# CS512 - Assignment 2

Mohammadreza Asherloo
Department of Mechanical, Materials and Aerospace Engineering
Illinois Institute of Technology

October 22, 2020

## Problem Statement:

Implementation of greedy active contour

## Solution:

The algorithm used is as follows:

1. Take initial guess points from user
2. Determine the average distance between points.
3. Compute the E
4. Move each point and compute the new E and if the new E is smaller that the old E, save the new position for that point
5. Repeat the steps while the E is decreasing

## Details:

We open the image and convert it to grayscale and after that to binary:

```
winName = "Image"
image = cv.imread("test.jpg")
gray = cv.cvtColor(image, cv.COLOR_RGB2GRAY)
ret, binary = cv.threshold(gray,127,255,cv.THRESH_BINARY)
```

We create a window and put the trackbars for controlling the variables:

```
cv.namedWindow(winName)
cv.setMouseCallback(winName,mouse_pos)
cv.createTrackbar("Alpha", winName, 0, 10, nothing)
cv.createTrackbar("Beta", winName, 0, 10, nothing)
cv.createTrackbar("Gamma", winName, 0, 10, nothing)
```

```
    cv.createTrackbar("Iteration", winName, 0, 300, nothing)
    cv.createTrackbar("Delay(ms)", winName, 0, 1000, nothing)
```

Then we compute the average distance between the points:

```
t = 0
for i in range(-1, len(points)-1):
    t += np.sqrt((points[i+1][0]-points[i][0])**2 + (points[i+1][1]-points[i][1])**2)
    d = t/len(points)
```

Then we need to compute the initial E:

```
for i in range(-1, len(points)-1):
    Econt[i] += ((np.sqrt((points[i+1][0] - points[i][0])**2 +
    (points[i+1][1] - points[i][1])**2))**2)*alpha
    Ecur[i] += ((points[i+1][0] - 2*points[i][0] + points[i-1][0])**2 +
    (points[i+1][1] - 2*points[i][1] + points[i-1][1])**2)*beta
    Eimg[i] += (gradient[points[i][1]][points[i][0]]**2)*gamma
E = sum(Econt) + sum(Ecur) - sum(Eimg)
```

Then we need to move the points one by one and compute the new E:

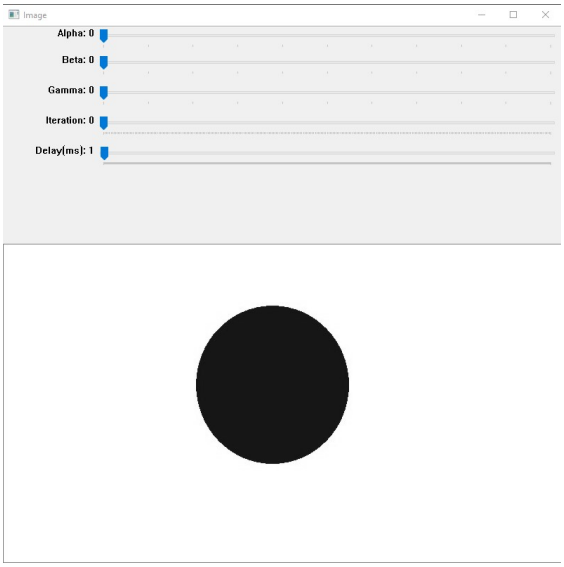
```
def EnCal(i, lenPoints):
    global points, d
    En = np.zeros((9, 3))
    g = 0
    for j in [-1, 0, 1]:
        for k in [-1, 0, 1]:
            En_cont = [0] * lenPoints
            En_cur = [0] * lenPoints
            En_img = [0] * lenPoints
            temp = points
            temp[i][0] += j
            temp[i][1] += k
            En[g][0] = j
            En[g][1] = k
            for h in range(-1, lenPoints-1):
                En_cont[h] += ((np.sqrt((temp[h+1][0] - temp[h][0])**2 + (temp[h+1][1] -
                En_cur[h] += ((temp[h+1][0] - 2*(temp[h][0]) + temp[h-1][0])**2 + (temp[
                En_img[h] += (gradient[temp[h][1]][temp[h][0]]**2)*gamma
            En[g][2] = sum(En_cont) + sum(En_cur) - sum(En_img)
            g += 1
    return En

for i in range(-1, len(points)-1):
    En = EnCal(i, len(points))
    En = En[np.argsort(En[:,2])]
```

