Potential Field Algorithm - Assignment 3

Himanshu Kishor Choubey
190376

April 2022

1 Python Code using Pygame Library

```
#Note commented code was used for debugging and 2nd part of the problem
#(changing obstacles for local minima trap)


import sys,pygame
import os
import time
import math
import random
import numpy as np
import matplotlib.pyplot as plt
from random import randint as ri

def sqrt(n):
        return math.sqrt(n)

pygame.font.init()
pygame.init()

WIDTH, HEIGHT = 900, 500  #window size

gui_font = pygame.font.Font(None, 30)

WIN = pygame.display.set_mode((WIDTH, HEIGHT))

pygame.display.set_caption("Potential Method")

BORDER = pygame.Rect(445, 0, 10, HEIGHT)

#defined colors
```

```python
WHITE = (255,255,255)

BLACK = (0,0,0)

BLUE = (0,0,255)

RED = (255,0,0)

YELLOW = (255, 255, 0)

GREY = (128,128,128)

FPS = 30

#defined constants for attractive, repulsive forces and step constants

A = 1 #step constant
C = 0.01 #attractive constant
D = 8000000 #repulsive constant


iters = 0
#start point

x = 45
y = 45

#goal point

x_goal = 830
y_goal = 380

WIN.fill(WHITE)



#returns attractive potential due to goal point

def attraction(x,y,x_goal, y_goal, C):
        d = 200
        if((x-x_goal)**2 + (y-y_goal)**2 <= d**2):
                U = 0.5*C*((x-x_goal)**2 + (y-y_goal)**2)
        else:
                U = d*C*sqrt(((x-x_goal)**2 + (y-y_goal)**2)) - 0.5*C*(d**2)

        return U
```

```python
#returns attractive force due to goal point
def att_force(x,y,x_goal, y_goal, C):
        d = 200
        if((x-x_goal)**2 + (y-y_goal)**2 <= d**2):
                del_U_x = C*(x-x_goal)
                del_U_y = C*(y-y_goal)
        else:
                del_U_y = d*C*(y-y_goal)/sqrt(((x-x_goal)**2 + (y-y_goal)**2))
                del_U_x = d*C*(x-x_goal)/sqrt(((x-x_goal)**2 + (y-y_goal)**2))
        return (del_U_x, del_U_y)



#returns repulsive potential due to obstacles
def repulsion():
        Q_star = 70
        U = 0
        if((x-450)**2 + (y-100)**2 <= Q_star**2):
                U = 0.5*D*(1/(sqrt((x-450)**2 + (y-100)**2 )) - 1/(Q_star))**2
        else:
                U = 0

        d = sqrt((x-350)**2 + (y-350)**2)

        if(d<=100):
                U += 0.5*D*(1/d - 1/100)**2

        else:
                U+=0

        d = sqrt((x-200)**2 + (y-175)**2)

        if(d<200):
                U += 0.5*D*(1/d - 1/200)**2

        d = sqrt((x-650)**2 + (y-300)**2)

        if(d<150):
                U += 0.5*D*(1/d - 1/150)**2

        # d = sqrt((x-390)**2 + (y-215)**2)

        # if(d<100):
        #         U += 0.5*D*(1/d - 1/100)**2
```

```
        d = sqrt((x-380)**2 + (y-220)**2)

        if(d<90):
                U += 0.5*D*(1/d - 1/90)**2

        return U


#returns repulsive force due to obstacles
def rep_force(x,y,x_goal, y_goal, D):
        del_U_x = 0
        del_U_y = 0

        d = sqrt((x-450)**2 + (y-100)**2)

        if(d<=70):
                del_U_x+=D*(1/70 - 1/d)*(1/(d**2))*((x-450)/d)
                del_U_y+=D*(1/70 - 1/d)*(1/(d**2))*((y-100)/d)

        d = sqrt((x-350)**2 + (y-350)**2)

