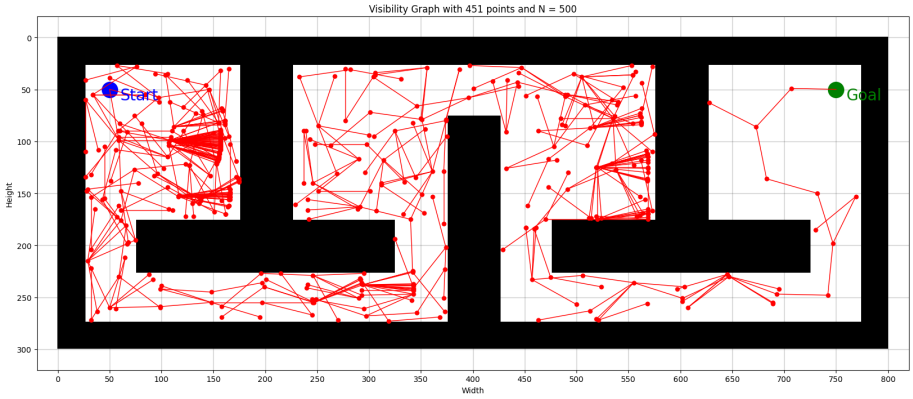
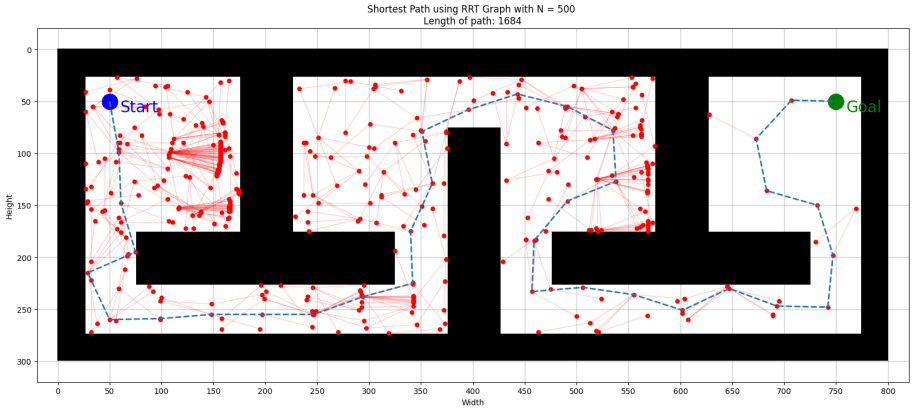


Figure 16: RRT for N = 100.



(a) PRM algorithm with N = 500.



(b) RRT algorithm with N = 500 and shortest path, path length = 1684

Figure 17: RRT Algorithm Results N=500

As we can see from figure 15 when N=50 the algorithm does not come to far. It stops and a lot of the nodes connect to only one.

From 16 when N=100 the algorithm comes further but still is not reaching the goal.

When N = 500 we have enough nodes and we reach the goal as seen in figure 17a. The shortest path has a length of 1684.

The difference in performance in RRT and PRM is that its more likely to find a path using PRM then when using RRT but RRT runs much faster 1.2 s compared to 28.4 s. The path from N = 500 RRT is 1684 much longer then for PRM.

Method	Path Length
Mod. Djikstra Shortest Path	1132
Safest Path	1534
PRM N=50	1466
PRM N=100	1305
PRM N=500	1285
RRT N=50	no solution
RRT N=100	no solution
RRT N=500	1684

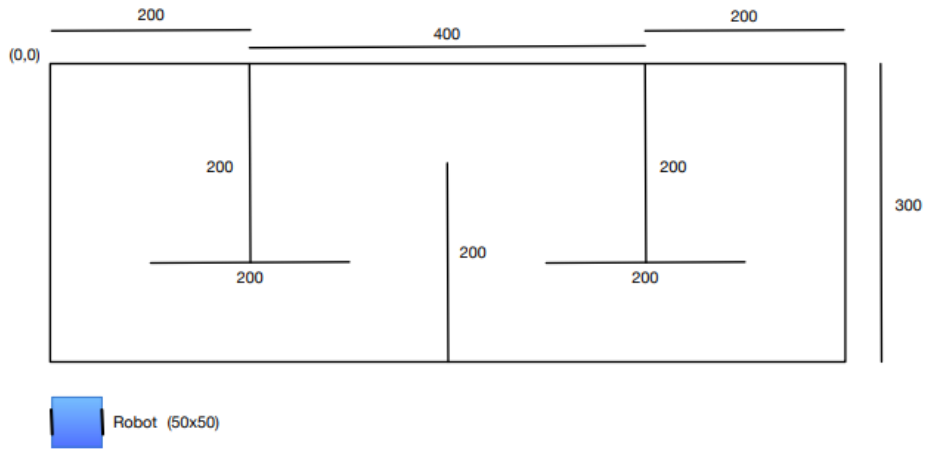
Table 2: Path Lengths for Different Methods

Name: Adrian Langmo Pavlak

PID: U09830551

Homework 5 - CSE 276 - Math for Robotics
Due: Sunday, 10 December 2023

The world model is shown in figure 1. The robot is a differential drive system with a square geometry of size 50x50.



In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.transforms import Affine2D
from scipy.signal import convolve2d
from scipy.ndimage import rotate
from skimage import morphology
from skimage.morphology import  skeletonize, skeletonize_3d
from skimage.util import invert
import heapq
```

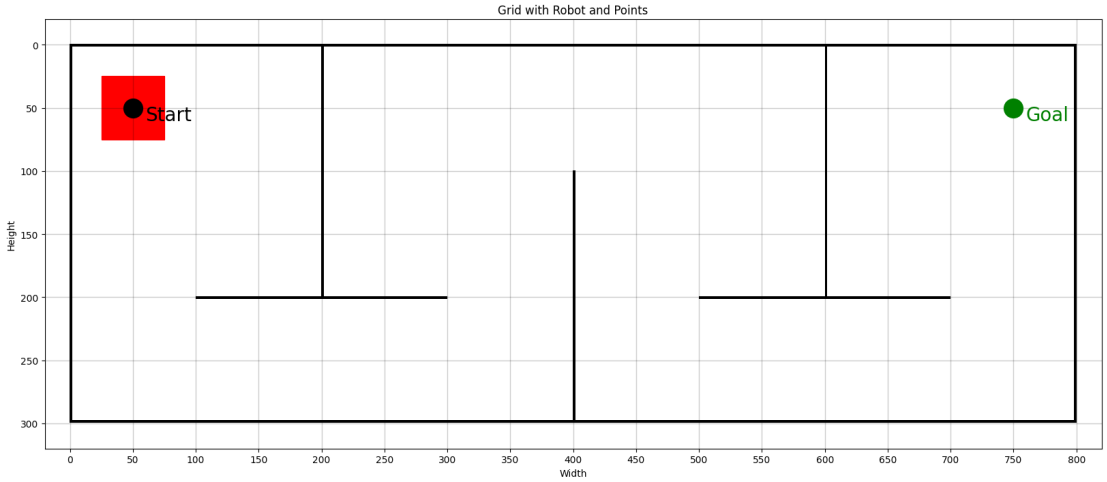
Problem 1

Generate the configuration space for the robot with a grid size of 2x2 and 5 deg in angular resolution. Generate an illustration of what the configuration space looks like with the robot at orientations 0, 45 and 90 deg.

In [ ]:
```
def make_blank_grid(height, width):
    grid = np.zeros((height, width), dtype=int)
    return grid

def add_wall(x0,y0,dir,length,size,grid):
    if dir == 'h':
        grid[y0:y0+size,x0:x0+length] = 1
    elif dir == 'v':
        grid[y0:y0+length,x0:x0+size] = 1
    return grid

def add_boarders(size, grid):
```

```
    grid[:size, :] = 1
    grid[-size:, :] = 1
    grid[:, :size] = 1
    grid[:, -size:] = 1
    return grid

def plot_grid(grid, title="Grid"):
    plt.figure(figsize=(20, 15))
    plt.imshow(grid, cmap='Greys',  interpolation='nearest')
    plt.xlabel("Width")
    plt.ylabel("Height")
    plt.title(title)
    pad = 20
    plt.xlim(-pad, grid.shape[1] + pad)
    plt.ylim(grid.shape[0] + pad, -pad)  # Invert y axis
    plt.grid(which='both', linestyle='-', linewidth=1, color='black', alpha=0.2)
    plt.xticks(np.arange(0, grid.shape[1]+1, 50))
    plt.yticks(np.arange(0, grid.shape[0]+1, 50))

def plot_robot(grid, robot_x0, robot_y0, robot_size, angle=0, title="Grid with R
    plot_grid(grid, title)
    x = int(robot_x0 - robot_size/2)
    y = int(robot_y0 - robot_size/2)
    w = robot_size
    h = robot_size
    rect = plt.Rectangle((x, y), w, h, angle=angle, color='r')
    plt.gca().add_patch(rect)

def plot_robot_and_points(grid, robot_x0, robot_y0, robot_size, px1, py1, px2, p
    plot_robot(grid, robot_x0, robot_y0, robot_size, angle, title)
    plt.plot(px1, py1, 'ko', markersize=20)
    plt.plot(px2, py2, 'go', markersize=20)

    # Add text with start and goal
    plt.text(px1 + 10, py1 + 10, 'Start', fontsize=20, color='k')
    plt.text(px2 + 10, py2 + 10, 'Goal', fontsize=20, color='g')

def print_grid(grid):
    for i in range(grid.shape[0]):
        for j in range(grid.shape[1]):
            print(grid[i,j],end='')
        print('')

# Parameters
square_size = 2
angle_resolution = 5
height = 300
width = 800
robot_size = 50
robot_x0 = 50
robot_y0 = 50
goal_x = 750
goal_y = 50

# Create grid
grid = make_blank_grid(height, width)

# Add boarders
grid = add_boarders(square_size,grid)
```

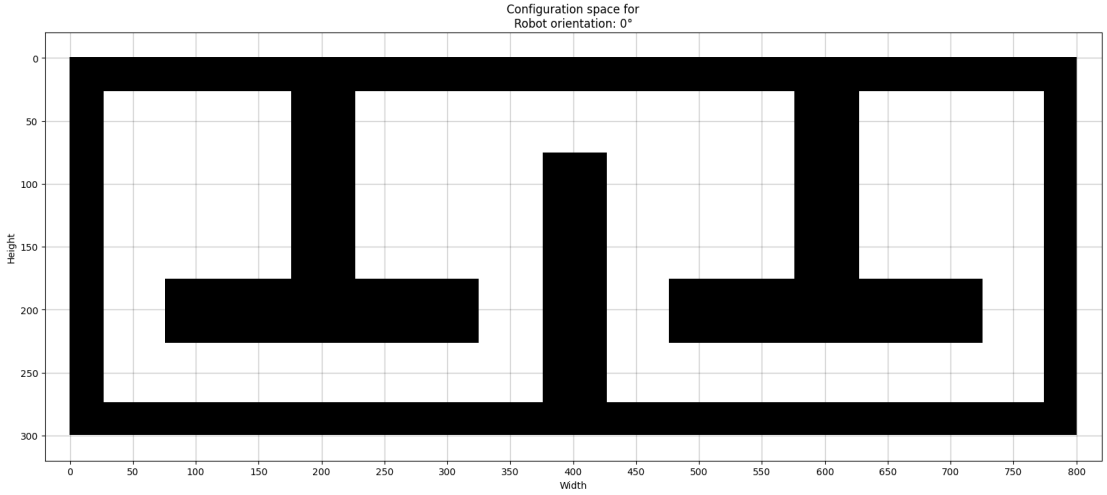
```
# Add walls
grid = add_wall(200,0,'v',200,square_size,grid)
grid = add_wall(600,0,'v',200,square_size,grid)
grid = add_wall(100,200,'h',200,square_size,grid)
grid = add_wall(500,200,'h',200,square_size,grid)
grid = add_wall(400, 100,'v',200,square_size,grid)

