Q1

Q1

(a)

$$P(\theta | Y_1 \cdots Y_m) = \frac{P(Y_1 \cdots Y_m | \theta) P(\theta)}{P(Y_1 \cdots Y_m)}$$

The estimated error implies that all Y are independent

Thus above $\propto \underline{p(Y_1|\theta)\, p(Y_2|\theta) \cdots p(Y_n|\theta)\, p(\theta)}$

(b) (7) $\Rightarrow$ $\varepsilon_i = \bar{Y}_i - A_i\theta - b_i$ plug in (3)

$$p(Y_i|\theta) = \frac{1}{\sqrt{\det(2\pi \Sigma_i)}} \exp\left(-\frac{1}{2}(\bar{Y}_i - A_i\theta - b_i - u_i)^T \Sigma_i^{-1} (\bar{Y}_i - A_i\theta - b_i - u_i)\right)$$

(c) We know that $p(\theta|Y_1 \cdots Y_m) \propto p(\theta) \prod_{i=1}^{m} p(Y_i|\theta)$

while $p(\theta)$ can plug into (3)

$$\Rightarrow p(\theta|Y_1 \cdots Y_m) \propto \exp\left(-\frac{1}{2}(\theta - u_0)^T \Sigma_0^{-1} (\theta - u_0)\right) \times \prod_{i=1}^{m} \exp\left(-\frac{1}{2}(\bar{Y}_i - A_i\theta - b_i - u_i)^T \Sigma_i^{-1} (\bar{Y}_i - A_i\theta - b_i - u_i)\right)$$

Q2

(a)

(i)

```python
def recoverPath(start, goal, pred):
    length = list()
    trace_back = [goal]
    while goal != start:
        length.append(d(goal, pred[goal]))
        goal = pred[goal]
        trace_back.append(goal)
    trace_back = list(reversed(trace_back))
    return trace_back, sum(length)
```

(ii)

```python
def A_star(Vertex, start, goal, Neighbor, weight, heuristic):
    # Initialization
    pred = dict()
    CostTo = dict()
    EstTotalCost = dict()
    Q = list()

    for v in Vertex:
        CostTo[v] = float('inf')
        EstTotalCost[v] = float('inf')

    CostTo[start] = 0
    EstTotalCost[start] = heuristic(start, goal)

    heapq.heappush(Q, (heuristic(start, goal), start))

    while Q:
        priority, v = heapq.heappop(Q)
        # If reached the goal, end the loop and start the trace back algorithm
        if v == goal:
            return recoverPath(start, goal, pred)
        for index in Neighbor(v):
            # print(f'{index}: {list(Neighbor(v))}')
            pvi = CostTo[v] + weight(v, index)
            if pvi < CostTo[index]:
                # Update based on heuristic
                pred[index] = v
                CostTo[index] = pvi
                EstTotalCost[index] = pvi + heuristic(index, goal)
                # insert here
                # heapq.has(index) ?
                # if any(index == b for a, b in Q):
                #     heapq.heapreplace()
                heapq.heappush(Q, (heuristic(index, goal), index))

    return None
```

(b)

(i)

```python
def N(v):
    col, row = v
    return (
        (col + g, row + h)
        for g in (-1, 0, 1)
        for h in (-1, 0, 1)
        if g != 0 or h != 0
        if 0 <= col + g & col + g < len(occupancy_map)
        if 0 <= row + h & row + h < len(occupancy_map[0])
        if occupancy_map[col + g][row + h] == 1)
```

This ensures that g and h is the 8 surrounding neighbor of the chosen v and it's not out of bound, and lastly it's a valid space. This pastes it all into a list and returns it.

(ii)