        if(d<=100):
                del_U_x += D*(1/100 - 1/d)*(1/(d**2))*((x-350)/d)
                del_U_y += D*(1/100 - 1/d)*(1/(d**2))*((y-350)/d)

        d = sqrt((x-200)**2 + (y-175)**2)

        if(d<=200):
                del_U_x += D*(1/200 - 1/d)*(1/(d**2))*((x-200)/d)
                del_U_y += D*(1/200 - 1/d)*(1/(d**2))*((y-175)/d)

        d = sqrt((x-650)**2 + (y-300)**2)

        if(d<150):
                del_U_x += D*(1/150 - 1/d)*(1/(d**2))*((x-650)/d)
                del_U_y += D*(1/150 - 1/d)*(1/(d**2))*((y-300)/d)

        d = sqrt((x-380)**2 + (y-220)**2)

        if(d<90):
                del_U_x += D*(1/90 - 1/d)*(1/(d**2))*((x-380)/d)
                del_U_y += D*(1/90 - 1/d)*(1/(d**2))*((y-220)/d)

        # d = sqrt((x-390)**2 + (y-215)**2)
```

```python
        # if(d<100):
        #         del_U_x += D*(1/100 - 1/d)*(1/(d**2))*((x-390)/d)
        #         del_U_y += D*(1/100 - 1/d)*(1/(d**2))*((y-215)/d)


        # print(del_U_x, del_U_y)

        return (del_U_x, del_U_y)



#finds the path here

def start_path(x,y,iters):
        del_U1 = rep_force(x,y,x_goal,y_goal,D)
        del_U2 = att_force(x,y,x_goal,y_goal,C)

        plx = del_U1[0]+del_U2[0]
        ply = del_U1[1]+del_U2[1]

        xnew = x-A*(plx)
        ynew = y-A*(ply)

        # print(xnew,ynew)

        # print(plx, ply)

        pygame.draw.line(WIN, BLUE, (x, y), (xnew, ynew))



        if (x_goal-x <=5 and x_goal-x>=-5) and (y_goal-y <=5 and y_goal-y>=-5):
                return

        pygame.display.update()
        iters+=1



        start_path(xnew, ynew ,iters)




def draw_window():
        pygame.draw.rect(WIN,BLACK,(25,25,825,400),5)
```

```python
        pygame.draw.rect(WIN,BLACK,(300,300,100,100))

        pygame.draw.circle(WIN, BLACK, (450,100), 50)

        pygame.draw.rect(WIN,BLACK,(100,100,200,150))

        pygame.draw.circle(WIN, BLACK, (380,220), 70)##

        # pygame.draw.circle(WIN, BLACK, (390,215), 80)##

        pygame.draw.circle(WIN, BLACK, (650,300), 100)

        pygame.draw.rect(WIN,YELLOW,(40,40,10,10))

        pygame.draw.rect(WIN,RED,(825,375,10,10))

        pygame.display.update()

def main():
        level = 1
        run = True
        clock = pygame.time.Clock()
        press = False
        b_color = GREY

        while(run):
                clock.tick(FPS)


                for event in pygame.event.get():

                        if event.type == pygame.QUIT:
                                run = False
                                pygame.quit()
                        if run==False:
                                pygame.quit()
                                break
                        if event.type == pygame.KEYDOWN:
                                if event.key == pygame.K_RETURN:
                                        start_path(x,y, iters)
                                if event.key == pygame.K_LCTRL:
                                        plots()
```

```
                draw_window()

        main()

if __name__=="__main__":
    main()
```

