الجلسة التاسعة

Object Detection (Cirular Hough Transform, Region Growing, Active Contouring)

1.1 Circular Hough Transform (Circle Object Detection)

نفذ الكود التالي:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
# Read the image
image = cv2.imread("iris.jpg", 1)
# Convert to grayscale
gray = cv2.cvtColor(image, cv2. COLOR_BGR2GRAY)
# Apply Gaussian blur to reduce noise
gray_blur = cv2.GaussianBlur(gray, (5, 5), 2, 2)
# Detect circles using Hough Circle Transform
circles = cv2.HoughCircles(
  gray_blur,
  cv2.HOUGH_GRADIENT,
 dp=1,
  minDist=gray.shape[0] // 16,
  param1=50,
  param2=30,
  minRadius=75,
  maxRadius=100,  # Adjust these parameters for larger circles
)
if circles is not None:
  circles = np.uint16(np.around(circles))
  for circle in circles[0]:
```

بداية يتم تطبيق مرشح غوص لتنعيم التغيرات البسيطة في الصورة ويمكن إهمال هذه المرحلة في حال كانت الصورة لا تتضمن ضجيج أو في حال لم نرد حذف الحواف الضعيفة

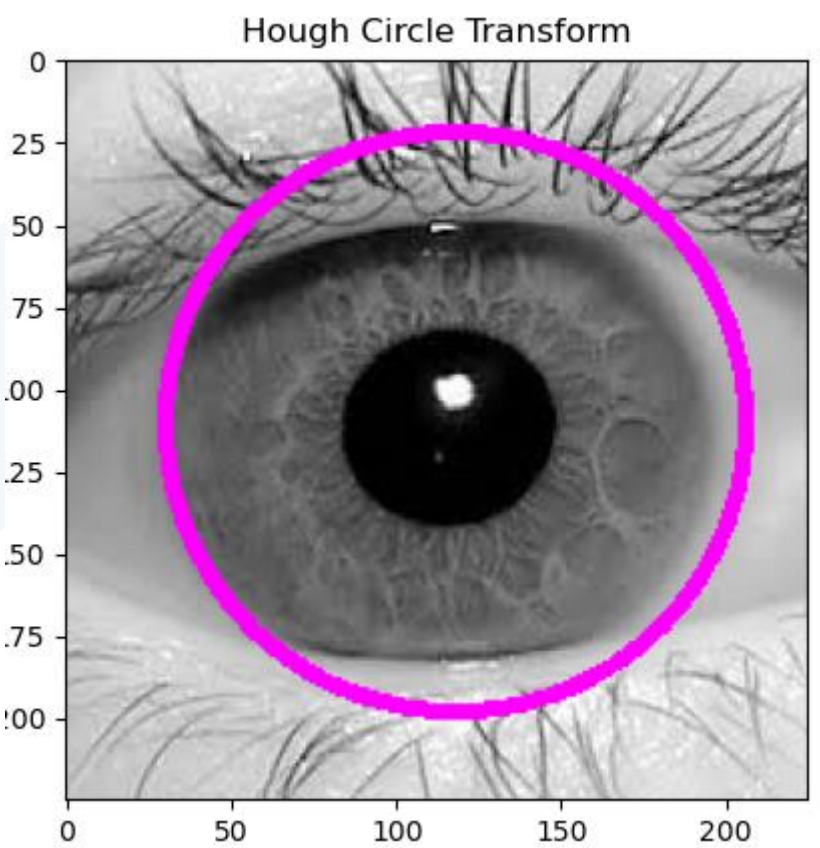
تطبيق تحويل هاف بالبارامترات التالية:
الصورة
طريقة كشف الدوائر وهي طريقة المشتقات cv2.HOUGH_GRADIENT،
minDist وهي المسافة الأقصر بين مراكز الدوائر المسموح بها،
Param1 العتبة العليا لكشف احواف في الصورة ،
Param2 عتبة مصفوفة المراكم المسموح بها وتصغير هذه العتبة يعني قبول عدد دوائر كثير وتكبيرها يقلل عدد الدوائر الناتجة
miRadius, maxRadius أقل قيمة وأعلى قيمة نصف قطر للدوائر المقبولة

تعليمات رسم الدوائر:
من أجل كل الدوائر التي تم كشفها:
احصل على مركز الدائرة ونصف قطرها
ثم استخدم تابع cv2.circle لرسم دائرة فوق الصورة image بلون Magenta وبحجم 3 لخط الرسم وباستخدام إحداثيات مركز الدائرة ونصف قطرها.