# Plot grid
original_grid = grid.copy()
plot_robot_and_points(grid,robot_x0,robot_y0,robot_size,robot_x0,robot_y0,goal_x
plt.show()
```

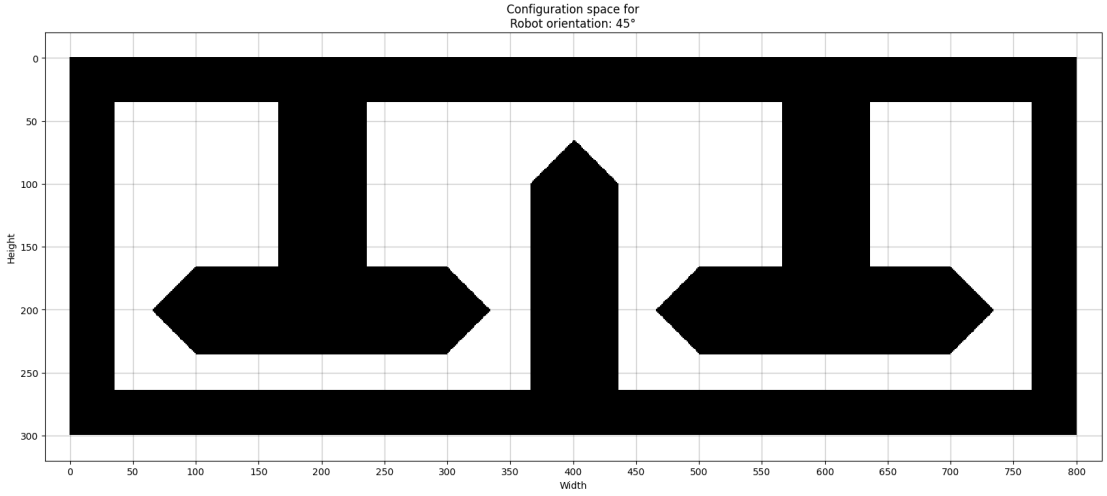


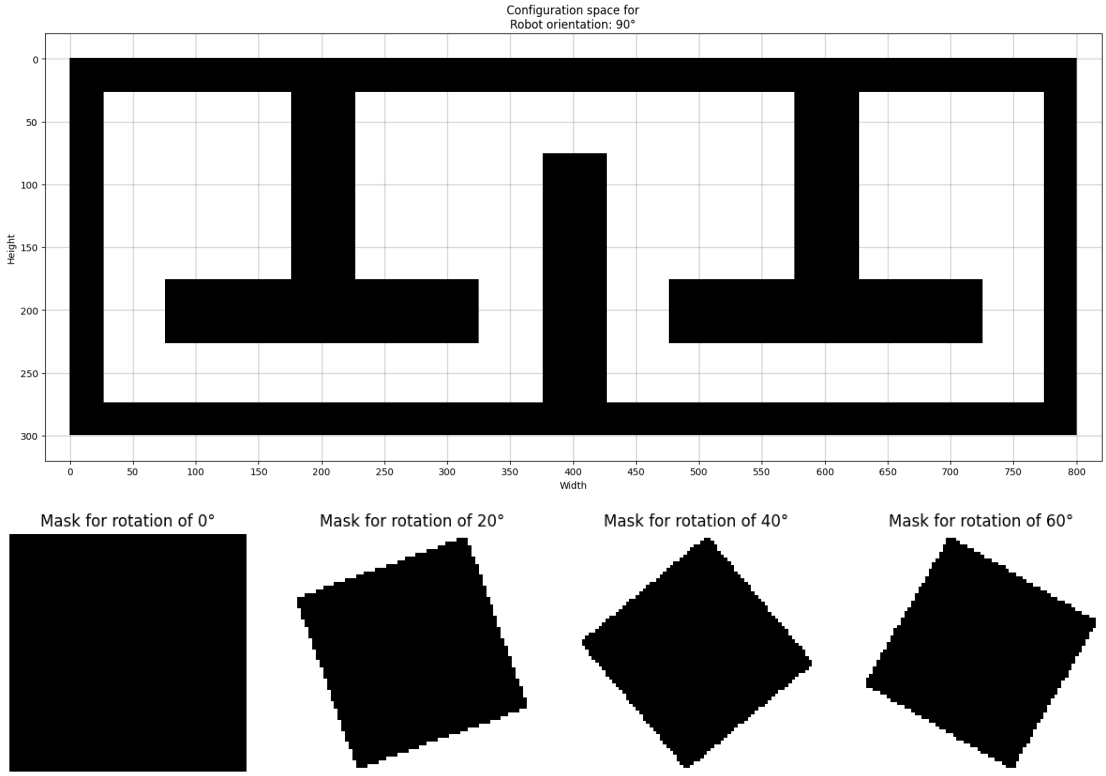
In [ ]:
```
# Make a configuration space for the robot in the grid
# The robot is a square with size robot_size
# The configuration space is a grid with the same size as the grid
# The configuration space is 1 where the robot can not be and 0 where it can be

# Create configuration space
def config_space_for_robot(robot_size, grid, robot_orientation):
    robot_mask = np.ones((robot_size, robot_size), dtype=int)
    robot_mask_rotated = rotate(robot_mask, robot_orientation)
    config_space = convolve2d(grid, robot_mask_rotated, mode='same', boundary='f

    # Set all non-zero values to 1
    config_space[config_space != 0] = 1
    return config_space

# Make configuration space for the robot given angle resolution
robot_orientations = np.arange(0, 180, angle_resolution)
config_spaces = np.zeros((height, width, len(robot_orientations)))

for i in range(len(robot_orientations)):
    orientation = robot_orientations[i]
    config_space = config_space_for_robot(robot_size, grid, orientation)
    config_spaces[:,:,i] = config_space
```

In [ ]:
```
# Plot configuration space for different orientations in the task
test_orientations = [0, 45, 90]
for orientation in test_orientations:
    index = np.where(robot_orientations == orientation)[0][0]
    config_space = config_spaces[:,:,index]
    plot_grid(config_space, "Configuration space for\nRobot orientation: {}°".fo
    plt.show()
```

```
# Plot masks for different orientations
fig, axes = plt.subplots(1, 4, figsize=(15, 5))
for i, ax in enumerate(axes):
    orientation = robot_orientations[(i)*4]

    # Create and plot the mask for the rotation
    robot_mask = np.ones((robot_size, robot_size), dtype=int)
    robot_mask_rotated = rotate(robot_mask, orientation)
    robot_mask_rotated[robot_mask_rotated != 0] = 1

    robot_mask_rotated[robot_mask_rotated != 0] = 1
    if orientation == 0:
        color = 'Greys_r'
    else:
        color = 'Greys'
    ax.imshow(robot_mask_rotated, cmap=color, interpolation='nearest')

    ax.set_title("Mask for rotation of {}°".format(orientation))
    ax.axis('off')  # Turn off axis for cleaner visualization

plt.show()
```







Problem 2

Use greedy search to find the shortest path between start-point (50,50) and end-point (750,50). Illustrate the path and provide its length.

In [ ]:
```python
def hit_obstacle_3D(x, y, z, config_spaces):
    return config_spaces[y, x, z] == 1

def hit_obstacle_2D(x, y, config_space):
    return config_space[y, x] == 1

graph = config_spaces
config_space_for_robot = config_spaces

start = (robot_x0, robot_y0, 0)
goal = (goal_x, goal_y, 0)

# Dijkstra's algorithm
def dijkstra_3D(graph, start, goal):
    rows, cols, depth = graph.shape
    distance = np.full((cols, rows, depth), fill_value=np.inf)
    distance[start] = 0
    parent = np.full((cols, rows, depth, 3), fill_value=-1, dtype=int)
    visited = np.zeros((cols, rows, depth), dtype=bool)

    priority_queue = [(0, start)]

    while priority_queue:
        current_cost, current_node = heapq.heappop(priority_queue)

        if visited[current_node]:
            continue
```

```
        visited[current_node] = True

        if current_node == goal:
            break

        x, y, z = current_node

        neighbors = [
            (x + 1, y, z),
            (x - 1, y, z),
            (x, y + 1, z),
            (x, y - 1, z),
            (x + 1, y + 1, z),
            (x + 1, y - 1, z),
            (x - 1, y + 1, z),
            (x - 1, y - 1, z),
            # Add depth neighbors
            (x, y, z + 1),
            (x, y, z - 1),
        ]

        for neighbor in neighbors:
            nx, ny, nz = neighbor
            if 0 <= nx < cols and 0 <= ny < rows and 0 <= nz < depth and not vis
                # If we are not moving in depth, add 1 to the distance
                dist_to_goal = np.sqrt((nx - goal[0])**2 + (ny - goal[1])**2)
                current_dist_to_goal = np.sqrt((x - goal[0])**2 + (y - goal[1])*
                diff_dist_to_goal = dist_to_goal - current_dist_to_goal

                if nz == z:
                    neighbor_distance = distance[x, y, z] + 2 + diff_dist_to_goa
                else:
                    neighbor_distance = distance[x, y, z] + 1 + diff_dist_to_goa

                if neighbor_distance < distance[nx, ny, nz]:
                    distance[nx, ny, nz] = neighbor_distance
                    parent[nx, ny, nz] = (x, y, z)
                    heapq.heappush(priority_queue, (neighbor_distance, neighbor)

    # Reconstruct the path
    path = [goal]
    current_node = goal
    while not np.array_equal(current_node, start):
        path.append(tuple(parent[current_node]))
        current_node = tuple(parent[current_node])

    path.reverse()
    return path

# Find the shortest path
path = dijkstra_3D(graph, start, goal)
print("Shortest Path:", path)
print("Length of path:", len(path))
```