Then we check if the new E is smaller than the old E and if it is, save the new location of that point:

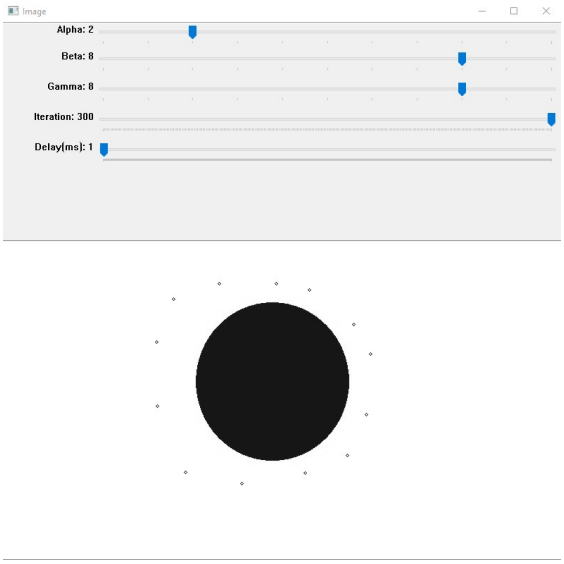
```
if En[0][2] < E:
    points[i][0] += int(En[0][0])
    points[i][1]+= int(En[0][1])
```

## Results and discussion:

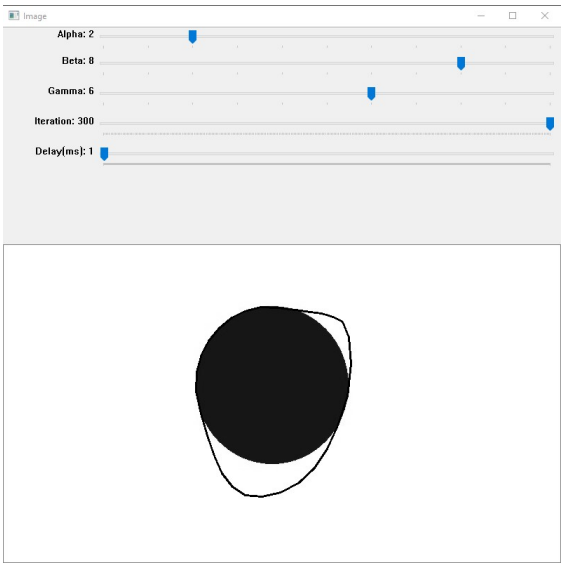
This is the original image without any noise:



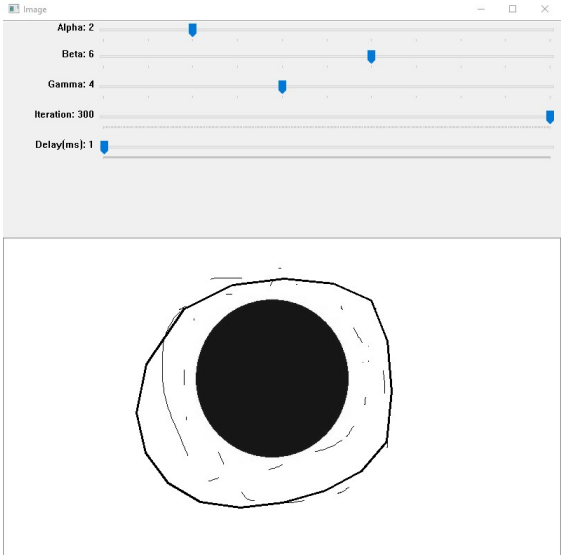

This is the image before implementing the active contour algorithm:




This is the image after implementing the active contour algorithm:




And this is the image after implementing the active contour algorithm on an image with noises:




There are some weaknesses in the implemented algorithm:

1. The user can't put the points randomly because the program connects points in an ordered manner
2. The calculations speed is slow so I needed to reduce the step size and as a result, reaching the convergence takes more iterations.
3. The algorithm is prone to noises and if there are noises in the image, gradient of image will change and has a significant effect on the convergence.

There are some strength in the implemented algorithm:

1. The user can't put the points randomly because the program connects points in an

ordered manner

2. The calculations speed is slow so I needed to reduce the step size and as a result, reaching the convergence takes more iterations.
3. The algorithm is prone to noises and if there are noises in the image, gradient of image will change and has a significant effect on the convergence.