```
    center = (circle[0], circle[1])
    radius = circle[2]
    cv2.circle(image, center, radius, (255, 0, 255), 3)  # Draw the circle
# Display the result
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.title("Hough Circle Transform")
plt.show()
```

والنتيجة باستخدام هذه المعاملات:

```
param1=50,
  param2=30,
  minRadius=75,
  maxRadius=100,
```



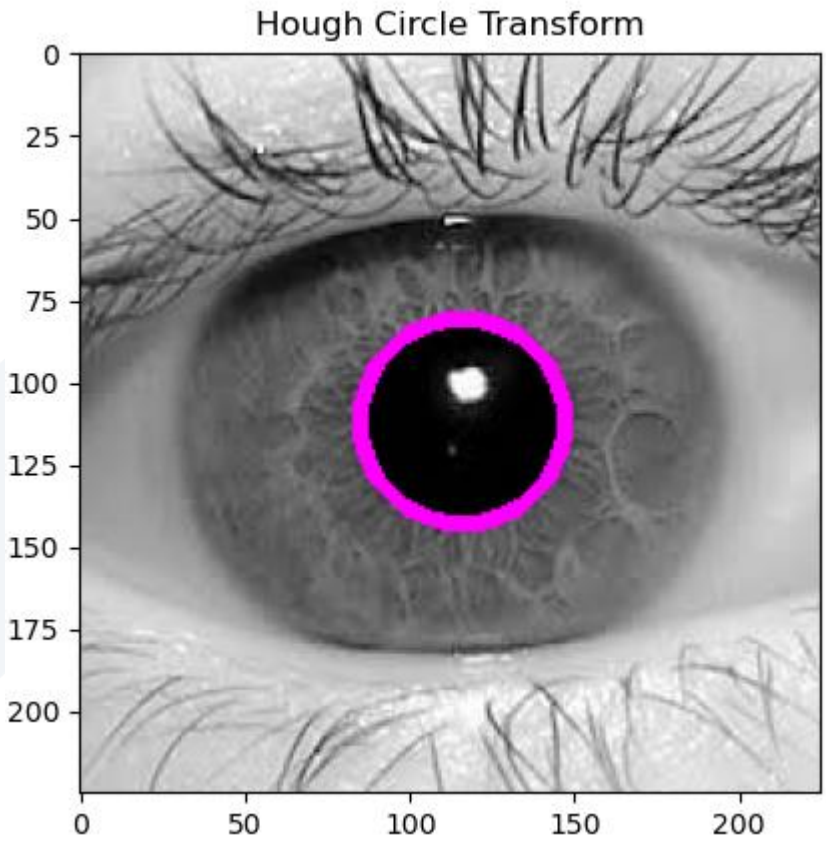
بتغيير حجم القطر الممكن البحث ضمنه نحصل على الدائرة الداخلية

```
param1=50,
```

param2=30,

minRadius=25,

maxRadius=50



1.2 Region Growing Algorithm

تتضمن الخطوات الآتية:

1- اختيار بكسل أو عدة بكسلات بذرة وهي عبارة عن الإحداثيات الممثلة لمراكز المناطق المهمة في الصورة (ليس بالضرورة اختيار مركز المنطقة وإنما أي بكسل ضمن المنطقة المرغوبة).

2- البدء بضم البكسلات المجاورة للبكسل المدروس بحساب التشابه بين البكسل ومجاوراته وبالتالي تتوسع المناطق تكرار بعد تكرار.

3- نستمر بتكرار الخطوة 2 حتى نحصل على تكرارين متتالين متطابقين لا يحدث فيهما أي ضم جديد لأي بكسلات.

ملاحظة: لا تتضمن هذه الخوارزمية عدد تكرارات محدد وإنما شرط توقف وهو عدم وجود بكسلات جديدة لضمها أو انتهاء دراسة كل البكسلات في الصورة.

أثناء عملية المقارنة بين البكسل ومجاوراته يتم حساب الفرق ومقارنته مع عتبة معينة فإذا تم تجاوز قيمة العتبة لا يضم البكسل.

نفذ الكود:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from skimage.color import rgb2gray
from skimage.io import imread
from matplotlib.path import Path

def manual_seed_selection(image, num_seeds):
    fig, ax = plt.subplots(figsize=(7, 7))
    ax.imshow(image, cmap=plt.cm.gray)
    ax.set_xticks([]), ax.set_yticks([])
    plt.title("Select seed points. Press Enter when done.")
    # Use ginput to get seed points
    seeds = plt.ginput(num_seeds)
    plt.show()
    return seeds
```