```
Shortest Path: [(50, 50, 0), (50, 50, 1), (50, 50, 2), (50, 50, 3), (50, 50, 4),
(50, 50, 5), (50, 50, 6), (50, 50, 7), (50, 50, 8), (50, 50, 9), (51, 51, 9), (5
2, 52, 9), (53, 53, 9), (54, 54, 9), (55, 55, 9), (56, 56, 9), (57, 57, 9), (58,
58, 9), (59, 59, 9), (60, 60, 9), (61, 61, 9), (62, 62, 9), (63, 63, 9), (64, 64,
9), (65, 65, 9), (66, 66, 9), (67, 67, 9), (68, 68, 9), (69, 69, 9), (70, 70, 9),
(71, 71, 9), (72, 72, 9), (73, 73, 9), (74, 74, 9), (75, 75, 9), (76, 76, 9), (7
7, 77, 9), (78, 78, 9), (79, 79, 9), (80, 80, 9), (81, 81, 9), (82, 82, 9), (83,
83, 9), (84, 84, 9), (85, 85, 9), (86, 86, 9), (87, 87, 9), (88, 88, 9), (89, 89,
9), (90, 90, 9), (91, 91, 9), (92, 92, 9), (93, 93, 9), (94, 94, 9), (95, 95, 9),
(96, 96, 9), (97, 97, 9), (98, 98, 9), (99, 99, 9), (100, 100, 9), (101, 101, 9),
(102, 102, 9), (103, 103, 9), (104, 104, 9), (105, 105, 9), (106, 106, 9), (107,
107, 9), (108, 108, 9), (109, 109, 9), (110, 110, 9), (111, 111, 9), (112, 112,
9), (113, 113, 9), (114, 114, 9), (115, 115, 9), (116, 116, 9), (117, 117, 9), (1
18, 118, 9), (119, 119, 9), (120, 120, 9), (121, 121, 9), (122, 122, 9), (123, 12
3, 9), (124, 124, 9), (125, 125, 9), (126, 126, 9), (127, 127, 9), (128, 128, 9),
(129, 129, 9), (130, 130, 9), (131, 131, 9), (132, 132, 9), (132, 133, 9), (131,
134, 9), (130, 135, 9), (129, 136, 9), (128, 137, 9), (127, 138, 9), (126, 139,
9), (125, 140, 9), (124, 141, 9), (123, 142, 9), (122, 143, 9), (121, 144, 9), (1
20, 145, 9), (119, 146, 9), (118, 147, 9), (117, 148, 9), (116, 149, 9), (115, 15
0, 9), (114, 151, 9), (113, 152, 9), (112, 153, 9), (111, 154, 9), (110, 155, 9),
(109, 156, 9), (108, 157, 9), (107, 158, 9), (106, 159, 9), (105, 160, 9), (104,
161, 9), (103, 162, 9), (102, 163, 9), (101, 164, 9), (100, 165, 9), (99, 166,
9), (98, 167, 9), (97, 168, 9), (96, 169, 9), (95, 170, 9), (94, 171, 9), (93, 17
2, 9), (92, 173, 9), (91, 174, 9), (90, 175, 9), (89, 176, 9), (88, 177, 9), (87,
178, 9), (86, 179, 9), (85, 180, 9), (84, 181, 9), (83, 182, 9), (82, 183, 9), (8
1, 184, 9), (80, 185, 9), (79, 186, 9), (78, 187, 9), (77, 188, 9), (76, 189, 9),
(75, 190, 9), (74, 191, 9), (73, 192, 9), (72, 193, 9), (71, 194, 9), (70, 195,
9), (69, 196, 9), (68, 197, 9), (67, 198, 9), (66, 199, 9), (65, 200, 9), (65, 20
1, 9), (66, 202, 9), (67, 203, 9), (68, 204, 9), (69, 205, 9), (70, 206, 9), (71,
207, 9), (72, 208, 9), (73, 209, 9), (74, 210, 9), (75, 211, 9), (76, 212, 9), (7
7, 213, 9), (78, 214, 9), (79, 215, 9), (80, 216, 9), (81, 217, 9), (82, 218, 9),
(83, 219, 9), (84, 220, 9), (85, 221, 9), (86, 222, 9), (87, 223, 9), (88, 224,
9), (89, 225, 9), (90, 226, 9), (91, 227, 9), (92, 228, 9), (93, 229, 9), (94, 23
0, 9), (95, 231, 9), (96, 232, 9), (97, 233, 9), (98, 234, 9), (99, 235, 9), (10
0, 236, 9), (101, 236, 9), (102, 236, 9), (103, 236, 9), (104, 236, 9), (105, 23
6, 9), (106, 236, 9), (107, 236, 9), (108, 236, 9), (109, 236, 9), (110, 236, 9),
(111, 236, 9), (112, 236, 9), (113, 236, 9), (114, 236, 9), (115, 236, 9), (116,
236, 9), (117, 236, 9), (118, 236, 9), (119, 236, 9), (120, 236, 9), (121, 236,
9), (122, 236, 9), (123, 236, 9), (124, 236, 9), (125, 236, 9), (126, 236, 9), (1
27, 236, 9), (128, 236, 9), (129, 236, 9), (130, 236, 9), (131, 236, 9), (132, 23
6, 9), (133, 236, 9), (134, 236, 9), (135, 236, 9), (136, 236, 9), (137, 236, 9),
(138, 236, 9), (139, 236, 9), (140, 236, 9), (141, 236, 9), (142, 236, 9), (143,
236, 9), (144, 236, 9), (145, 236, 9), (146, 236, 9), (147, 236, 9), (148, 236,
9), (149, 236, 9), (150, 236, 9), (151, 236, 9), (152, 236, 9), (153, 236, 9), (1
54, 236, 9), (155, 236, 9), (156, 236, 9), (157, 236, 9), (158, 236, 9), (159, 23
6, 9), (160, 236, 9), (161, 236, 9), (162, 236, 9), (163, 236, 9), (164, 236, 9),
(165, 236, 9), (166, 236, 9), (167, 236, 9), (168, 236, 9), (169, 236, 9), (170,
236, 9), (171, 236, 9), (172, 236, 9), (173, 236, 9), (174, 236, 9), (175, 236,
9), (176, 236, 9), (177, 236, 9), (178, 236, 9), (179, 236, 9), (180, 236, 9), (1
81, 236, 9), (182, 236, 9), (183, 236, 9), (184, 236, 9), (185, 236, 9), (186, 23
6, 9), (187, 236, 9), (188, 236, 9), (189, 236, 9), (190, 236, 9), (191, 236, 9),
(192, 236, 9), (193, 236, 9), (194, 236, 9), (195, 236, 9), (196, 236, 9), (197,
236, 9), (198, 236, 9), (199, 236, 9), (200, 236, 9), (201, 236, 9), (202, 236,
9), (203, 236, 9), (204, 236, 9), (205, 236, 9), (206, 236, 9), (207, 236, 9), (2
08, 236, 9), (209, 236, 9), (210, 236, 9), (211, 236, 9), (212, 236, 9), (213, 23
6, 9), (214, 236, 9), (215, 236, 9), (216, 236, 9), (217, 236, 9), (218, 236, 9),
(219, 236, 9), (220, 236, 9), (221, 236, 9), (222, 236, 9), (223, 236, 9), (224,
236, 9), (225, 236, 9), (226, 236, 9), (227, 236, 9), (228, 236, 9), (229, 236,
9), (230, 236, 9), (231, 236, 9), (232, 236, 9), (233, 236, 9), (234, 236, 9), (2
35, 236, 9), (236, 236, 9), (237, 236, 9), (238, 236, 9), (239, 236, 9), (240, 23
```

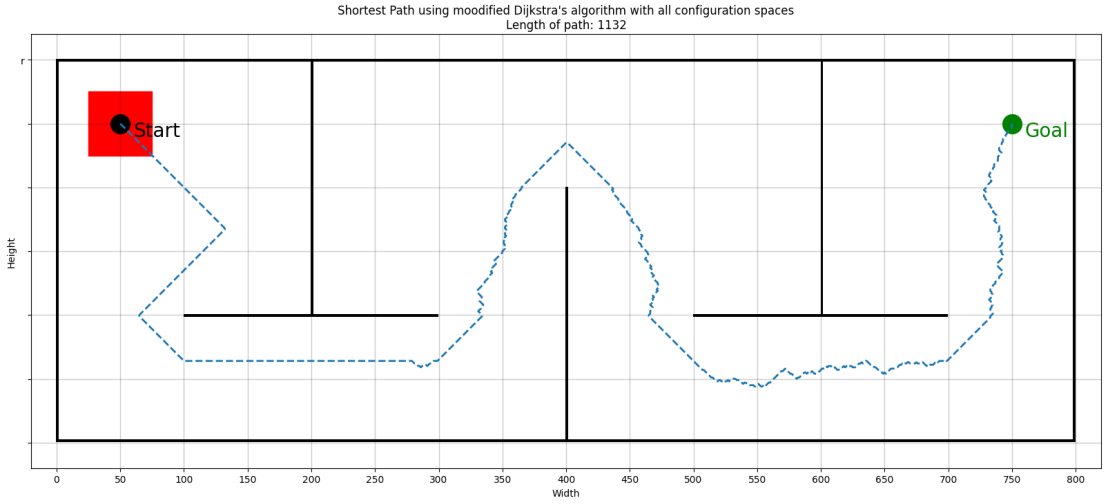
6, 9), (241, 236, 9), (242, 236, 9), (243, 236, 9), (244, 236, 9), (245, 236, 9), (246, 236, 9), (247, 236, 9), (248, 236, 9), (249, 236, 9), (250, 236, 9), (251, 236, 9), (252, 236, 9), (253, 236, 9), (254, 236, 9), (255, 236, 9), (256, 236, 9), (257, 236, 9), (258, 236, 9), (259, 236, 9), (260, 236, 9), (261, 236, 9), (262, 236, 9), (263, 236, 9), (264, 236, 9), (265, 236, 9), (266, 236, 9), (267, 236, 9), (268, 236, 9), (269, 236, 9), (270, 236, 9), (271, 236, 9), (272, 236, 9), (273, 236, 9), (274, 236, 9), (275, 236, 9), (276, 236, 9), (277, 236, 9), (278, 236, 9), (279, 236, 9), (280, 237, 9), (281, 238, 9), (282, 238, 9), (283, 239, 9), (284, 240, 9), (285, 240, 9), (286, 241, 9), (287, 240, 9), (288, 239, 9), (289, 239, 9), (290, 239, 9), (291, 240, 9), (292, 239, 9), (293, 238, 9), (294, 238, 9), (295, 237, 9), (296, 236, 9), (297, 236, 9), (298, 236, 9), (299, 236, 9), (300, 235, 9), (301, 234, 9), (302, 233, 9), (303, 232, 9), (304, 231, 9), (305, 230, 9), (306, 229, 9), (307, 228, 9), (308, 227, 9), (309, 226, 9), (310, 225, 9), (311, 224, 9), (312, 223, 9), (313, 222, 9), (314, 221, 9), (315, 220, 9), (316, 219, 9), (317, 218, 9), (318, 217, 9), (319, 216, 9), (320, 215, 9), (321, 214, 9), (322, 213, 9), (323, 212, 9), (324, 211, 9), (325, 210, 9), (326, 209, 9), (327, 208, 9), (328, 207, 9), (329, 206, 9), (330, 205, 9), (331, 204, 9), (332, 203, 9), (333, 202, 9), (334, 201, 9), (334, 200, 9), (333, 199, 9), (332, 198, 9), (331, 197, 9), (332, 196, 9), (333, 195, 9), (333, 194, 9), (334, 193, 9), (335, 192, 9), (334, 191, 9), (333, 190, 9), (332, 189, 9), (333, 188, 9), (333, 187, 9), (334, 186, 9), (333, 185, 9), (332, 184, 9), (331, 183, 9), (330, 182, 9), (330, 181, 9), (331, 180, 9), (332, 179, 9), (333, 178, 9), (334, 177, 9), (335, 176, 9), (336, 175, 9), (336, 174, 9), (337, 173, 9), (338, 172, 9), (339, 171, 9), (340, 170, 9), (340, 169, 9), (341, 168, 9), (342, 167, 9), (341, 166, 9), (342, 165, 9), (343, 164, 9), (342, 163, 9), (343, 162, 9), (344, 161, 9), (345, 160, 9), (344, 159, 9), (343, 158, 9), (344, 157, 9), (345, 156, 9), (346, 155, 9), (347, 154, 9), (348, 153, 9), (349, 152, 9), (350, 151, 9), (350, 150, 9), (351, 149, 9), (352, 148, 9), (352, 147, 9), (351, 146, 9), (352, 145, 9), (351, 144, 9), (351, 143, 9), (351, 142, 9), (352, 141, 9), (351, 140, 9), (352, 139, 9), (353, 138, 9), (352, 137, 9), (352, 136, 9), (353, 135, 9), (352, 134, 9), (353, 133, 9), (352, 132, 9), (352, 131, 9), (353, 130, 9), (352, 129, 9), (352, 128, 9), (353, 127, 9), (352, 126, 9), (352, 125, 9), (353, 124, 9), (354, 123, 9), (355, 122, 9), (354, 121, 9), (355, 120, 9), (355, 119, 9), (356, 118, 9), (357, 117, 9), (358, 116, 9), (357, 115, 9), (357, 114, 9), (357, 113, 9), (358, 112, 9), (358, 111, 9), (357, 110, 9), (358, 109, 9), (359, 108, 9), (360, 107, 9), (360, 106, 9), (361, 105, 9), (362, 104, 9), (363, 103, 9), (364, 102, 9), (365, 101, 9), (365, 100, 9), (366, 99, 9), (367, 98, 9), (368, 97, 9), (369, 96, 9), (370, 95, 9), (371, 94, 9), (372, 93, 9), (373, 92, 9), (374, 91, 9), (375, 90, 9), (376, 89, 9), (377, 88, 9), (378, 87, 9), (379, 86, 9), (380, 85, 9), (381, 84, 9), (382, 83, 9), (383, 82, 9), (384, 81, 9), (385, 80, 9), (386, 79, 9), (387, 78, 9), (388, 77, 9), (389, 76, 9), (390, 75, 9), (391, 74, 9), (392, 73, 9), (393, 72, 9), (394, 71, 9), (395, 70, 9), (396, 69, 9), (397, 68, 9), (398, 67, 9), (399, 66, 9), (400, 65, 9), (401, 65, 9), (402, 66, 9), (403, 67, 9), (404, 68, 9), (405, 69, 9), (406, 70, 9), (407, 71, 9), (408, 72, 9), (409, 73, 9), (410, 74, 9), (411, 75, 9), (412, 76, 9), (413, 77, 9), (414, 78, 9), (415, 79, 9), (416, 80, 9), (417, 81, 9), (418, 82, 9), (419, 83, 9), (420, 84, 9), (421, 85, 9), (422, 86, 9), (423, 87, 9), (424, 88, 9), (425, 89, 9), (426, 90, 9), (427, 91, 9), (428, 92, 9), (429, 93, 9), (430, 94, 9), (431, 95, 9), (432, 96, 9), (433, 97, 9), (434, 98, 9), (435, 99, 9), (436, 100, 9), (437, 101, 9), (436, 102, 9), (437, 103, 9), (437, 104, 9), (438, 105, 9), (438, 106, 9), (439, 107, 9), (440, 108, 9), (441, 109, 9), (441, 110, 9), (441, 111, 9), (442, 112, 9), (443, 113, 9), (444, 114, 9), (445, 115, 9), (446, 116, 9), (447, 117, 9), (447, 118, 9), (448, 119, 9), (448, 120, 9), (449, 121, 9), (450, 122, 9), (450, 123, 9), (451, 124, 9), (452, 125, 9), (451, 126, 9), (452, 127, 9), (453, 128, 9), (454, 129, 9), (455, 130, 9), (456, 131, 9), (457, 132, 9), (457, 133, 9), (458, 134, 9), (459, 135, 9), (459, 136, 9), (459, 137, 9), (460, 138, 9), (459, 139, 9), (459, 140, 9), (459, 141, 9), (458, 142, 9), (457, 143, 9), (458, 144, 9), (459, 145, 9), (459, 146, 9), (460, 147, 9), (461, 148, 9), (462, 149, 9), (463, 150, 9), (464, 151, 9), (464, 152, 9), (465, 153, 9), (464, 154, 9), (463, 155, 9), (462, 156, 9), (463, 157, 9), (464, 158, 9), (464, 159, 9), (465, 160, 9), (466, 161, 9), (467, 162, 9),

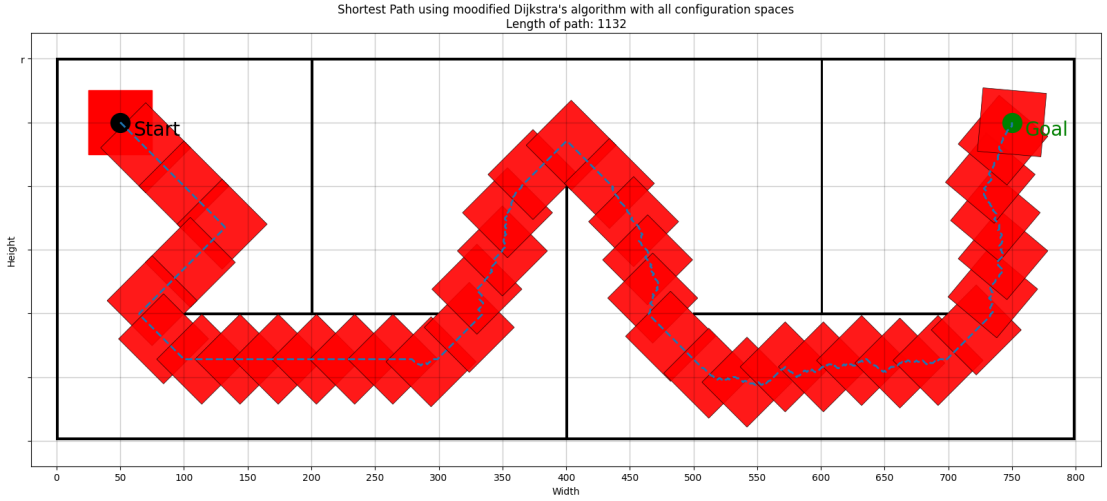
(466, 163, 9), (467, 164, 9), (468, 165, 9), (467, 166, 9), (467, 167, 9), (468, 168, 9), (469, 169, 9), (468, 170, 9), (468, 171, 9), (469, 172, 9), (470, 173, 9), (471, 174, 9), (472, 175, 9), (472, 176, 9), (473, 177, 9), (473, 178, 9), (472, 179, 9), (471, 180, 9), (472, 181, 9), (471, 182, 9), (470, 183, 9), (469, 184, 9), (468, 185, 9), (469, 186, 9), (468, 187, 9), (468, 188, 9), (469, 189, 9), (468, 190, 9), (467, 191, 9), (466, 192, 9), (467, 193, 9), (468, 194, 9), (468, 195, 9), (467, 196, 9), (466, 197, 9), (466, 198, 9), (466, 199, 9), (465, 200, 9), (465, 201, 9), (466, 202, 9), (467, 203, 9), (468, 204, 9), (469, 205, 9), (470, 206, 9), (471, 207, 9), (472, 208, 9), (473, 209, 9), (474, 210, 9), (475, 211, 9), (476, 212, 9), (477, 213, 9), (478, 214, 9), (479, 215, 9), (480, 216, 9), (481, 217, 9), (482, 218, 9), (483, 219, 9), (484, 220, 9), (485, 221, 9), (486, 222, 9), (487, 223, 9), (488, 224, 9), (489, 225, 9), (490, 226, 9), (491, 227, 9), (492, 228, 9), (493, 229, 9), (494, 230, 9), (495, 231, 9), (496, 232, 9), (497, 233, 9), (498, 234, 9), (499, 235, 9), (500, 236, 9), (501, 237, 9), (502, 238, 9), (503, 239, 9), (504, 240, 9), (505, 241, 9), (506, 242, 9), (507, 243, 9), (508, 243, 9), (509, 244, 9), (510, 245, 9), (511, 246, 9), (512, 247, 9), (513, 248, 9), (514, 249, 9), (515, 249, 9), (516, 250, 9), (517, 251, 9), (518, 252, 9), (519, 252, 9), (520, 251, 9), (521, 251, 9), (522, 252, 9), (523, 252, 9), (524, 251, 9), (525, 252, 9), (526, 252, 9), (527, 251, 9), (528, 252, 9), (529, 251, 9), (530, 251, 9), (531, 250, 9), (532, 250, 9), (533, 250, 9), (534, 251, 9), (535, 251, 9), (536, 251, 9), (537, 252, 9), (538, 253, 9), (539, 254, 9), (540, 253, 9), (541, 253, 9), (542, 254, 9), (543, 254, 9), (544, 255, 9), (545, 255, 9), (546, 255, 9), (547, 255, 9), (548, 256, 9), (549, 256, 9), (550, 255, 9), (551, 254, 9), (552, 255, 9), (553, 256, 9), (554, 256, 9), (555, 255, 9), (556, 254, 9), (557, 255, 9), (558, 254, 9), (559, 253, 9), (560, 253, 9), (561, 252, 9), (562, 251, 9), (563, 250, 9), (564, 249, 9), (565, 248, 9), (566, 247, 9), (567, 246, 9), (568, 245, 9), (569, 244, 9), (570, 244, 9), (571, 243, 9), (572, 242, 9), (573, 243, 9), (574, 244, 9), (575, 245, 9), (576, 246, 9), (577, 246, 9), (578, 247, 9), (579, 248, 9), (580, 249, 9), (581, 250, 9), (582, 249, 9), (583, 249, 9), (584, 248, 9), (585, 247, 9), (586, 246, 9), (587, 245, 9), (588, 245, 9), (589, 246, 9), (590, 245, 9), (591, 244, 9), (592, 245, 9), (593, 244, 9), (594, 245, 9), (595, 246, 9), (596, 246, 9), (597, 246, 9), (598, 245, 9), (599, 244, 9), (600, 243, 9), (601, 242, 9), (602, 242, 9), (603, 241, 9), (604, 240, 9), (605, 241, 9), (606, 240, 9), (607, 240, 9), (608, 240, 9), (609, 241, 9), (610, 242, 9), (611, 241, 9), (612, 242, 9), (613, 243, 9), (614, 243, 9), (615, 242, 9), (616, 241, 9), (617, 241, 9), (618, 240, 9), (619, 241, 9), (620, 240, 9), (621, 241, 9), (622, 240, 9), (623, 240, 9), (624, 240, 9), (625, 241, 9), (626, 241, 9), (627, 241, 9), (628, 240, 9), (629, 239, 9), (630, 238, 9), (631, 238, 9), (632, 237, 9), (633, 238, 9), (634, 237, 9), (635, 236, 9), (636, 236, 9), (637, 237, 9), (638, 238, 9), (639, 239, 9), (640, 239, 9), (641, 240, 9), (642, 241, 9), (643, 241, 9), (644, 242, 9), (645, 243, 9), (646, 244, 9), (647, 245, 9), (648, 245, 9), (649, 246, 9), (650, 245, 9), (651, 246, 9), (652, 245, 9), (653, 244, 9), (654, 243, 9), (655, 242, 9), (656, 242, 9), (657, 242, 9), (658, 242, 9), (659, 241, 9), (660, 241, 9), (661, 240, 9), (662, 239, 9), (663, 239, 9), (664, 238, 9), (665, 239, 9), (666, 238, 9), (667, 238, 9), (668, 237, 9), (669, 237, 9), (670, 238, 9), (671, 238, 9), (672, 238, 9), (673, 238, 9), (674, 238, 9), (675, 237, 9), (676, 238, 9), (677, 238, 9), (678, 238, 9), (679, 238, 9), (680, 237, 9), (681, 238, 9), (682, 239, 9), (683, 240, 9), (684, 239, 9), (685, 238, 9), (686, 239, 9), (687, 239, 9), (688, 239, 9), (689, 238, 9), (690, 237, 9), (691, 236, 9), (692, 237, 9), (693, 236, 9), (694, 236, 9), (695, 236, 9), (696, 236, 9), (697, 237, 9), (698, 236, 9), (699, 236, 9), (700, 235, 9), (701, 234, 9), (702, 233, 9), (703, 232, 9), (704, 231, 9), (705, 230, 9), (706, 229, 9), (707, 228, 9), (708, 227, 9), (709, 226, 9), (710, 225, 9), (711, 224, 9), (712, 223, 9), (713, 222, 9), (714, 221, 9), (715, 220, 9), (716, 219, 9), (717, 218, 9), (718, 217, 9), (719, 216, 9), (720, 215, 9), (721, 214, 9), (722, 213, 9), (723, 212, 9), (724, 211, 9), (725, 210, 9), (726, 209, 9), (727, 208, 9), (728, 207, 9), (729, 206, 9), (730, 205, 9), (731, 204, 9), (732, 203, 9), (733, 202, 9), (733, 202, 8), (733, 201, 8), (734, 200, 8), (735, 199, 8), (735, 198, 8), (734, 197, 8), (733, 196, 8), (733, 195, 8), (734, 194, 8), (734, 193, 8), (735, 192, 8), (735, 191, 8), (734, 190, 8), (733, 189, 8), (732, 188, 8), (733, 187, 8), (734, 186, 8),

