**a_star_FNU_aryan_keyur.py**

```
#ENPM 661: Planning for Robotics
#Project 3 - Phase 1
#authors: Keyur Borad, Aryan Mishra, FNU Koustubh



# import libraries

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

start_time = time.time()
canvas = np.ones((501,1201,3))   # creating a frame for video generation
obstacle_set = set()             # set to store the obstacle points
obstacle_list = []               # list to store the obstacle points in order for videp

c2c_node_grid = [[float('inf')] * 500 for _ in range(1200)]       # create a 2D array for storing cost to come
tc_node_grid = [[float('inf')] * 500 for _ in range(1200)]        # create a 2D array for storing cost to come
closed_set = []               # set to store the value of visited and closed points
closed_list = np.zeros((1200, 500, 12))
visited={}

'''
Loop to define the obstacle points in the map
'''
C = int(input("Enter the clearance from the obstacle in pixel: "))     # Get clearance from the user
R = int(input("Enter the radius of the robot in pixel: "))             # Get the robot radius from user


x_goal = 0
y_goal = 0
x_start = 0
y_start = 0

# Funtion to update the visted nodes
def visited_node(node):
    visited.update({node[2]:node[4]})

def move_forward(node):
    new_heading = node[5]
    x = node[4][0] + L*np.cos(np.deg2rad(new_heading))
    y = node[4][1] + L*np.sin(np.deg2rad(new_heading))
    x = round(x)
    y = round(y)
    c2c = node[1]+L
    c2g = math.sqrt((y_goal-y)**2 + (x_goal-x)**2)
```

```
    tc = c2c + c2g
    return (x,y),new_heading,tc,c2c


def move_30(node):
    new_heading = (node[5] + 30) % 360
    x = node[4][0] + L*np.cos(np.deg2rad(new_heading))
    y = node[4][1] + L*np.sin(np.deg2rad(new_heading))
    x = round(x)
    y = round(y)
    c2c = node[1]+L
    c2g = math.sqrt((y_goal-y)**2 + (x_goal-x)**2)
    tc = c2c + c2g
    return (x,y),new_heading,tc,c2c


def move_minus_30(node):
    new_heading = (node[5] - 30) % 360
    x = node[4][0] + L*np.cos(np.deg2rad(new_heading))
    y = node[4][1] + L*np.sin(np.deg2rad(new_heading))
    x = round(x)
    y = round(y)
    c2c = node[1]+L
    c2g = math.sqrt((y_goal-y)**2 + (x_goal-x)**2)
    tc = c2c + c2g
    return (x,y),new_heading,tc,c2c


def move_60(node):
    new_heading = (node[5] + 60) % 360
    x = node[4][0] + L*np.cos(np.deg2rad(new_heading))
    y = node[4][1] + L*np.sin(np.deg2rad(new_heading))
    x = round(x)
    y = round(y)
    c2c = node[1]+L
    c2g = math.sqrt((y_goal-y)**2 + (x_goal-x)**2)
    tc = c2c + c2g
    return (x,y),new_heading,tc,c2c

def move_minus_60(node):
    new_heading = (node[5] - 60) % 360
    x = node[4][0] + L*np.cos(np.deg2rad(new_heading))
    y = node[4][1] + L*np.sin(np.deg2rad(new_heading))
    x = round(x)
    y = round(y)
    c2c = node[1]+L
    c2g = math.sqrt((y_goal-y)**2 + (x_goal-x)**2)
    tc = c2c + c2g
    return (x,y),new_heading,tc,c2c

T = C + R

for y in range(500):
    for x in range(1200):
        canvas[y,x] = [255,255,255]
        if (0<=y<=T):                       # points in the bottom boundary
```

```
113             obstacle_set.add((x,y))
114             obstacle_list.append((x,y))
115             c2c_node_grid[x][y] = -1
116             tc_node_grid[x][y] = -1
117         elif (0<=x<=T):                     # points in the left boundary
118             obstacle_set.add((x,y))
119             obstacle_list.append((x,y))
120             c2c_node_grid[x][y] = -1
121             tc_node_grid[x][y] = -1
122         elif (500-T<=y<500):                  # points in the top boundary
123             obstacle_set.add((x,y))
124             obstacle_list.append((x,y))
125             c2c_node_grid[x][y] = -1
126             tc_node_grid[x][y] = -1
127         elif (1200-T<=x<1200):                # points in the right boundary
128             obstacle_set.add((x,y))
129             obstacle_list.append((x,y))
130             c2c_node_grid[x][y] = -1
131             tc_node_grid[x][y] = -1
132         elif (100-T<=x<=175+T) and (100-T<=y<500):         # points in first
    rectangle
133             obstacle_set.add((x,y))
134             obstacle_list.append((x,y))