تابع لالتقاط بكسلات البذرة من خلال عرض الصورة وترك المستخدم يختار عدد بكسلات بذرة في المناطق التي يرغب بها (العدد محدد بالبارامتر num_seeds)

```
def region_growing(img, seeds, threshold):
    # parameters
    neighbors = [(0,1), (1,0), (-1,0), (0,-1)]  # 4-connectivity

    height, width = img.shape
    visited = np.zeros_like(img, dtype=np.uint8)
    out_img = np.zeros_like(img, dtype=np.uint8)
```

خوارزمية Region Growing

Threshold هي قيمة العتبة ويمكن تغييرها

Seed هي إحداثيات بكسلات البذرة

Img الصورة الأصلية

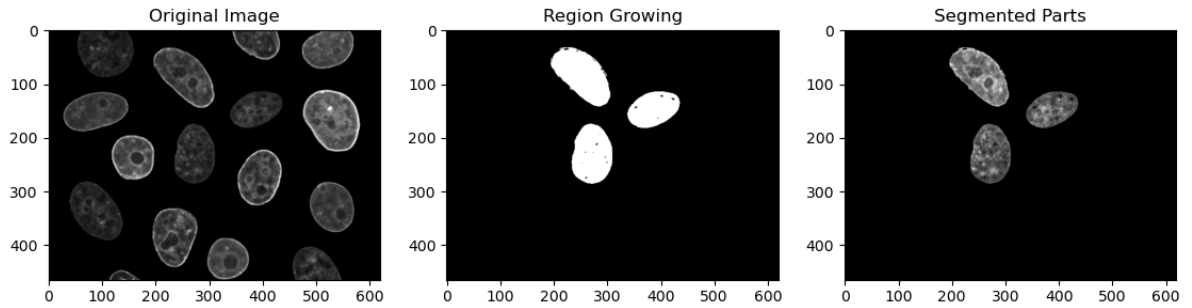
```
# stack of pixels
stack = []
for seed in seeds:
  stack.append((int(seed[1]), int(seed[0])))  # note the y,x order because ginput returns x,y

while(len(stack) > 0):
  s = stack.pop()
  x, y = s

  # Convert the seed coordinates to integers
  seed_x, seed_y = int(seeds[0][1]), int(seeds[0][0])

  if(np.abs(int(img[x, y]) - int (img[seed_x, seed_y])) <= threshold):
    out_img[x, y] = 255
    visited[x, y] = 1

    # add neighbors to stack
    for dx, dy in neighbors:
      nx, ny = x + dx, y + dy
      if nx >= 0 and nx < height and ny >= 0 and ny < width:
        if visited[nx, ny] == 0:
          stack.append((nx, ny))

return out_img, visited
```

طرح البكسل من مجاوراته فإن كانت القيمة أقل من العتبة يقبل ويضم للمنطقة وتضاف إحداثياته لمصففوفة البكسلات التي تمت زيارتها visited

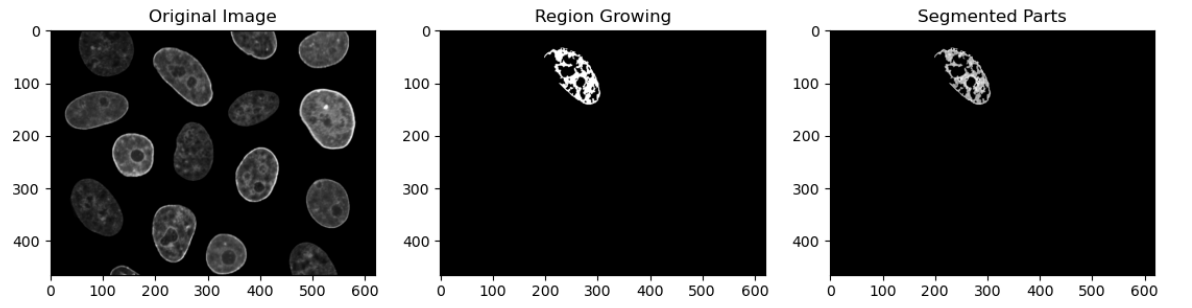
```
# Load your cell image (replace 'cell.jpg' with your actual image path)
img = cv2.imread('cell.jpg',  cv2.IMREAD_GRAYSCALE)

# Manually select the seed points
seeds = manual_seed_selection(img, 10)

# Apply region growing algorithm
out_img, mask = region_growing(img, seeds, 50)

# Extract the segmented parts from the original image
segmented = np.where(mask, img, 0)

# Display the results
plt.figure(figsize=(10, 10))

plt.subplot(1, 3, 1)
plt.imshow(img, cmap=plt.cm.gray)
plt.title("Original Image")

plt.subplot(1, 3, 2)
plt.imshow(out_img, cmap=plt.cm.gray)
plt.title("Region Growing")

plt.subplot(1, 3, 3)
plt.imshow(segmented, cmap=plt.cm.gray)
plt.title("Segmented Parts")

plt.show()
```