```
(735, 185, 8), (735, 184, 8), (734, 183, 8), (734, 182, 8), (733, 181, 8), (733,
180, 8), (733, 179, 8), (734, 178, 8), (735, 177, 8), (736, 176, 8), (737, 175,
8), (737, 174, 8), (738, 173, 8), (739, 172, 8), (740, 171, 8), (740, 170, 8), (7
41, 169, 8), (742, 168, 8), (742, 167, 8), (743, 166, 8), (742, 165, 8), (741, 16
4, 8), (740, 163, 8), (739, 162, 8), (739, 161, 8), (738, 160, 8), (739, 159, 8),
(740, 158, 8), (741, 157, 8), (742, 156, 8), (743, 155, 8), (743, 154, 8), (743,
153, 8), (742, 152, 8), (742, 151, 8), (741, 150, 8), (742, 149, 8), (741, 148,
8), (740, 147, 8), (741, 146, 8), (742, 145, 8), (742, 144, 8), (743, 143, 8), (7
42, 142, 8), (741, 141, 8), (741, 140, 8), (742, 139, 8), (742, 138, 8), (741, 13
7, 8), (740, 136, 8), (739, 135, 8), (740, 134, 8), (740, 133, 8), (740, 132, 8),
(740, 131, 8), (740, 130, 8), (739, 129, 8), (738, 128, 8), (737, 127, 8), (736,
126, 8), (737, 125, 8), (737, 124, 8), (738, 123, 8), (738, 122, 8), (737, 121,
8), (736, 120, 8), (735, 119, 8), (736, 118, 8), (735, 117, 8), (734, 116, 8), (7
33, 115, 8), (733, 114, 8), (732, 113, 8), (731, 112, 8), (731, 111, 8), (730, 11
0, 8), (729, 109, 8), (729, 108, 8), (728, 107, 8), (728, 106, 8), (729, 105, 8),
(730, 104, 8), (729, 103, 8), (728, 102, 8), (729, 101, 8), (730, 100, 8), (731,
99, 8), (731, 98, 8), (732, 97, 8), (733, 96, 8), (734, 95, 8), (733, 94, 8), (73
2, 93, 8), (733, 92, 8), (732, 91, 8), (732, 90, 8), (733, 89, 8), (733, 88, 8),
(734, 87, 8), (735, 86, 8), (735, 85, 8), (736, 84, 8), (736, 83, 8), (736, 82,
8), (737, 81, 8), (738, 80, 8), (739, 79, 8), (739, 78, 8), (739, 77, 8), (739, 7
6, 8), (740, 75, 8), (741, 74, 8), (742, 73, 8), (741, 72, 8), (740, 71, 8), (74
0, 70, 8), (740, 69, 8), (741, 68, 8), (741, 67, 8), (742, 66, 8), (742, 65, 8),
(743, 64, 8), (743, 63, 8), (743, 62, 8), (744, 61, 8), (745, 60, 8), (745, 59,
8), (746, 58, 8), (746, 57, 8), (747, 56, 8), (748, 55, 8), (749, 54, 8), (750, 5
3, 8), (750, 52, 8), (750, 51, 8), (750, 50, 8), (750, 50, 7), (750, 50, 6), (75
0, 50, 5), (750, 50, 4), (750, 50, 3), (750, 50, 2), (750, 50, 1), (750, 50, 0)]
Length of path: 1132
```

In [ ]:

```
plot_robot_and_points(original_grid, robot_x0, robot_y0, robot_size, robot_x0, r
plt.title("Shortest Path using moodified Dijkstra's algorithm with all configura
plt.plot([x for x, y, z in path], [y for x, y, z in path],'--', 'r', linewidth=2
plt.show()

from matplotlib.patches import Rectangle

def plot_path_orientation(path, grid, title="Shortest Path using moodified Dijks
    plot_robot_and_points(original_grid, robot_x0, robot_y0, robot_size, robot_x
    i = 0
    for x,y,z in path:
        i += 1
        x = int(x - robot_size/2)
        y = int(y - robot_size/2)
        angle = robot_orientations[z]
        w = robot_size
        h = robot_size
        rect = plt.Rectangle((x, y), w, h, angle=0, facecolor='r', alpha=0.9, li
        rect.set_transform(Affine2D().rotate_deg_around(x + w/2, y + h/2, angle)

        if (i % 30 == 0) or (i == len(path)-1):
            plt.gca().add_patch(rect)

    plt.plot([x for x, y, z in path], [y for x, y, z in path],'--', 'r', linewid
    plt.show()

plot_path_orientation(path, original_grid, "Shortest Path using moodified Dijkst
```

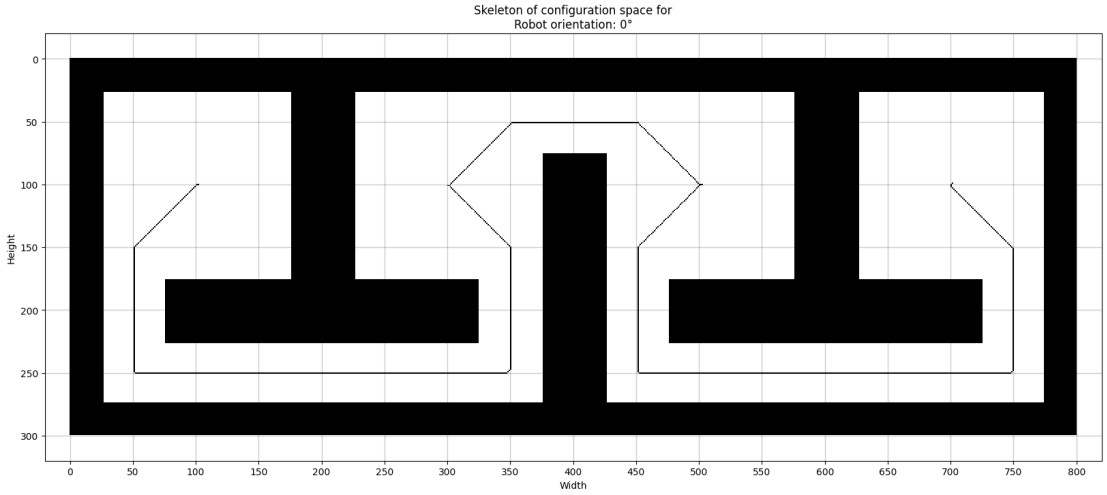




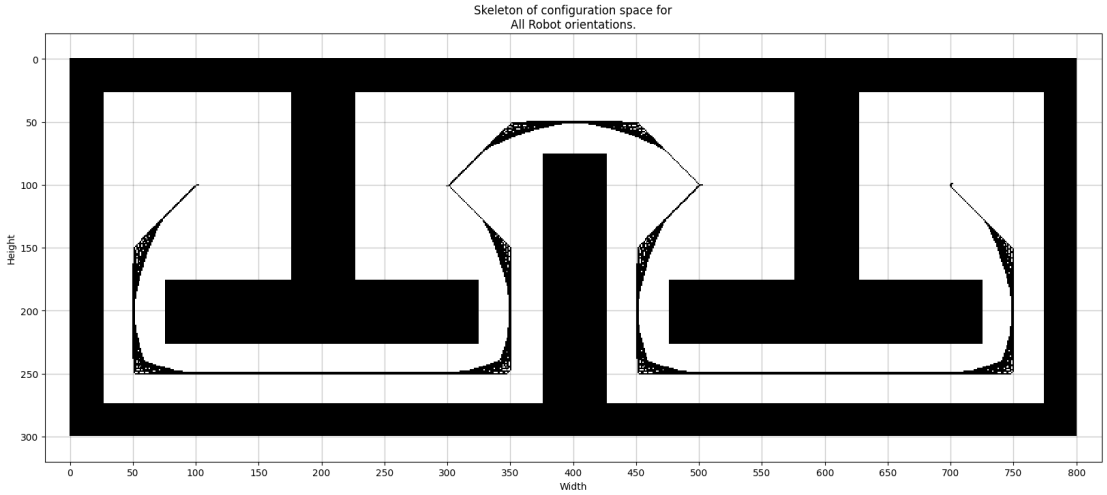
Problem 3

Compute the safest path from start to finish (hint: medial axis transform/Voronoi). Illustrate the path and provide its length.

In [ ]:
```
# Using maximum clearance with skeletonization
graph = config_spaces[:,:,0]
graph_inverted = invert(graph)
skeleton = skeletonize(graph_inverted)
graph_with_skeleton = graph + skeleton