135             c2c_node_grid[x][y] = -1
136             tc_node_grid[x][y] = -1
137         elif (275-T<=x<=350+T and 0<=y<=400+T):          # points in second rectangle
138             obstacle_set.add((x,y))
139             obstacle_list.append((x,y))
140             c2c_node_grid[x][y] = -1
141             tc_node_grid[x][y] = -1
142 
143         # Points int the C-shaped obstacle
144         elif (900-T<=x<=1015+T) and (50-T<=y<=125+T):
145             obstacle_set.add((x,y))
146             obstacle_list.append((x,y))
147             c2c_node_grid[x][y] = -1
148             tc_node_grid[x][y] = -1
149         elif (1020-T<=x<1100+T and 50-T<=y<=450+T):
150             obstacle_set.add((x,y))
151             obstacle_list.append((x,y))
152             c2c_node_grid[x][y] = -1
153             tc_node_grid[x][y] = -1
154         elif (900-T<=x<=1015+T and 375-T<=y<=450+T):
155             obstacle_set.add((x,y))
156             obstacle_list.append((x,y))
157             c2c_node_grid[x][y] = -1
158             tc_node_grid[x][y] = -1
159 
160         # Points in the hexagon obstacle
161         elif(520-T<=x<=780+T) and (175<=y<=325):
162             obstacle_set.add((x,y))
163             obstacle_list.append((x,y))
164             c2c_node_grid[x][y] = -1
165             tc_node_grid[x][y] = -1
166         elif(325<=y<=400+T) and ((((y-400-T)*(520-T-650)/(325-400-T))-(x-650))<=0)
    and ((((y-400-T)*(780+T-650)/(325-400-T))-(x-650))>=0):
167             obstacle_set.add((x,y))
```

```
            obstacle_list.append((x,y))
            c2c_node_grid[x][y] = -1
            tc_node_grid[x][y] = -1
        elif(100-T<=y<=175) and ((((y-175)*(650-520+T)/(100-T-175))-(x-520+T)))<=0 and ((((y-175)*(650-780-T)/(100-T-175))-(x-780-T))>=0):
            obstacle_set.add((x,y))
            obstacle_list.append((x,y))
            c2c_node_grid[x][y] = -1
            tc_node_grid[x][y] = -1

valid_start = False

while not valid_start:
        x_start = int(input("Enter the start x coordinate: "))
        y_start = int(input("Enter the start y coordinate: "))
        theta_start = int(input("Enter the initial heading: "))
        if (x_start, y_start) in obstacle_set:
            print("Invalid coordinates, Enter again: ")
        else:
            initial_node = (0, 0, 1, [], (x_start, y_start), theta_start)
            valid_start = True

valid_goal = False

while not valid_goal:
        x_goal = int(input("Enter the goal x coordinate: "))
        y_goal = int(input("Enter the goal y coordinate: "))
        theta_goal = int(input("Enter the goal heading: "))
        if (x_goal, y_goal) in obstacle_set:
            print("Invalid coordinates, Enter again: ")
        else:
            goal = (x_goal, y_goal)
            valid_goal = True

valid_step = False

while not valid_step:
        L = int(input("Enter step size: "))
        if not (1 <= L <= 10):
            print("Invalid Step Size, Enter Again: ")
        else:
            valid_step = True


new_index = 1
open_list = []
hq.heappush(open_list,initial_node)        # Push initial node to the list
hq.heapify(open_list)                      # covers list to heapq data type


while(open_list):
    # total cost ,cost to come, index, parent node index, coordinate values (x,y), orientation
    node = hq.heappop(open_list)       # pop the node with lowest cost to come
    closed_set.append(node[4])            # add the node coordinates to closed set
    closed_list[int(node[4][0]), int(node[4][1]), int(node[5]/30)] = 1         # add the node to the closed list
```

```
222     visited_node(node)
223     index = node[2]                     # store the index of the current node
224     parent_index = node[3]              # store the parent index list of current node
225 
226     node_dist = math.sqrt((node[4][0]-x_goal)**2 + (node[4][1]-y_goal)**2)
227 
228     if node_dist<3 and (abs(node[5]-theta_goal)<=30 or abs(theta_goal-node[5])<=30):
# if the node is goal position, exit the loop
229         print("Goal reached")
230         break
231 
232     point, new_heading, tc, c2c = move_30(node)                          # get the new
node's coordinates and cost to come
