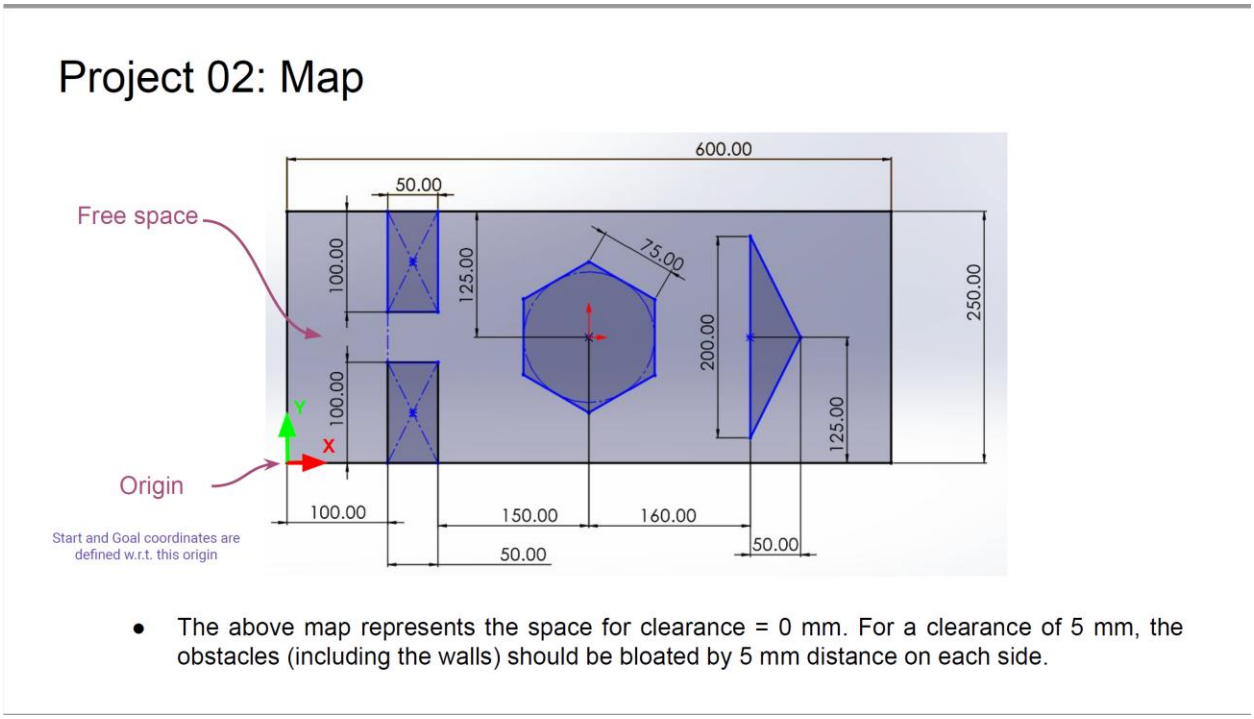
ENPM_661 PLANNING

PROJECT 2: DIJKSTRA ALGORITHM FOR POINT ROBOT

HARSHAL SHIRSATH, 119247419

shirsath@umd.edu

MAP



- The above map represents the space for clearance = 0 mm. For a clearance of 5 mm, the obstacles (including the walls) should be bloated by 5 mm distance on each side.

