**ENPM\_661 PLANNING**

# **PROJECT 2: DIJKSTRA ALGORITHM FOR POINT ROBOT**

## HARSHAL SHIRSATH, 119247419

[shirsath@umd.edu](mailto:shirsath@umd.edu)

MAP

Chart

Description automatically generated with medium confidence

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))