# Plot the configuration space
plot_grid(graph_with_skeleton, "Skeleton of configuration space for\nRobot orien
```



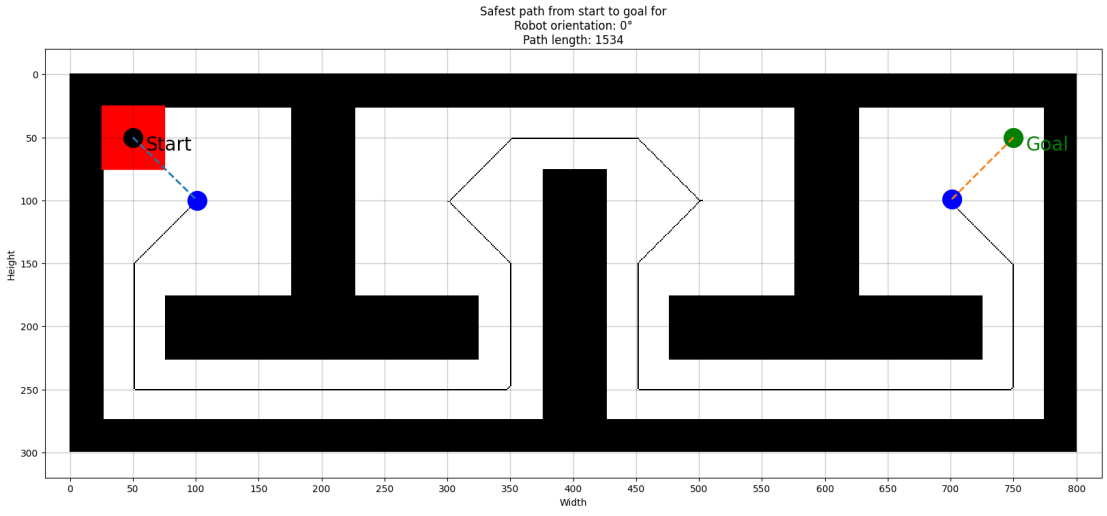
In [ ]:
```
height_skel, width_skel, num_orientations = config_spaces.shape
all_skeletons = np.zeros((height_skel, width_skel), dtype=int)
skeleton3d = np.zeros((height, width, num_orientations), dtype=int)

for conf in range(num_orientations):
    graph = config_spaces[:, :, conf]
    graph_inverted = invert(graph)
    skeleton_orientation = morphology.skeletonize(graph_inverted)

    # Add the skeleton to the overall skeleton
    all_skeletons += skeleton_orientation
    skeleton3d[:,:,conf] = skeleton_orientation

# Make skeleton eiter 0 or 1
all_skeletons[all_skeletons > 0] = 1

graph_with_all_skeleton = config_spaces[:, :, 0] + all_skeletons

plot_grid(graph_with_all_skeleton, "Skeleton of configuration space for\nAll Rob
```



In [ ]:
```
# Find the shortest path
def path_to_skeleton(x, y, skeleton):
    # Find the closest skeleton point
    skeleton_points = np.argwhere(skeleton == 1)

    distances = np.zeros(len(skeleton_points))
    for i in range(len(skeleton_points)):
```

```
        skeleton_point = skeleton_points[i]
        distances[i] = np.sqrt((x - skeleton_point[1])**2 + (y - skeleton_point[

    closest_skeleton_point = skeleton_points[np.argmin(distances)]
    return closest_skeleton_point

robot_x0 = 50
robot_y0 = 50
goal_x = 750
goal_y = 50

start_skeleton = path_to_skeleton(robot_x0, robot_y0, skeleton)
goal_skeleton = path_to_skeleton(goal_x, goal_y, skeleton)

# Plot the configuration space with the start and goal
plot_robot_and_points(graph_with_skeleton, robot_x0, robot_y0, robot_size, robot
plt.plot(start_skeleton[1], start_skeleton[0], 'bo', markersize=20)
plt.plot(goal_skeleton[1], goal_skeleton[0], 'bo', markersize=20)

# Add line betweenn start and start skeleton
plt.plot([robot_x0, start_skeleton[1]], [robot_y0, start_skeleton[0]],'--', line
plt.plot([goal_x, goal_skeleton[1]], [goal_y, goal_skeleton[0]],'--', linewidth=

# Path length
path_length = np.sqrt((robot_x0 - start_skeleton[1])**2 + (robot_y0 - start_skel
path_length += np.sum(skeleton==1)
path_length = int(path_length)

print("Path length:", path_length)

plt.title("Safest path from start to goal for\nRobot orientation: {}°\nPath leng

plt.show()
```

```
Path length: 1534
```

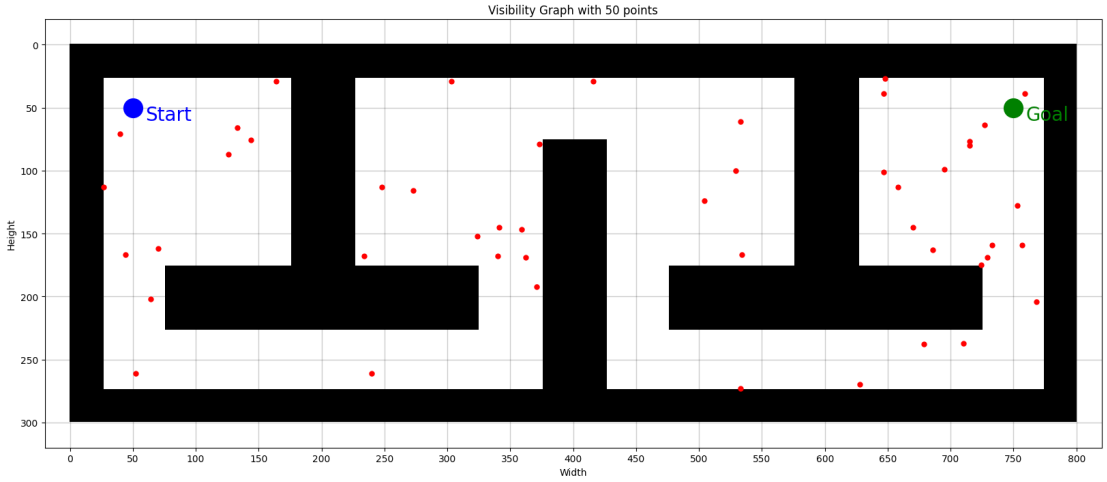


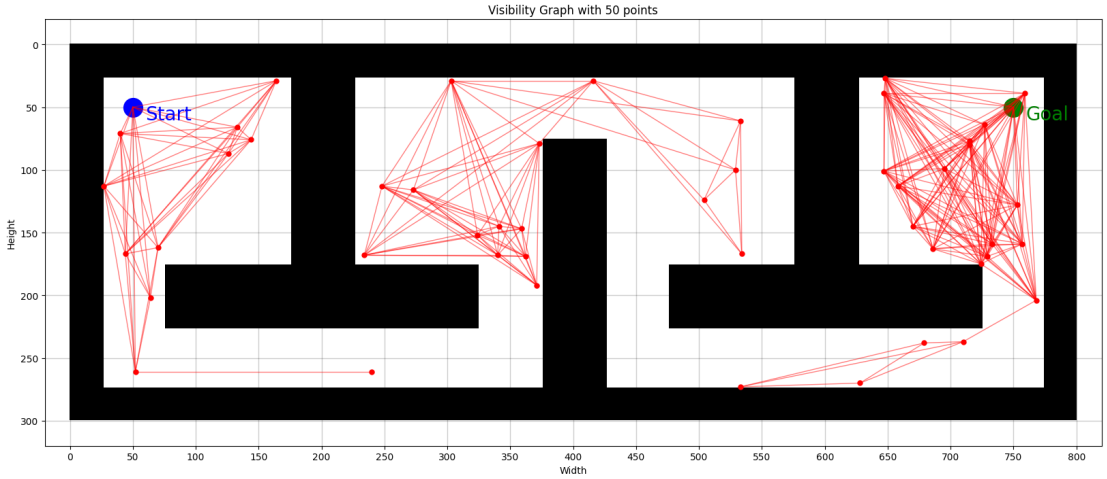
Problem 4

Use probabilistic roadmaps (PRM) to compute a path between start and end-points with 50, 100 and 500 sample points. What is the difference in path length? Illustrate each computed path.

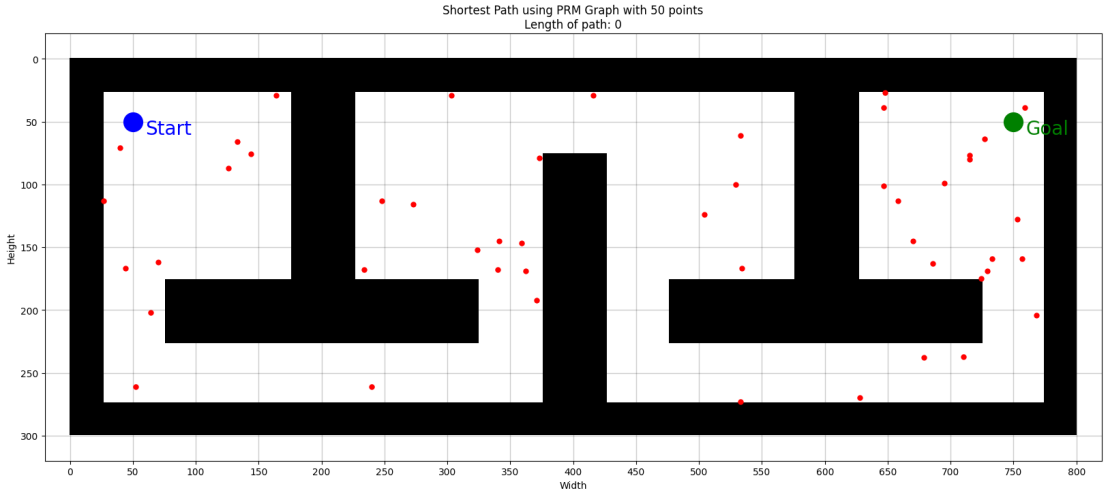
In [ ]:
```python
import networkx as nx
def has_line_of_sight(x1, y1, x2, y2, config_space):
    line = np.linspace([x1, y1], [x2, y2], max(abs(x2 - x1), abs(y2 - y1)) + 1).
    return not np.any(config_space[line[:, 1], line[:, 0]] == 1)

def get_points_and_edges(N, start_x, start_y, goal_x, goal_y, config_space_for_r
    # Generate N random points within the grid and add start and goal
    points = np.zeros((N, 2), dtype=int)
    points[0] = [start_x, start_y]
    points[1] = [goal_x, goal_y]

    for i in range(2,N):
        x = np.random.randint(0, width)
        y = np.random.randint(0, height)

        while hit_obstacle_2D(x, y, config_space_for_robot):
            x = np.random.randint(0, width)
            y = np.random.randint(0, height)

        points[i, 0] = x
        points[i, 1] = y

    # Find edges between points that have line no obstacles between them
    edges = []
    for i in range(N):
        for j in range(i + 1, N):
            x1, y1 = points[i]
            x2, y2 = points[j]

            if has_line_of_sight(x1, y1, x2, y2, config_space_for_robot):
                edges.append((i, j))

    return points, edges

def plot_points(points, config_space_for_robot, title="Visibility Graph"):
    plot_grid(config_space_for_robot, title)
    plt.plot(points[:, 0], points[:, 1], 'ro', markersize=5)
    # Plot the start and goal as different colors and with text
    plt.plot(points[0, 0], points[0, 1], 'bo', markersize=20)
    plt.plot(points[1, 0], points[1, 1], 'go', markersize=20)
    plt.text(points[0, 0] + 10, points[0, 1] + 10, 'Start', fontsize=20, color='
    plt.text(points[1, 0] + 10, points[1, 1] + 10, 'Goal', fontsize=20, color='g

def plot_points_and_edges(points, edges, config_space_for_robot, title="Visibili
    plot_points(points, config_space_for_robot, title)
    for i, j in edges:
        x1, y1 = points[i]
        x2, y2 = points[j]
        plt.plot([x1, x2], [y1, y2], 'r-', linewidth=1, alpha=alpha)