نحمل صورة اختبار ونمرر الصورة لتابع اختيار البذرات seed pixels وبعد ذلك نمرر الصورة مع seed pixels مع العتبة لتابع region_growing ونحصل منه على صورة القناع mask التي تتضمن 1 (أبيض) في المناطق التي تم ضمها و 0 (أسود) في باقي المناطق

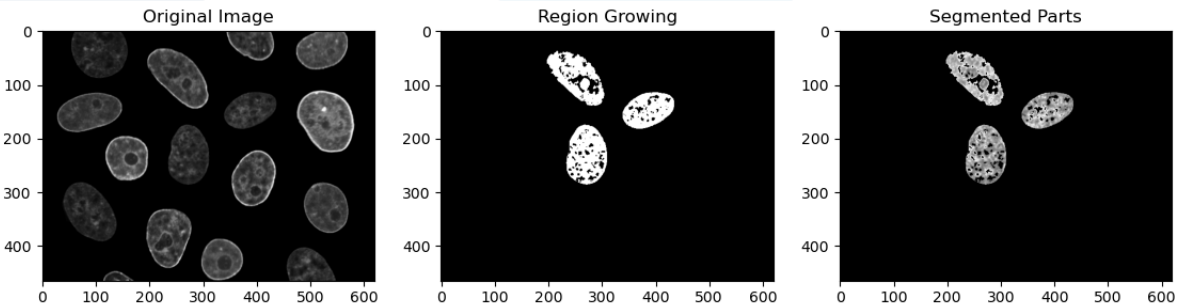
لنختبر الخوارزمية بداية باستخدام 3 بكسلات seed pixels وباستخدام عتبة مقارنة 50.



نكرر نفس الاختبار لكن باختيار عتبة 20

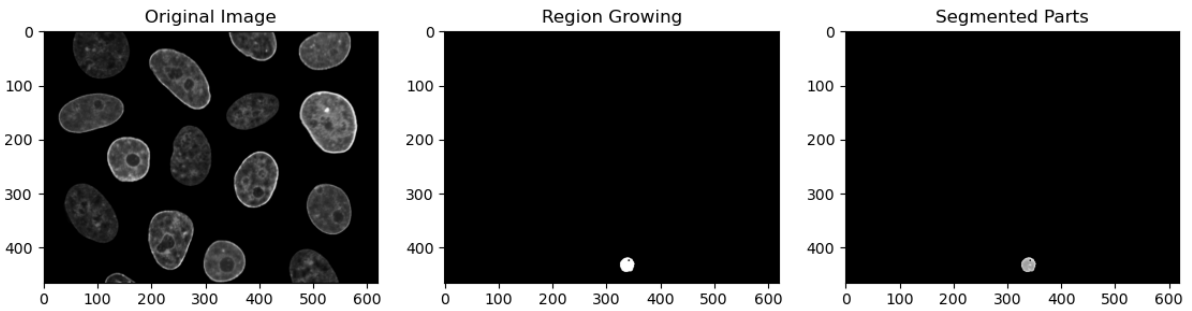


لنجرب نفس العتبة لكن مع اختيار 6 بكسلات ضمن نفس المناطق.

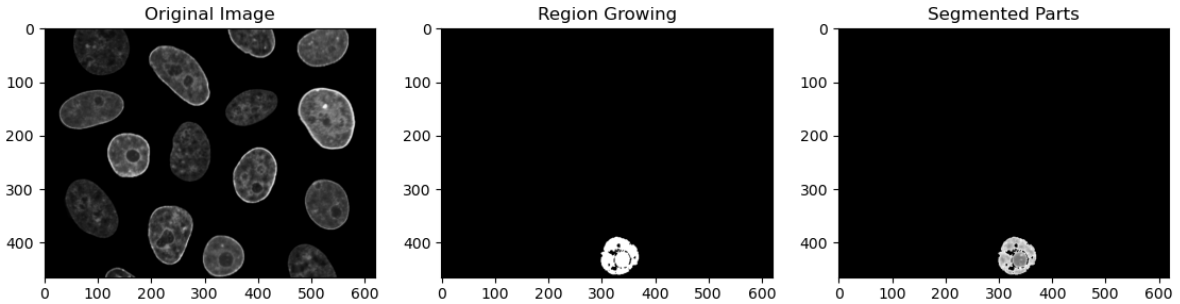


تغيير العتبة وتغيير عدد بكسلات Seed وتغيير مواقعها يغير أداء خوارزمية Region Growing.