Libraries used: Numpy, Cv2, Time, Heapq

CODE:

```
import cv2
import heapq as hq
import numpy as np
import time

index = 1

######## DEFINING 8 ACTION SET ########
def plot(surface,start,goal_path):
    nodes = {}
```

```
    nodes[start] = (0,0,None)

    open_loop = []
    close_loop = set()

    ######## DEFINING COST
    ######## X COORDINATES FOR EVERY MOVE
    ######## Y COORDINATES FOR EVERY MOVE
    def north(surface, x_ynode, cost):
        x,y = x_ynode
        x = x
        if y < surface.shape[0]-1:
            y += 1
            cost += 1
            node_new = (x,y)
            return (surface,node_new,cost)
        return (surface,None, None)

    def northeast(surface, x_ynode, cost):
        x,y = x_ynode
        if x < (surface.shape[1]-1) and y < (surface.shape[0]-1) :
            x += 1
            y += 1
            cost += 1.4
            node_new = (x,y)
            return (surface,node_new, cost)
        return (surface,None, None)

    def east(surface, x_ynode, cost):
        x,y = x_ynode
        y = y
        if x < (surface.shape[1]-1):
            x += 1
            cost += 1
            node_new = (x,y)
            return (surface,node_new,cost)
        return (surface,None, None)

    def southeast(surface, x_ynode, cost):
        x,y = x_ynode
        if x < (surface.shape[1]-1) and y > 0 :
            x += 1
            y -= 1
            cost += 1.4
            node_new = (x,y)
```

```
            return (surface,node_new,cost)
        return (surface,None, None)

    def south(surface, x_ynode, cost):
        x,y = x_ynode
        x = x
        if y > 0 :
            y -= 1
            cost += 1
            node_new = (x,y)
            return (surface,node_new,cost)
        return (surface,None, None)

    def southwest(surface, x_ynode, cost):
        x,y = x_ynode
        if x > 0 and y > 0 :
            x -= 1
            y -= 1
            cost += 1.4
            node_new = (x,y)
            return (surface,node_new,cost)
        return (surface,None, None)

    def west(surface, x_ynode, cost):
        x,y = x_ynode
        y = y
        if x > 0:
            x -= 1
            cost += 1
            node_new = (x,y)
            return (surface,node_new,cost)
        return (surface,None, None)

    def northwest(surface, x_ynode, cost):
        x,y = x_ynode
        if x > 0 and y < surface.shape[0]-1 :
            x -= 1
            y += 1
            cost += 1.4
            node_new = (x,y)
            return (surface,node_new,cost)
        return (surface,None,None)


    ###### TO CHECK VISITED NODES ######
```

```
    def to_be_visited(surface,x_ynode):
        parent = x_ynode
        peak_cost = nodes[x_ynode][0]
        for i in [north,northeast,east,southeast,south,southwest,west,northwest]:
            surface,node_new,cost = i(surface,x_ynode, peak_cost)
            if node_new == None:
                continue
            (x,y)  = node_new
            y_up = (surface.shape[0]-1)-y
            if node_new in close_loop:
                continue
            layer = visual(surface,node_new)
            if np.any(layer[:,:,1]):
                continue

            if node_new not in nodes or cost < nodes[node_new][0]:
                global index
                index += 1
                nodes[(x,y)] = (cost, index, parent)

                hq.heappush(open_loop, (cost,node_new))
            if node_new == goal_path:
                break

    ###### VISUALIZE THE NODE POSITIONING #####
    def visual(frame,x_ynode):
        x,y = x_ynode
        r_circle = 5
        layer = np.zeros_like(frame)
        cv2.circle(layer, (x, y), r_circle, (255,255,255), -1)
        mask_circle = cv2.bitwise_and(frame, layer)

        return mask_circle
    ###### DEFINING POINTER AND TRACKING IT'S POSITION WHEN EXPLORED ######
    def pointer(surface,trace):
        r_circle = 1
        color = (0, 255, 255)

        for x_ynode in trace:
            x,y = x_ynode

            cv2.circle(surface, (x,y), r_circle, color, 1)
            cv2.imshow("Map", surface)
            cv2.waitKey(2)
            cv2.circle(surface, (x,y), r_circle+1, (255,0,0), -1)
```

```
            cv2.imshow("Map", surface)
            cv2.waitKey(1)

        cv2.waitKey(2000)
        cv2.destroyAllWindows()
    ######## CHECKING IF THE NODE WAS ALREADY EXPLORED OR NOT IN THE CLOSED SET
#######
    def visualisation(surface, nodes):
        parents_node = {}
        for x_ynode, (cost, index, parent) in nodes.items():
            if parent is not None:
                if parent not in parents_node:
                    parents_node[parent] = []
                parents_node[parent].append(x_ynode)
        color = (0,0,255)
        initial = 0


        for parent, visited_nodes in parents_node.items():
            surface_shape = surface.shape[0]-1
            x_ycoo = np.array([(surface_shape-x_ynode[1], x_ynode[0]) for x_ynode
in visited_nodes]).transpose()
            surface[x_ycoo[0], x_ycoo[1], :] = color
            initial += len(visited_nodes)
            if initial % 100 == 0:
                cv2.waitKey(1)