233     # print(int(point[0]), int(point[1]), int(new_heading/30))
234     if point not in obstacle_set and closed_list[int(point[0]), int(point[1]),
int(new_heading/30)] == 0:           # check if the new node is in the obstacle set
or visited list
235         x = point[0]
236         y = point[1]
237         if tc<tc_node_grid[x][y]:                                          # check if
the new cost to come is less than original cost to come
238             new_parent_index = parent_index.copy()
239             new_parent_index.append(index)                               # Append the
current node's index to the new node's parent index list
240             new_index+=1
241             tc_node_grid[x][y] = tc
242             c2c_node_grid[x][y] = c2c                                      # Update
the new cost to come
243             new_node = (tc, c2c, new_index, new_parent_index, point, new_heading)
244             hq.heappush(open_list, new_node)                             # push the
new node to the open list
245             # print(new_node)
246 
247     point, new_heading, tc, c2c = move_60(node)                                    #
get the new node's coordinates and cost to come
248     # print(int(point[0]), int(point[1]), int(new_heading/30))
249     if point not in obstacle_set and closed_list[int(point[0]), int(point[1]),
int(new_heading/30)] == 0:           # check if the new node is in the obstacle set
or visited list
250         x = point[0]
251         y = point[1]
252         if tc<tc_node_grid[x][y]:                                          # check if
the new cost to come is less than original cost to come
253             new_parent_index = parent_index.copy()
254             new_parent_index.append(index)                               # Append the
current node's index to the new node's parent index list
255             new_index+=1
256             tc_node_grid[x][y] = tc
257             c2c_node_grid[x][y] = c2c                                      # Update
the new cost to come
258             new_node = (tc, c2c, new_index, new_parent_index, point, new_heading)
259             hq.heappush(open_list, new_node)                             # push the
new node to the open list
260             # print(new_node)
261 
262 
263     point, new_heading, tc, c2c = move_forward(node)
# get the new node's coordinates and cost to come
264     # print(int(point[0]), int(point[1]), int(new_heading/30))
```

```
265     if point not in obstacle_set and closed_list[int(point[0]), int(point[1]), int(new_heading/30)] == 0:           # check if the new node is in the obstacle set or visited list
266         x = point[0]
267         y = point[1]
268         if tc<tc_node_grid[x][y]:                                                   # check if the new cost to come is less than original cost to come
269             new_parent_index = parent_index.copy()
270             new_parent_index.append(index)                                      # Append the current node's index to the new node's parent index list
271             new_index+=1
272             tc_node_grid[x][y] = tc
273             c2c_node_grid[x][y] = c2c                                             # Update the new cost to come
274             new_node = (tc, c2c, new_index, new_parent_index, point, new_heading)
275             hq.heappush(open_list, new_node)                                    # push the new node to the open list
276             # print(new_node)
277
278
279     point, new_heading, tc, c2c = move_minus_30(node)                           # get the new node's coordinates and cost to come
280     # print(int(point[0]), int(point[1]), int(new_heading/30))
281     if point not in obstacle_set and closed_list[int(point[0]), int(point[1]), int(new_heading/30)] == 0:           # check if the new node is in the obstacle set or visited list
282         x = point[0]
283         y = point[1]
284         if tc<tc_node_grid[x][y]:                                                   # check if the new cost to come is less than original cost to come
285             new_parent_index = parent_index.copy()