2 Cases for Successful Path Planning

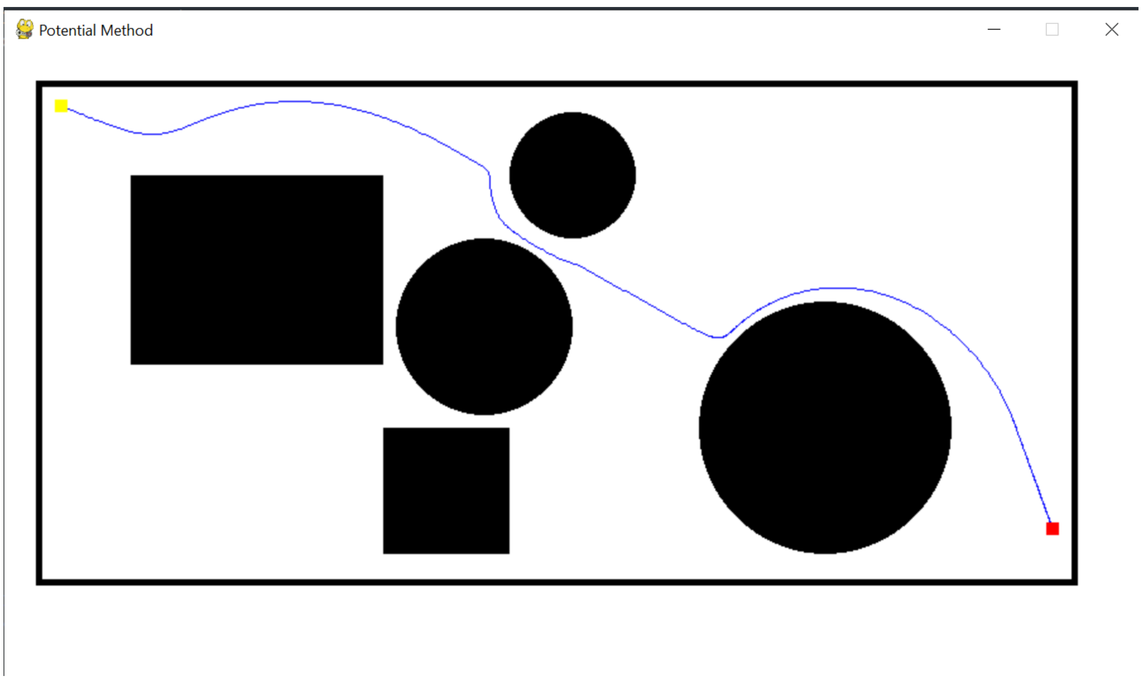

Figure 1: Path for
A = 1 #step constant
C = 0.01 #attractive constant
D = 8000000 #repulsive constant

3 Case for Local Minima Trap

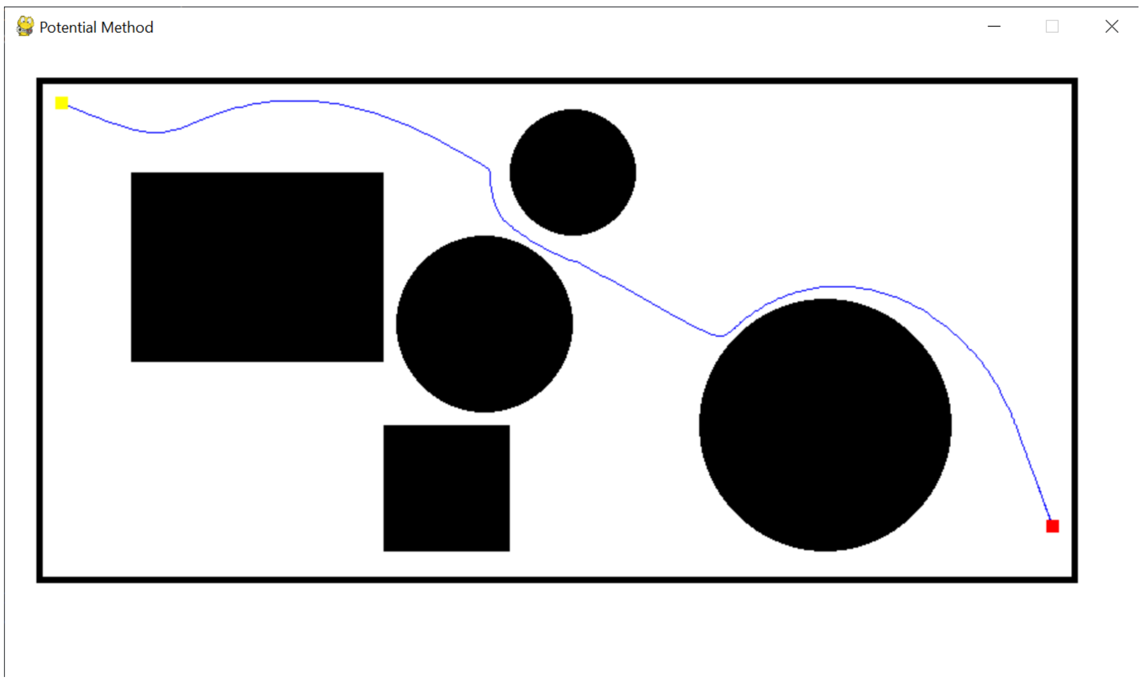

Figure 2: Path for
A = 1 #step constant
C = 0.02 #attractive constant
D = 15000000 #repulsive constant