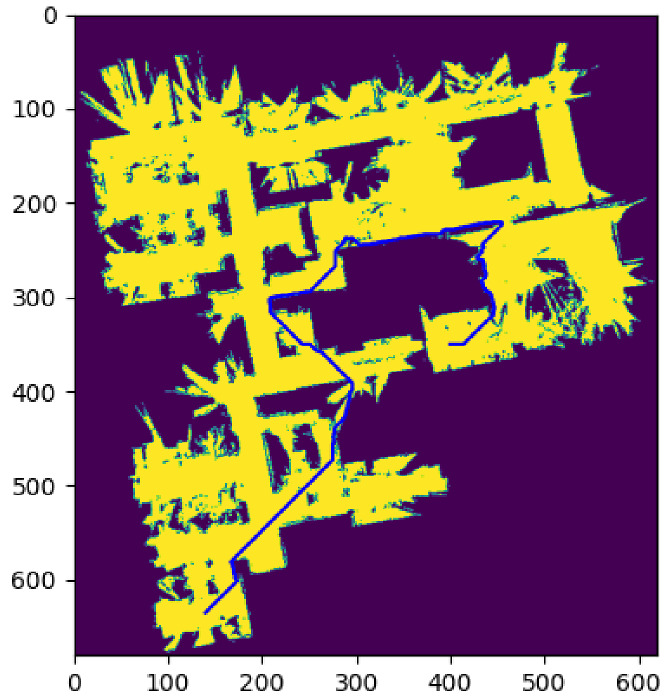
```python
def d(v1, v2):
    return math.dist([v1[0], v1[1]], [v2[0], v2[1]])
```

It can be done with math. dist in one line

(iii)

```python
V = list()

for i in range(0, len(occupancy_grid)):
    for j in range(0, len(occupancy_grid[1])):
        x = i, j
        V.append(x)

path, le = A_star(V, (635, 140), (350, 400), N, d, d)

print("Sum of length taken is: ", le)

# print(list(N((0, 0))))

matplotlib.pyplot.imshow(matplotlib.pyplot.imread(r'C:\Users\froze\Desktop\EECE 5550\HW2\occupancy_map.png'),
                         extent=[0, 620, 680, 0])
matplotlib.pyplot.plot(numpy.array(path)[:, 1], numpy.array(path)[:, 0], 'b')

matplotlib.pyplot.show()
```

And the graph generated for A* is as follows



The total length of the path taken:

```
Sum of length taken is:  938.2834046956888
```

938.2834

(c)

(i)

```python
def sample(G):
    row, col = G.shape
    sample1 = random.uniform(0, row)
    sample2 = random.uniform(0, col)
    while G.item(int(sample1), int(sample2)) == 0:
        sample1 = random.uniform(0, row)
        sample2 = random.uniform(0, col)
    out = (int(sample1), int(sample2))
    return out
```

(ii)

```python
def valid_Path(G, start, goal):
    for v in list(bresenham(start[0], start[1], goal[0], goal[1])):
        if G.item(v) == 0:
            return False
    return True
```

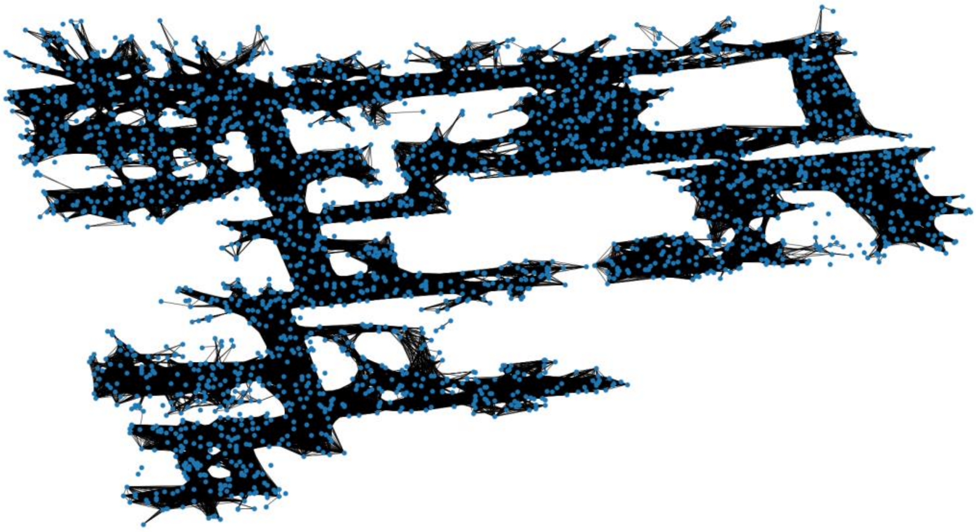
Bresenham returns all the vertices on a straight line and this algorithm checks if they are occupied or not

(iii)

```python
def d(v1, v2):
    return math.dist([v1[0], v1[1]], [v2[0], v2[1]])


def add_Vertex(G, vnew, dmax):
    global V, E
    V.append(vnew)
    # Nodes named after its number, with a pos attr that record its place in the graph
    PRM.add_node(PRM.number_of_nodes() + 1, pos=vnew)

    for i in V:
        if i != vnew and d(i, vnew) < dmax and valid_Path(G, i, vnew):
            PRM.add_edge((i, vnew), weight=d(i, vnew))
            E.append((i, vnew))
    return PRM


def construct_PRM(G, N, dmax):
    global V, E
    V = None
    E = None
    for _ in range(N):
        # if vnew not in G
        #   break
        # Implemented inside the sample function
        vnew = sample(G)
        add_Vertex(G, vnew, dmax)
    return PRM


V = list()
E = list()

PRM = networkx.Graph()
```

With the provided Algorithm, networkX provided the graph and the nodes and edges are all added to the PRM constructed by it.