مثلاً في المثال الآتي نريد اكتشاف مركز الخلية الظاهر باللون الداكن سنقوم بتحديد عدد seed pixels ببكسل واحد فقط والعتبة 20 ونختار بكسل seed ضمن مركز الخلية. سنلاحظ أن ناتج الكشف كان فقط منطقة مركز الخلية.



لنرفع العتبة إلى 40:



1.3 خوارزمية Active Contouring AC

تختلف عن Region Growing في الآتي:

1- تمتلك عدد تكرارات محدد تعمل وفقاً له.
2- تبدأ بحد خارجي محيط بالمنطقة المرغوبة وتتقلص Shrinking حتى الوصول لحدود المنطقة المهمة حيث تأخذ شكلها تماماً.
3- يمكن جعل التوقف يعتمد إما على التكرارات أو على الشرط: الوصول لتكرار لا يمكن بعده تغيير الحد.

إذا تعمل الخوارزمية من الخارج للداخل Shrinking في حين تعمل regionGrowing من الداخل للخارج Growing.

نفذ الكود:

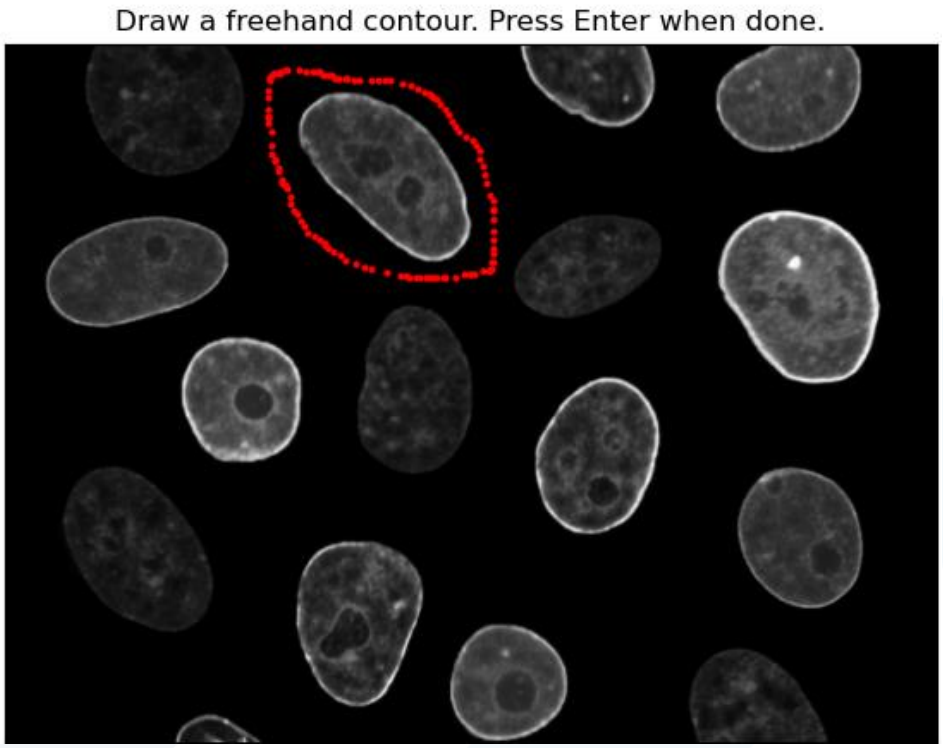
```
import numpy as np
import matplotlib.pyplot as plt
from skimage.color import rgb2gray
from skimage.filters import gaussian
from skimage.segmentation import active_contour
from skimage.io import imread
from matplotlib.path import Path


def manual_freehand_contour_selection(image):
    fig, ax = plt.subplots(figsize=(7, 7))
    ax.imshow(image, cmap=plt.cm.gray)
    ax.set_xticks([]), ax.set_yticks([])
    plt.title("Draw a freehand contour. Press Enter when done.")
```

```
# Initialize an empty list to store user-defined points
points = []

def ondrag(event):
    if event.button == 1:
        points.append((event.ydata, event.xdata))
        ax.plot(event.xdata, event.ydata, 'ro', markersize=2)
        fig.canvas.draw()

fig.canvas.mpl_connect('motion_notify_event', ondrag)
plt.show()

return np.array(points)

# Load your cell image (replace 'cell.jpg' with your actual image path)
img = imread('cell.jpg', as_gray=True)

# Manually select the freehand contour
init_contour = manual_freehand_contour_selection(img)

# Apply the active contour model
snake = active_contour(img,
                init_contour, alpha=0.015, beta=0.05, gamma=0.001,max_iterations=20)

# Display the results
fig, ax = plt.subplots(figsize=(7, 7))
ax.imshow(img, cmap=plt.cm.gray)
```