286             new_parent_index.append(index)                                      # Append the current node's index to the new node's parent index list
287             new_index+=1
288             tc_node_grid[x][y] = tc
289             c2c_node_grid[x][y] = c2c                                             # Update the new cost to come
290             new_node = (tc, c2c, new_index, new_parent_index, point, new_heading)
291             hq.heappush(open_list, new_node)                                    # push the new node to the open list
292             # print(new_node)
293
294
295     point, new_heading, tc, c2c = move_minus_60(node)                           # get the new node's coordinates and cost to come
296     # print(int(point[0]), int(point[1]), int(new_heading/30))
297     if point not in obstacle_set and closed_list[int(point[0]), int(point[1]), int(new_heading/30)] == 0:           # check if the new node is in the obstacle set or visited list
298         x = point[0]
299         y = point[1]
300         if tc<tc_node_grid[x][y]:                                                   # check if the new cost to come is less than original cost to come
301             new_parent_index = parent_index.copy()
302             new_parent_index.append(index)                                      # Append the current node's index to the new node's parent index list
303             new_index+=1
304             tc_node_grid[x][y] = tc
305             c2c_node_grid[x][y] = c2c                                             # Update the new cost to come
```

```
            new_node = (tc, c2c, new_index, new_parent_index, point, new_heading)
            hq.heappush(open_list, new_node)                                # push the
new node to the open list
            # print(new_node)
    # print(open_list)
    # break

print(node[4])
print(node[5])









# Mark the obstacle points in the frame, including points after bloating
for point in obstacle_list:
    canvas[point[1],point[0]] = [255, 0, 0]

# Draw the obstacles in the frame, excluding the points after bloating
cv2.rectangle(canvas, (100, 499), (175, 100), (0 , 0, 255), -1)
cv2.rectangle(canvas, (275, 400), (350, 0), (0 , 0, 255), -1)
cv2.rectangle(canvas, (900, 125), (1100, 50), (0 , 0, 255), -1)
cv2.rectangle(canvas, (900, 450), (1100, 375), (0 , 0, 255), -1)
cv2.rectangle(canvas, (1020, 450), (1100, 50), (0, 0, 255), -1)
pts = np.array([[650, 400], [780, 325],
                [780, 175], [650, 100],
                [520, 175], [520, 325]],
                np.int32)
canvas = cv2.fillPoly(canvas, [np.array(pts)], color=(0, 0, 255))
cv2.circle(canvas,(x_start, y_start), 5, (0,0,255),-1)
cv2.circle(canvas,(x_goal, y_goal), 5, (0,0,255), -1)


print("Processing Video...")

path = node[3]            # Get the parent node list
counter = 0               # counter to count the frames to write on video

fourcc = cv2.VideoWriter_fourcc(*'mp4v')  # Codec for MP4 format
video_writer = cv2.VideoWriter('output.mp4', fourcc, 60, (1200, 500))

'''
Loop to mark the explored nodes in order on the frame
'''
print("Exploring map")

for node in closed_set:
    canvas[node[1], node[0]] = [0, 255, 0]
    counter +=1
    if counter%500 == 0 or counter == 0:
        canvas_flipped = cv2.flip(canvas,0)
        canvas_flipped_uint8 = cv2.convertScaleAbs(canvas_flipped)
        # cv2.imshow('window',canvas_flipped_uint8)
```

```
        # cv2.waitKey(1)
        video_writer.write(canvas_flipped_uint8)

'''
Loop to mark the path created
'''
print("Backtracking")

for index in path:
    coord=visited[index]
    cv2.circle(canvas, (coord[0],coord[1]), 1, [0,0,0], -1)

    canvas_flipped = cv2.flip(canvas,0)
    canvas_flipped_uint8 = cv2.convertScaleAbs(canvas_flipped)
    # cv2.imshow('window',canvas_flipped_uint8)
    # cv2.waitKey(1)

    video_writer.write(canvas_flipped_uint8)



'''
Loop to add some additional frames at the end of the video
'''
for i in range(150):
    video_writer.write(canvas_flipped_uint8)

print("Video Processed")
# cv2.waitKey(0)
video_writer.release()
# cv2.destroyAllWindows()
end_time = time.time()
print(f"The runtime of my program is {end_time - start_time} seconds.")
```