```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Tue Mar 14 23:56:08 2023

@author: saksham
"""

import numpy as np
import time
import pygame
import matplotlib.pyplot as plt
from sys import exit

## Getting clearance, robot radius and step size inputs

while True:
    try:
        c = input("\nEnter the sum of wall clearance and robot radius : ")
        step = input("\nEnter the step size of robot : ")
        c = int(c)
        step = int(step)
        break

    except KeyboardInterrupt:
        print("\nExiting out of the program\n")
        exit()
    except:
        print("\nWrong values entered, please re-enter\n")
        continue

## Making obstacle map space for point p, clearence 5

# Rectangles
def obs1(p):
    if p[0] > (100-c) and p[0]<(150+c) and p[1]<(100+c):
        return False
    else:
        return True
def obs2(p):
    if p[0] > (100-c) and p[0]<(150+c) and p[1]>(150-c):
        return False
    else:
        return True

# Hexagon
P_act = [[235,162.5],[300,200],[365,162.5],[365,87.5],[300,50],[235,87.5]]

P     = [[300-(75+c)*np.cos(np.pi/6),125+(75+c)*np.sin(np.pi/6)],    [300,200+c],
          [300+(75+c)*np.cos(np.pi/6),125+(75+c)*np.sin(np.pi/6)],
          [300+(75+c)*np.cos(np.pi/6),125-(75+c)*np.sin(np.pi/6)],
          [300,50-c],   [300-(75+c)*np.cos(np.pi/6),125-(75+c)*np.sin(np.pi/6)]]
L1 = np.polyfit([P[0][0],P[1][0]] , [P[0][1],P[1][1]] , 1)
L2 = np.polyfit([P[1][0],P[2][0]] , [P[1][1],P[2][1]] , 1)
L3 = np.polyfit([P[3][0],P[4][0]] , [P[3][1],P[4][1]] , 1)
L4 = np.polyfit([P[4][0],P[5][0]] , [P[4][1],P[5][1]] , 1)

def obs3(p):
    L1_ = p[1] - L1[0]*p[0] - L1[1] <0
    L2_ = p[1] - L2[0]*p[0] - L2[1] <0
    L3_ = p[1] - L3[0]*p[0] - L3[1] >0
    L4_ = p[1] - L4[0]*p[0] - L4[1] >0
    if L1_ and L2_ and L3_ and L4_ and (300+(75+c)*np.cos(np.pi/6)) > p[0] >
```

```
(300-(75+c)*np.cos(np.pi/6)):
        return False
    else:
        return True

# Triangle
Q_act = [[460,225],[510,125],[460,25]]
Q     = [[460-c, 125 + ( 2*(50+c+(c/np.cos(0.46364761))) )],
         [510+c/np.cos(0.46364761),125],
         [460-c, 125 - ( 2*(50+c+(c/np.cos(0.46364761))) )]]
Q1 = np.polyfit([Q[0][0],Q[1][0]] , [Q[0][1],Q[1][1]] , 1)
Q2 = np.polyfit([Q[1][0],Q[2][0]] , [Q[1][1],Q[2][1]] , 1)

def obs4(p):
    Q1_ = p[1] - Q1[0]*p[0] - Q1[1] <0
    Q2_ = p[1] - Q2[0]*p[0] - Q2[1] >0
    if Q1_ and Q2_ and p[0]>(460-c) and (225+c)>=p[1]>=(25-c):
        return False
    else:
        return True
#%%
# Checker if a node falls in the obstacle space
def checkFeasibility(node):
    if obs1(node) and obs2(node) and obs3(node) and obs4(node):
        if node[0]>=c and node[0]<=600-c and node[1]>=c and node[1]<=250-c:
            return True
        else:
            return False
    else:
        return False
#%%
shifter = [60,30,0,-30,-60]

def costC(node,goal):
    d = np.sqrt((node[0]-goal[0])**2 + (node[1]-goal[1])**2)
    return d

def shift(node,cost,step):
    for i in range(len(shifter)):
        childNode = (node[0] + step * np.cos(np.deg2rad( shifter[i] + node[2])) ,
                     node[1] + step * np.sin(np.deg2rad( shifter[i] + node[2])) ,
                     node[2] + shifter[i])
        if checkFeasibility(childNode):
            yield childNode, cost+1

#%%
def closest_Node(node,dikt):
    nodesDist = np.array(list(dikt.keys())) - node
    nodesMin = np.sum(np.square(nodesDist[:,0:2])).argmin()

    Cnode = list(dikt.keys())[nodesMin]
    dist = costC(Cnode,node)

    return Cnode,dist

#%%
# Main algorithm

def astar(startState,goalState,step):
    step+=1
    # time calculation
    startTime = time.time()
```

```
    if not checkFeasibility(startState) or not checkFeasibility(goalState):
        print('Infeasible states! Check Input')
        return None, None, None, None

    closedNodes = {}
    openNodes = {startState:( costC(startState,goalState) , costC(startState,goalState) ,0,0,0)}
    # order is totalCost, cost2Goal, cost2come, parent, self
    nodeVisit = 255*np.ones((600,250))

    child = 1
    repeatSkip=0

    while True:

        # popping first node
        parent=list(openNodes.keys())[0]

        closedNodes[parent] = openNodes[parent]

        if costC(parent,goalState) < step/2:
            print("Goal Found after",len(closedNodes),"nodes in ",time.time()-startTime, "
seconds!\n")
            print("overwrote nodes :",repeatSkip)
            break

        for node,cost in shift(parent,openNodes[parent][2],step):

            if nodeVisit[round(node[0]),round(node[1])]==125:

                # If node in open nodes, update cost ........
                node_C,dist = closest_Node(node,openNodes)
                if cost < openNodes[node_C][2]:
                    repeatSkip=repeatSkip+1
                    openNodes[node_C] = (cost+openNodes[node_C][1] , openNodes[node_C][1],
                             cost, openNodes[parent][4],openNodes[node][4])
                pass

            else:

                if nodeVisit[round(node[0]),round(node[1])] == 255 and node != None:
                    # ...... and if not, add child
                    openNodes[node] = (costC(node,goalState) + cost,
                             costC(node,goalState),
                             cost,openNodes[parent][4],child)
                    child = child + 1
                    nodeVisit[round(node[0]),round(node[1])]=125

        nodeVisit[round(parent[0]),round(parent[1])] = 0
        del openNodes[parent]

        # Sort the dict before popping
        openNodes = dict(sorted(openNodes.items(), key=lambda x:x[1]))

    # backtracking
    backTrack = [node,parent]
    child = closedNodes[parent][3]
    while child >0:
        for key, value in closedNodes.items():

            if value[4] == child:
                node = key
                child = value[3]
                backTrack.append(node)
```

```
    backTrack.append(startState)
    backTrack.reverse()

    return backTrack,closedNodes,openNodes,nodeVisit
#%%

## Getting start and goal inputs, checking feasibility and running A-Star

while True:
    try:
        start = input("\nEnter start X and Y coordinates, and angle, separated by commas : ")
        goal  = input("\nEnter goal X and Y coordinates, and angle, separated by commas : ")
        print('\n')

        start = tuple(map(int, start.split(",")))
        goal = tuple(map(int, goal.split(",")))

        backTrack,closedNodes,openNodes,nodeVisit = astar(start,goal,step)

        if backTrack is not None:
            break

    except KeyboardInterrupt:
        print("\nKeyboard interrupt detected, exiting out of the program\n")
        exit()
    except IndexError as e:
        print("\r\nNo solution found\r\n")
        exit()
    except:
        print("\nError in values, please re-enter\n")
        continue

#%%

pygame.init()
screen = pygame.display.set_mode([1800, 750])

imgID = 0

running = True
clock = pygame.time.Clock()

while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

    screen.fill((255, 255, 255))

    pygame.draw.rect(screen, (200,200,0), pygame.Rect((100-c)*3, (150-c)*3, (50+c+c)*3,
(100+c)*3)) #dist from left, top, w,h
    pygame.draw.rect(screen, (200,200,0), pygame.Rect((100-c)*3, 0*3, (50+c+c)*3, (100+c)*3))
    pygame.draw.polygon(screen,(200,200,0), (np.array(Q)*3).tolist())
    pygame.draw.polygon(screen,(200,200,0), (np.array(P)*3).tolist())

    pygame.draw.rect(screen, (0,0,0), pygame.Rect(100*3, 150*3, 50*3, 100*3)) #dist from left,
top, w,h
    pygame.draw.rect(screen, (0,0,0), pygame.Rect(100*3, 0*3, 50*3, 100*3))
    pygame.draw.polygon(screen,(0,0,0), (np.array(Q_act)*3).tolist())
    pygame.draw.polygon(screen,(0,0,0), (np.array(P_act)*3).tolist())

    pygame.draw.rect(screen, (255,255,255), pygame.Rect(400*3, c*3, 100*3, (25-c-c)*3)) #dist
```

```
from left, top, w,h
    pygame.draw.rect(screen, (255,255,255), pygame.Rect(400*3, (225+c)*3, 100*3, (25-c-c)*3))

    pygame.draw.rect(screen, (200,200,0), pygame.Rect(0*3, 0*3, 600*3, (c)*3)) #dist from left,
top, w,h
    pygame.draw.rect(screen, (200,200,0), pygame.Rect(0*3, 0*3, c*3, (250)*3))
    pygame.draw.rect(screen, (200,200,0), pygame.Rect((600-c)*3, 0*3, c*3, (250)*3)) #dist from
left, top, w,h
    pygame.draw.rect(screen, (200,200,0), pygame.Rect(0*3, (250-c)*3, 600*3, (c)*3))

    pygame.draw.rect(screen, (0,0,200), pygame.Rect(start[0]*3, 750-start[1]*3, 4*3, 4*3))
    pygame.draw.rect(screen, (0,0,200), pygame.Rect(goal[0]*3, 750-goal[1]*3, 4*3, 4*3))

    for node in closedNodes:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                running = False
                break
        if not running: break

        pygame.draw.line(screen, (0,100,0), (3*node[0],750-3*node[1]),
                         ( 3* (node[0]+step*np.cos(np.deg2rad(node[2]))) ,
                          750 - 3*(node[1]+step*np.sin(np.deg2rad(node[2]))) ) )

        pygame.draw.circle(screen, (0,100,0), (3*node[0],750-3*node[1]), 2, width=0)

        clock.tick(200)
        pygame.display.update()

        name = 'Image'+str(imgID).zfill(8)+'.png'
        if imgID%1 == 0:
            #pygame.image.save(screen, name)
            pass
        imgID=imgID+1

    for i in range(len(backTrack)-2):
        node=backTrack[i]
        nodeN = backTrack[i+1]
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                running = False
                break
        if not running: break

        pygame.draw.line(screen, (200,0,0), (3*node[0],750-3*node[1]),
(3*nodeN[0],750-3*nodeN[1]), width=2)
        pygame.draw.circle(screen, (200,0,0), (3*nodeN[0],750-3*nodeN[1]), 3, width=0)
        clock.tick(50)
        pygame.display.update()

        name = 'Image'+str(imgID).zfill(8)+'.png'
        if imgID%1 == 0:
            #pygame.image.save(screen, name)
            pass
        imgID=imgID+1

    while running:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                running = False
                pygame.quit()
pygame.quit()
```