     #######DIJKSTRA ALGORITHM #####
    def tracking(nodes, start, goal_path):
        trace = []
        x_ynode = goal_path
        while x_ynode != start:
            trace.append(x_ynode)
            parent = nodes[x_ynode][2]
            x_ynode = parent
        trace.append(start)
        trace.reverse()
        shift = surface.shape[0]-1
        traced = [((x,shift-y)) for (x,y) in trace]
        return traced

    hq.heappush(open_loop,(0,start))

    while open_loop:
        cost, x_ynode = hq.heappop(open_loop)
```

```
        if x_ynode in close_loop:
            continue

        close_loop.add(x_ynode)
        if x_ynode == goal_path:
            print("Goal Reached\n\n")
            visualisation(surface, nodes)
            trace = tracking(nodes,start,goal_path)
            # track_animate(surface,trace)
            pointer(surface,trace)
            cv2.waitKey(4000)
            cv2.destroyAllWindows()
            return nodes

        if goal_path in open_loop:
            continue
        to_be_visited(surface,x_ynode)

######## SHOWING FRAMES WITH RESPECTIVE NODES EXPLORED AFTER FILTERING ########
    def show(frame):
        cv2.imshow('Surface',frame)

    def pop_color(frame,node,color):

        x,y = node
        x = x
        y = (frame.shape[0]-1)-y
        color = np.array([255, 0, 0])
        frame[x][y] = color

def visualize(start,goal):
    def shift_origin(frame,node):
        x,y = node
        x = x
        y = (frame.shape[0]-1)-y
        return (x,y)

    ####### STARTING GOAL NODE #######
    def start_init(frame,node):
        x,y = shift_origin(frame,node)
        cv2.circle(frame,(x,y),5,(0,0,255),-1)
        return frame

    ####### INITIATING GOAL NODE #######
    def goal_init(frame,node):
```

```
        x,y = shift_origin(frame,node)
        cv2.circle(frame,(x,y),5,(0,0,255),-1)
        return frame
    obstacles = (0,255,0)


    ####### TRIANGLE OBSTACLE #######
    def triangle(frame, width, height, x_coord,y_coord, angle=0):
        x_coord = x_coord
        y_coord = (frame.shape[0]-1)-y_coord
        x_3, y_3 = x_coord, y_coord - width/2
        x_2, y_2 = x_coord, y_coord + width/2
        x_1, y_1 = x_coord+height, y_coord
        points = np.array([[x_1, y_1], [x_2, y_2], [x_3, y_3]], dtype=np.int32)
        cv2.fillPoly(frame, [points], obstacles)
        return frame

    ####### HEXAGON OBSTACLE #######
    def poly_shape(frame,edges, length, x_axis, y_axis,shape):
        x_axis = x_axis
        y_axis = (frame.shape[0]-1)-y_axis
        the_ta = 2*np.pi/edges
        p = length*0.5/np.sin(the_ta/2)
        x = []
        y = []
        for j in range(edges):
            x.append(x_axis + p*np.cos(the_ta*j + shape))
            y.append(y_axis + p*np.sin(the_ta*j + shape))

        points = np.array([[int(x[j]), int(y[j])] for j in range(edges)],
dtype=np.int32)
        cv2.fillPoly(frame, [points], obstacles)

        return frame

    ####### RECTANGLE OBSTACLE #######
    def rectangle(frame, height,w,x_coord,y_coord):
        x_coord = x_coord
        y_coord = (frame.shape[0]-1)-y_coord
        height = height/2
        w = w/2
        pt1 = (int(x_coord + w), int(y_coord - height))
        pt2 = (int(x_coord - w), int(y_coord + height))

        cv2.rectangle(frame,pt1,pt2,obstacles,-1)
```

```
        return frame

    ####### DEFINING PLANE SURFACE COORIDNATES #######
    height = 600
    width = 250
    surface = np.zeros((width,height,3),np.uint8)
    surface = triangle(surface,200,50,460,125)
    surface = rectangle(surface,100,50,125,50)
    surface = poly_shape(surface,6,75,300,125,np.pi/2)
    surface = rectangle(surface,100,50,125,200)


    x,y = start
    y_axis_invert = (surface.shape[0]-1)-y
    if np.array_equal(surface[y_axis_invert, x], np.array([255, 0 , 0])):
        raise ValueError("Co-ordinate for start cannot be in obstacle space")

    x,y = goal
    y_axis_invert = (surface.shape[0]-1)-y
    if np.array_equal(surface[y_axis_invert, x], np.array([255,0, 0])):
        raise ValueError("Co-ordinate for goal cannot be in obstacle space")

    surface = start_init(surface,start)
    surface = goal_init(surface,goal)

    return surface

####### TAKING INPUTS AND PRINTING TIME TAKEN TO COMPETE DIJKSTRA ALGORITHM
#######
if __name__=="__main__":

    starting_time=time.time()
    starting_time = time.time()
    x, y = [int(x) for x in input("Start coordinates to be enterred with a space:
").split()]

    if not (0 <= x < 600) or not (0 <= y < 250):
        # raise ValueError("Start coordinates are not acceptable")
        print("Start coordinates are not acceptable")

    start = (x,y)

    x, y = [int(x) for x in input("Start coordinates to be enterred with a space:
").split()]
    if not (0 <= x < 600) or not (0 <= y < 250):
```

```
        print("Goal coordinates are not acceptable")
    goal = (x,y)

    surface = visualize(start,goal)
    plot(surface,start,goal)

#######PRINTING TIME TAKEN TO COMPLETE #####
    ending_time = time.time()
    timetorun = ending_time - starting_time
    print("Time to run: {:.4f} seconds".format(timetorun))
```