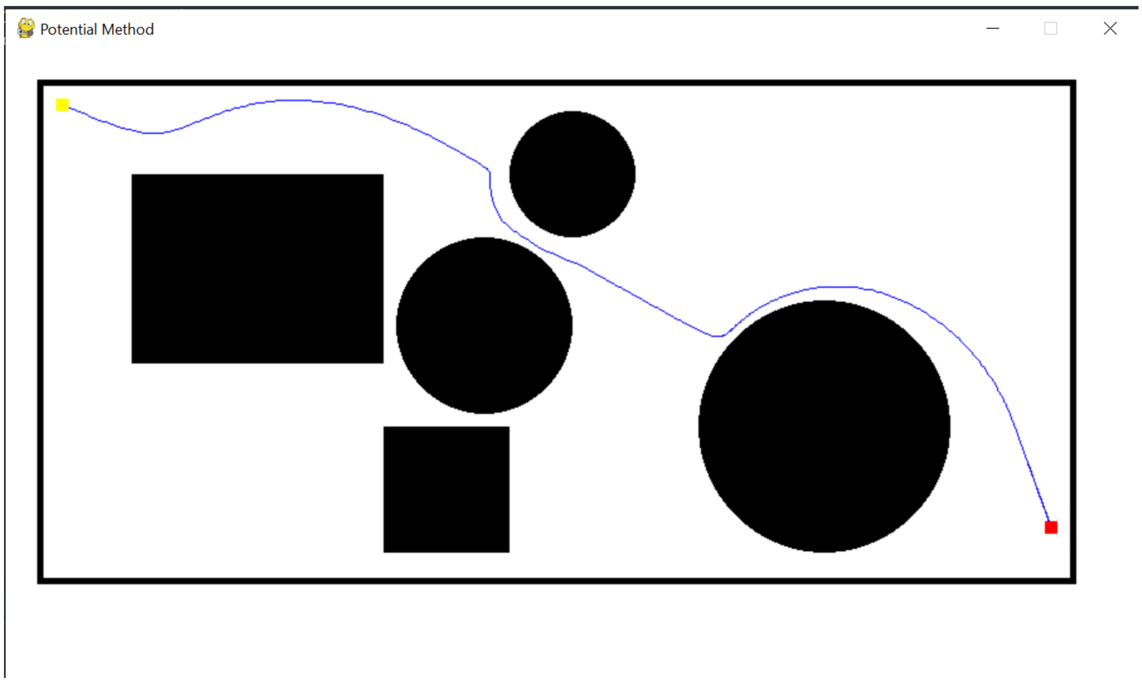

Figure 3: Path for
A = 1 #step constant
C = 0.015 #attractive constant
D = 12000000 #repulsive constant

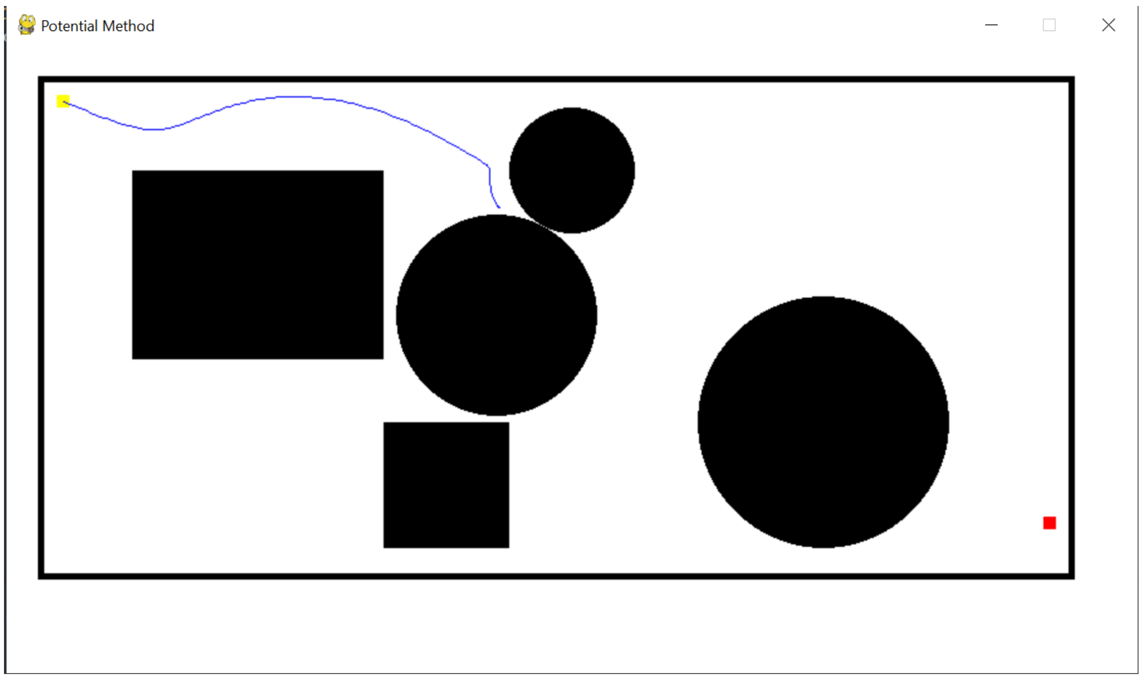

Figure 4: Path for
A = 1 #step constant
C = 0.015 #attractive constant
D = 12000000 #repulsive constant