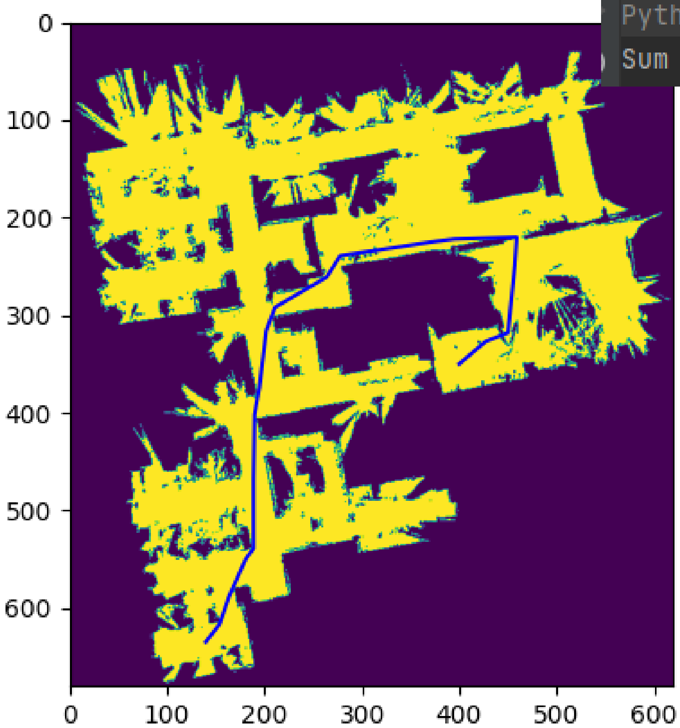
(iv)



```
V = list()
E = list()
N = 2500
dmax = 75
start = (635, 140)
goal = (350, 400)

PRM = nx.Graph()
PRM_complete = construct_PRM(occupancy_grid, N, dmax)

# Extract pos attr prom the PRM constructed
pos_list = nx.get_node_attributes(PRM_complete, 'pos')

# print(pos_list.get(1))

repos_list = {}

# Inverse the graph's coordinate to fit the original graph
for i in range(1, PRM_complete.number_of_nodes() + 1):
    inv_X = -pos_list.get(i)[0]
    inv_Y = pos_list.get(i)[1]
    repos_list[i] = (inv_Y, inv_X)

output_graph = matplotlib.pyplot.figure(1, figsize=(150, 150), dpi=60)
nx.draw_networkx(PRM_complete, repos_list, node_size=100, linewidths=0.1, with_labels=False)
```

(v)

```python
add_Vertex(occupancy_grid, start, dmax)
add_Vertex(occupancy_grid, goal, dmax)

startIndex = PRM_complete.number_of_nodes() - 1
goalIndex = PRM_complete.number_of_nodes()

# Find the index of start and goal (in case they are already added)
for t in range(PRM_complete.number_of_nodes()):
    if V[t] == start:
        startIndex = t + 1
    if V[t] == goal:
        goalIndex = t + 1

# Use A* Algorithm provided on the two new nodes
A_star = nx.astar_path(PRM_complete, startIndex, goalIndex)
length = nx.astar_path_length(PRM_complete, startIndex, goalIndex)

print("Sum of length taken is: ", length)

path = list()
for e in A_star:
    path.append(PRM_complete.nodes[e]['pos'])

print(path)

matplotlib.pyplot.imshow(matplotlib.pyplot.imread(
    r'C:\Users\froze\Desktop\EECE 5550\HW2\occupancy_map.png'),extent=[0, 620, 680, 0])

matplotlib.pyplot.plot(numpy.array(path)[:, 1], numpy.array(path)[:, 0], 'b')

matplotlib.pyplot.show()
```




```
Python 控制台
Sum of length taken is:  786.5429559510968
```

The length of the A* + PRM

path is 786.543