# Find the shortest path
def find_shortest_path(points, edges, start, goal):
    # Create NX graph
    G = nx.Graph()

    for i in range(len(points)):
        x = points[i, 0]
        y = points[i, 1]
        point = (x, y)
```

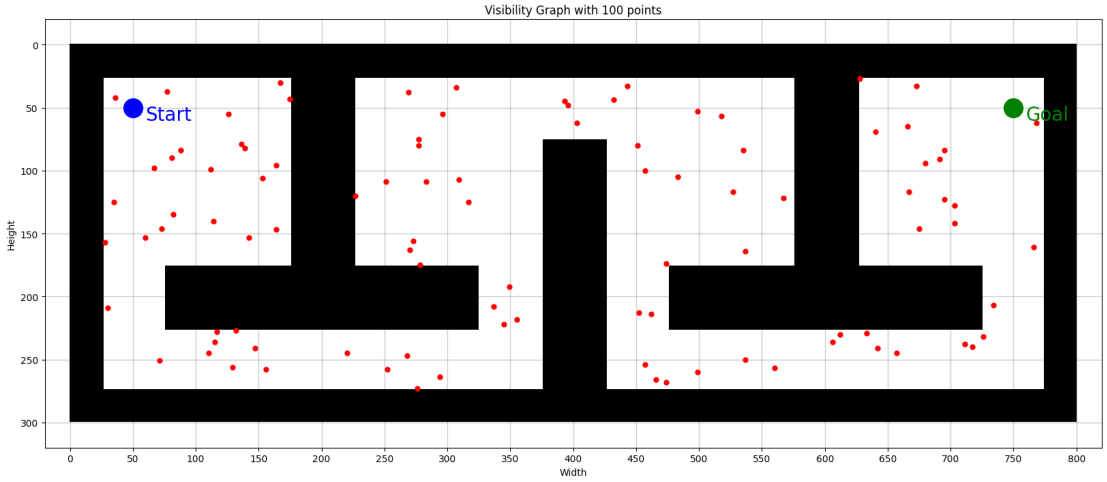
```
        G.add_node(point)

    for i, j in edges:
        x1, y1 = points[i]
        x2, y2 = points[j]
        point1 = (x1, y1)
        point2 = (x2, y2)
        length = np.sqrt((x1 - x2)**2 + (y1 - y2)**2)
        G.add_edge(point1, point2, weight=length)

    try:
        path = nx.shortest_path(G, start, goal, weight='weight')
    except nx.NetworkXNoPath:
        print("No path found")
        return []

    return path

# Probabilistic Roadmap
graph = config_spaces[:,:,0]
config_space_for_robot = config_spaces[:,:,0]

for n in [50,100,500]:
    N = n
    points, edges = get_points_and_edges(N, robot_x0, robot_y0, goal_x, goal_y,
    plot_points(points, config_space_for_robot, title="Visibility Graph with {}
    plt.show()
    plot_points_and_edges(points, edges, config_space_for_robot, title="Visibili
    plt.show()

    # Find the shortest path
    start = (robot_x0, robot_y0)
    goal = (goal_x, goal_y)

    path = find_shortest_path(points, edges, start, goal)
    print("Shortest Path:", path)
    print("Length of path:", len(path))

    path_length = 0
    for i in range(len(path)-1):
        x1, y1 = path[i]
        x2, y2 = path[i+1]
        path_length += np.sqrt((x1 - x2)**2 + (y1 - y2)**2)

    plot_points(points, config_space_for_robot, title="Shortest Path using PRM G
    plt.plot([x for x, y in path], [y for x, y in path],'--', linewidth=2)
    plt.show()
```

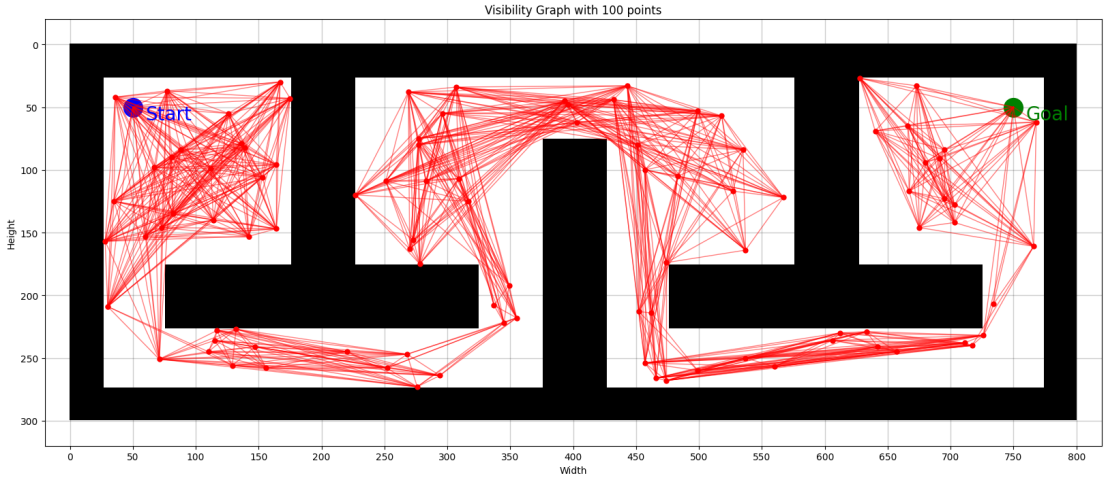




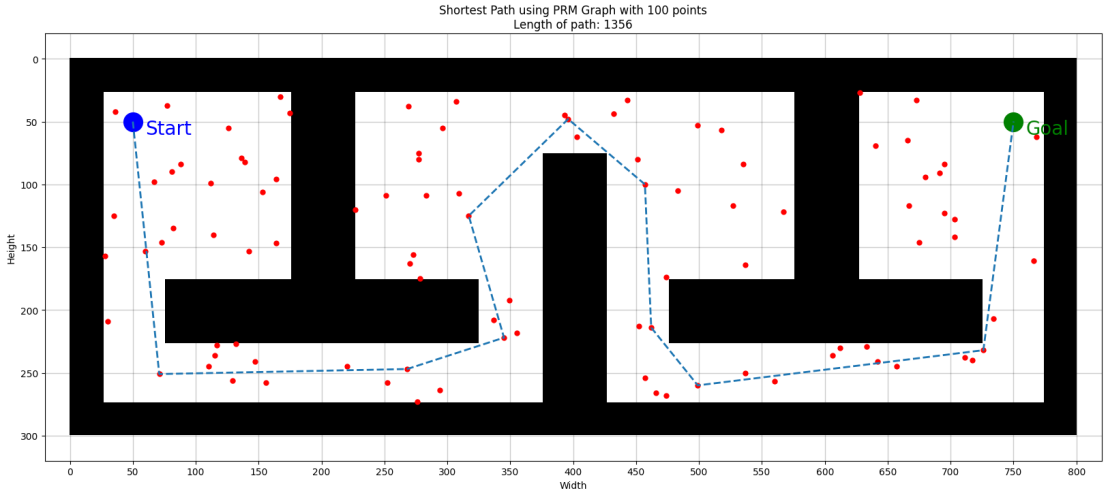

```
No path found
Shortest Path: []
Length of path: 0
```

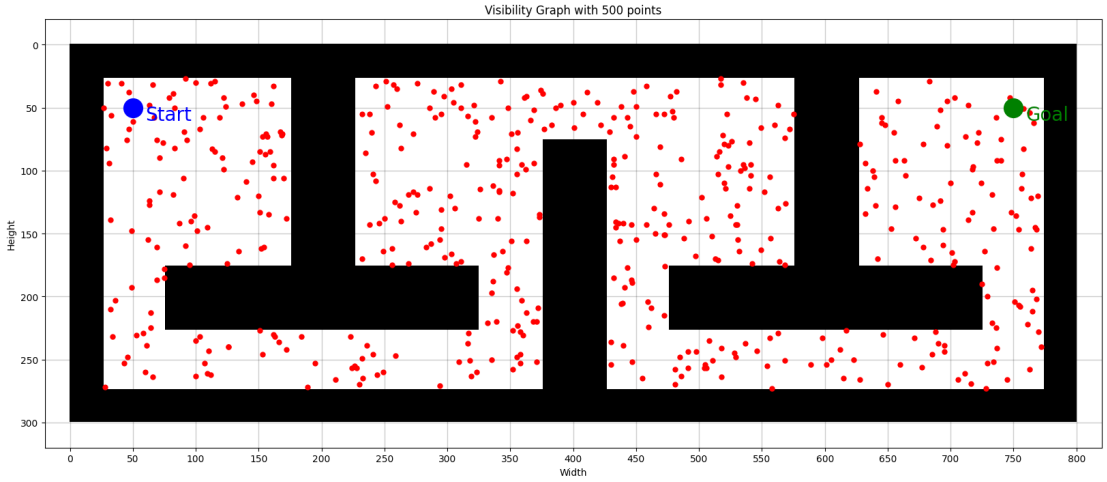




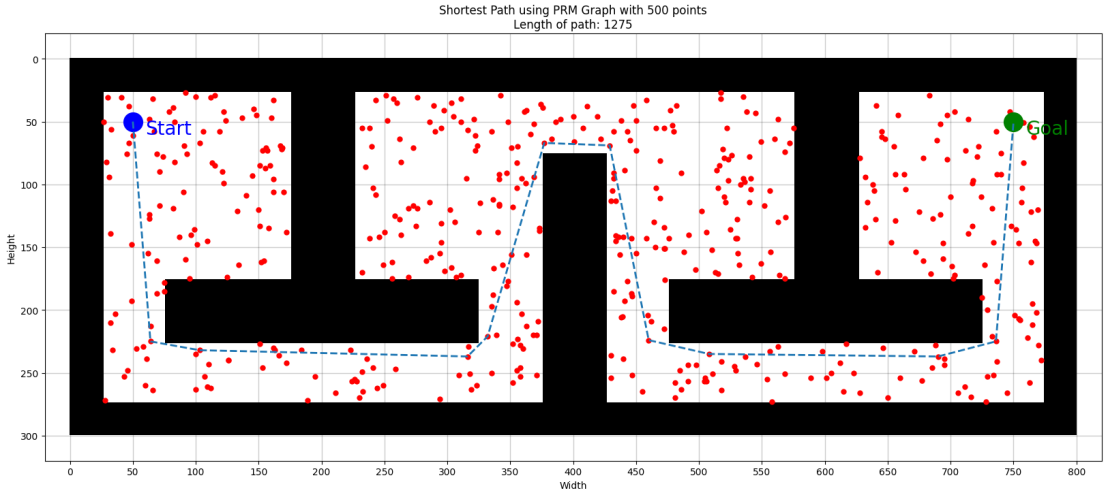



```
Shortest Path: [(50, 50), (71, 251), (268, 247), (345, 222), (317, 125), (396, 4
8), (457, 100), (462, 214), (499, 260), (726, 232), (750, 50)]
Length of path: 11
```






```
Shortest Path: [(50, 50), (64, 225), (103, 232), (316, 237), (332, 221), (377, 6
7), (429, 69), (460, 224), (508, 235), (690, 237), (736, 225), (750, 50)]
Length of path: 12
```



Problem 5

Do the same with Rapid exploring random trees (RRT). What are the main differences in performance between PRM and RRT? Illustrate each path.

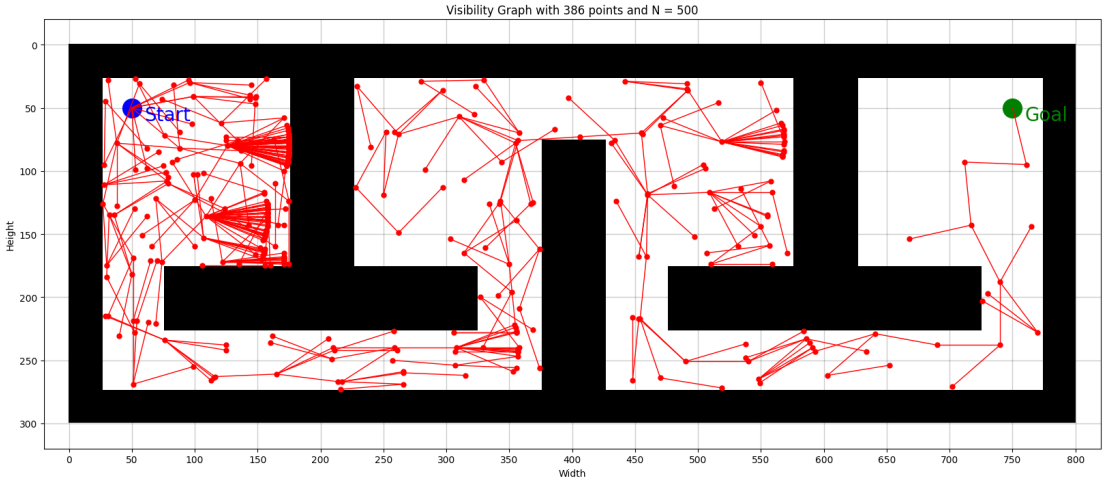
In [ ]:
```
# Rapid exploring random tree
def point_at_distance(start, end, distance):
```

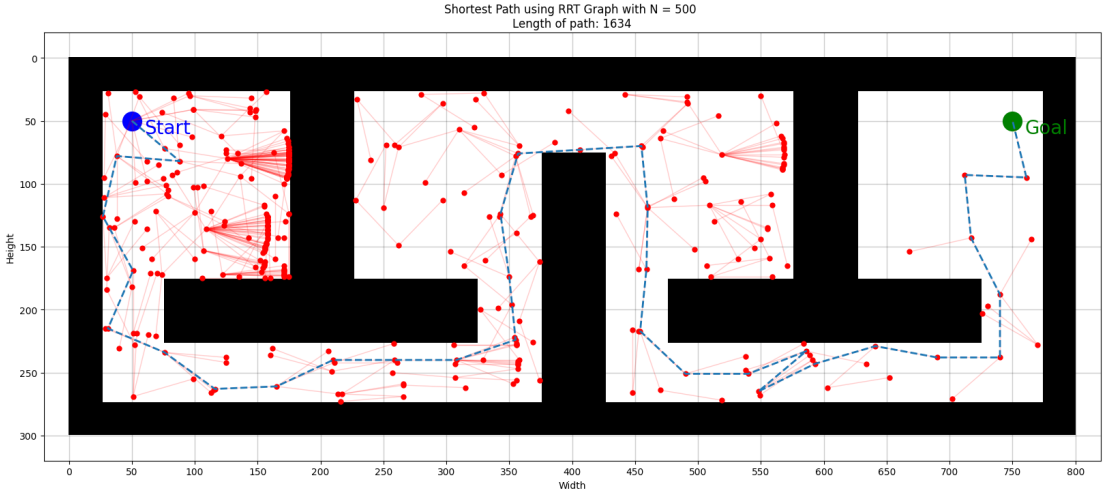
```
    # Convert points to numpy arrays for vector operations
    start = np.array(start)
    end = np.array(end)

    # Calculate the direction vector of the line
    direction_vector = end - start

    # Check if the direction vector is a zero vector
    if np.all(direction_vector == 0):
        return tuple(start)

    # Normalize the direction vector
    normalized_direction = direction_vector / np.linalg.norm(direction_vector)

    # Calculate the new point at the specified distance along the line
    new_point = start + normalized_direction * distance

    return tuple(new_point)

def bresenham_line(x1, y1, x2, y2):
    # Bresenham's line algorithm to get integer coordinates between (x1, y1) and
    points = []
    dx = abs(x2 - x1)
    dy = abs(y2 - y1)
    x, y = x1, y1
    sx = -1 if x1 > x2 else 1
    sy = -1 if y1 > y2 else 1

    if dx > dy:
        err = dx / 2.0
        while x != x2:
            points.append((x, y))
            err -= dy
            if err < 0:
                y += sy
                err += dx
            x += sx
    else:
        err = dy / 2.0
        while y != y2:
            points.append((x, y))
            err -= dx
            if err < 0:
                x += sx
                err += dy
            y += sy

    points.append((x, y))
    return np.array(points)

def has_line_of_sight(x1, y1, x2, y2, config_space):
    # Get integer coordinates between (x1, y1) and (x2, y2) using Bresenham's li
    line = bresenham_line(x1, y1, x2, y2)

    # Ensure that the indices are within the bounds of the config_space
    valid_indices = (line[:, 1] >= 0) & (line[:, 1] < config_space.shape[0]) & \
                    (line[:, 0] >= 0) & (line[:, 0] < config_space.shape[1])

    # Check if any obstacle is present along the line
    return not np.any(config_space[line[valid_indices, 1], line[valid_indices, 0
```

```
# Probabilistic Roadmap
for n in [500]:
    N = n
    graph = config_spaces[:,:,0]
    config_space_for_robot = config_spaces[:,:,0]

    max_distance = 50
    k = 1
    obs = 0
    goal_x = 750
    goal_y = 50

    # Create empty points np array
    points = np.zeros((N, 2), dtype=int)

    points[0] = [robot_x0, robot_y0]
    edges = []

    while k < N:
        # Generate random point
        x = np.random.randint(0, width)
        y = np.random.randint(0, height)

        # Check if point is valid
        while hit_obstacle_2D(x, y, config_space_for_robot):
            x = np.random.randint(0, width)
            y = np.random.randint(0, height)
        rand_point = (x, y)

        # Find closes point in tree to the random point
        distances = np.zeros(N)
        for i in range(N):
            distances[i] = np.sqrt((x - points[i][0])**2 + (y - points[i][1])**2

        points_sorted = points[np.argsort(distances)]
        dist_sorted = distances[np.argsort(distances)]

        for point in points_sorted:
            # Check if the point is the (0,0) point
            #if point[0] == 0 and point[1] == 0:
             #   continue

            closest_point = point
            point_in_direction = point_at_distance(point, rand_point, max_distan
            point_in_direction = (int(point_in_direction[0]), int(point_in_direc

            if has_line_of_sight(closest_point[0], closest_point[1], point_in_di
                points[k] = point_in_direction

                # Check if the point is close to the goal
                if np.sqrt((point_in_direction[0] - goal_x)**2 + (point_in_direc
                    edges.append((point_in_direction, (goal_x, goal_y)))
                    edges.append((point, point_in_direction))
                    k = N
                else:
                    edges.append((point, point_in_direction))
                k += 1
                break
```

```
# Remove all the points at (0,0)
points = points[~np.all(points == 0, axis=1)]


plot_grid(graph, "Visibility Graph with {} points".format(N))
# Plot title with number points and length of edges
plt.title("Visibility Graph with {} points and N = {}".format(len(points), N

plt.plot(points[:, 0], points[:, 1], 'ro', markersize=5)

# Plot start and goal
plt.plot(robot_x0, robot_y0, 'bo', markersize=20)
plt.plot(goal_x, goal_y, 'go', markersize=20)
plt.text(robot_x0 + 10, robot_y0 + 10, 'Start', fontsize=20, color='b')
plt.text(goal_x + 10, goal_y + 10, 'Goal', fontsize=20, color='g')

# Plot the edges
for i, j in edges:
    x1, y1 = i
    x2, y2 = j
    plt.plot([x1, x2], [y1, y2], 'r-', linewidth=1)

plt.show()

import networkx as nx

# Find the shortest path

# Create NX graph
G = nx.Graph()

for i in range(len(points)):
    x = points[i, 0]
    y = points[i, 1]
    point = (x, y)
    G.add_node(point)

for i, j in edges:
    x1, y1 = i
    x2, y2 = j
    point1 = (x1, y1)
    point2 = (x2, y2)
    length = np.sqrt((x1 - x2)**2 + (y1 - y2)**2)
    G.add_edge(point1, point2, weight=length)

try:
    path = nx.shortest_path(G, (robot_x0, robot_y0), (goal_x, goal_y), weigh
except nx.NetworkXNoPath:
    print("No path found")
    path = []

plot_grid(graph, "Shortest Path using RRT Graph with {} points".format(N))
plt.plot(points[:, 0], points[:, 1], 'ro', markersize=5)

# Plot start and goal with text
plt.plot(robot_x0, robot_y0, 'bo', markersize=20)
plt.plot(goal_x, goal_y, 'go', markersize=20)
plt.text(robot_x0 + 10, robot_y0 + 10, 'Start', fontsize=20, color='b')
plt.text(goal_x + 10, goal_y + 10, 'Goal', fontsize=20, color='g')
```

```
# Plot the edges
for i, j in edges:
    x1, y1 = i
    x2, y2 = j
    plt.plot([x1, x2], [y1, y2], 'r-', linewidth=1, alpha=0.2)

# Plot the path
plt.plot([x for x, y in path], [y for x, y in path],'--', linewidth=2)

path_length = 0
for i in range(len(path)-1):
    x1, y1 = path[i]
    x2, y2 = path[i+1]
    path_length += np.sqrt((x1 - x2)**2 + (y1 - y2)**2)

plt.title("Shortest Path using RRT Graph with N = {} \nLength of path: {}".f

plt.show()
```





References

[1] Erik Demaine and Srini Devadas. (2011). *MIT OpenCourseWare: Introduction to Algorithms - Lecture 16*. Retrieved from `https://ocw.mit.edu/courses/6-006-introduction-to-algorithms-fall-2011/6277a1f06100c26a7ff21031af6757b5_MIT6_006F11_lec16.pdf`, Accessed on: 6.des.2023.