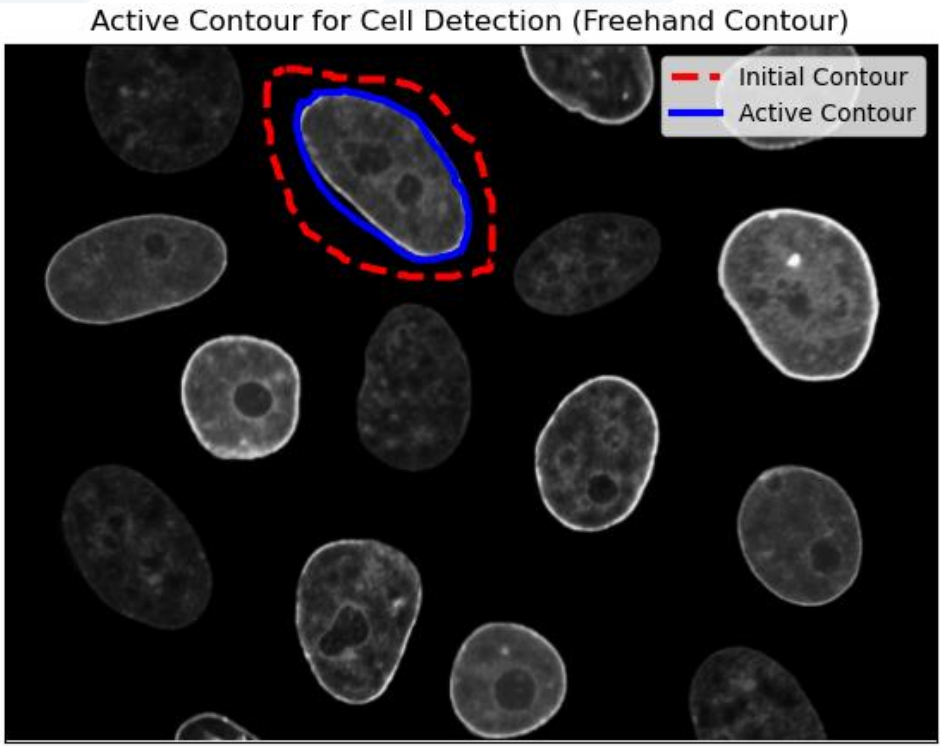
نمرر للخوارزمية AC الصورة ثم معاملات الخوارزمية alpha, beta, gama ثم عدد التكرارات الأعظمي المراد تنفيذه

```
ax.plot(init_contour[:, 1], init_contour[:, 0], '--r', lw=3, label="Initial Contour")
ax.plot(snake[:, 1], snake[:, 0], '-b', lw=3, label="Active Contour")
ax.set_xticks([]), ax.set_yticks([])
ax.axis([0, img.shape[1], img.shape[0], 0])
plt.legend()
plt.title("Active Contour for Cell Detection (Freehand Contour)")
plt.show()

# Create a path object from the final contour
path = Path(snake)

# Create a grid of points
grid = np.indices(img.shape).reshape(2, -1).T

# Use the path object to create a mask of the ROI
mask = path.contains_points(grid).reshape(img.shape)

# Extract the ROI from the original image
roi = np.where(mask, img, np.nan)

# Display the ROI
fig, ax = plt.subplots(figsize=(7, 7))
ax.imshow(roi, cmap=plt.cm.gray)
ax.set_xticks([]), ax.set_yticks([])
plt.title("Region of Interest (ROI)")
plt.show()
```

أولاً نحدد الإطار الخارجي حول المنطقة أو المناطق المهمة:

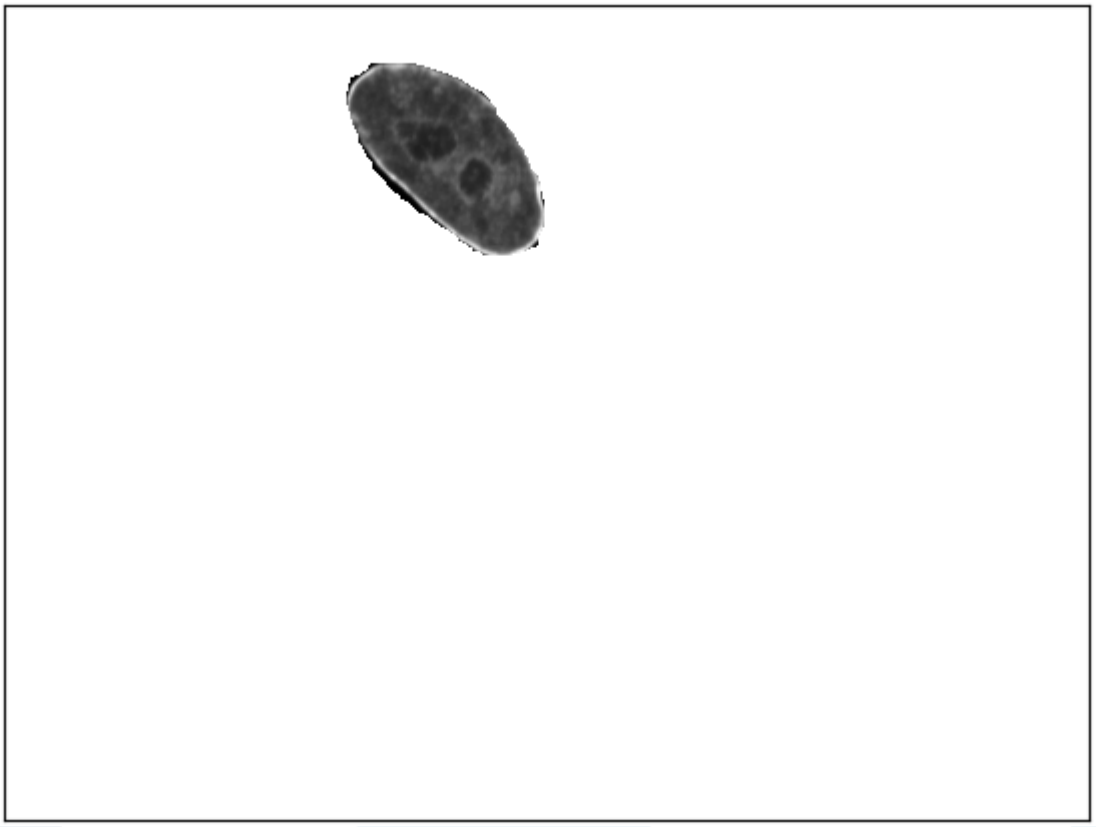


بعد ذلك نغلق الصورة فتبدأ خوارزمية AC بالعمل وعند الانتهاء تعطي الخرج التالي:



ثم يعطي الكود الصورة بعد أخذ ناتج الكشف واقتطاع ما يقابله من الصورة الأصلية

Region of Interest (ROI)



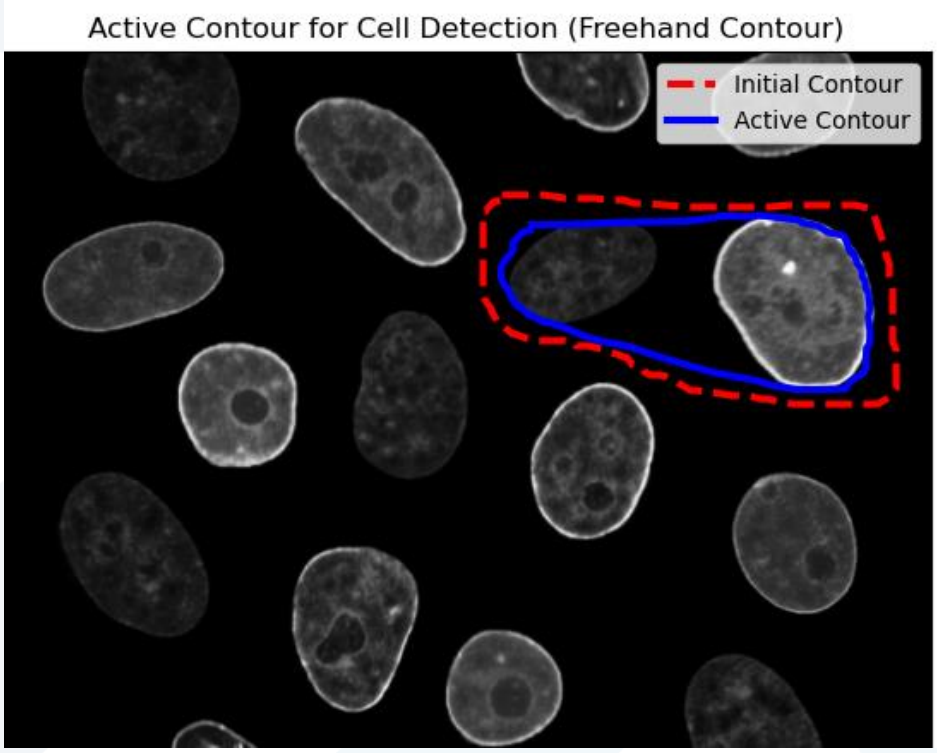
يمكن تغيير بارامترين في هذه الخوارزمية هما:

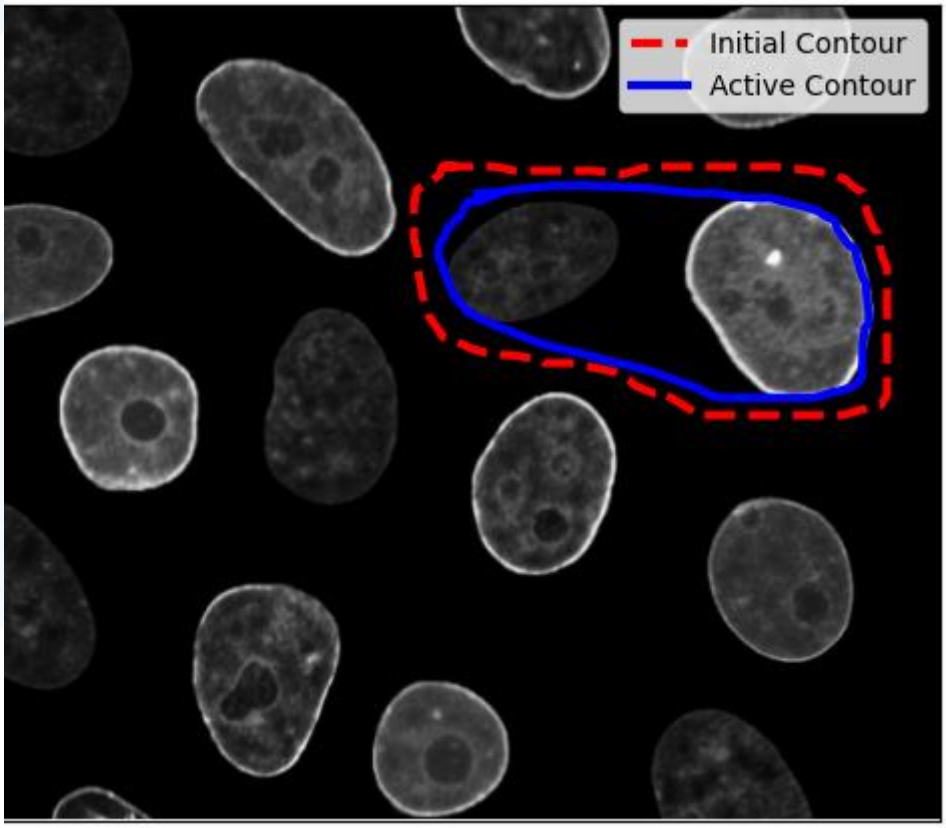
Alpha يمثل درجة المرونة elasticity بالتالي باستخدام قيم كبيرة لـ alpha يتقلص الحد بشكل سريع ليأخذ شكل أقرب منطقة له، وفي حال القيم الصغيرة يتمدد بشكل أكبر ليأخذ أكبر مساحة ممكنة للشكل الخارجي للمنطقة.

Beta ويتحكم في الصلابة Rigidity أو درجة نعومة الحدود. مع قيم عالية تكون الحدود أنعم والانحناءات في الحدود قليلة لكن مع قيم قليلة تكون الحدود أكثر انحناء وقد تكون هذه الحالة مفيدة خصوصاً عند وجود انحناءات في المنطقة المهمة.

المثالان الآتيان نفس الحد الخارجي تم رسمه فيهما لكن تم تنفيذ الخوارزمية مرة باستخدام beta=0.2 ومرة beta=0.05

عند قيمة beta صغيرة كانت المنحنيات أكثر انحناء.


Active Contour for Cell Detection (Freehand Contour)
Initial Contour
Active Contour


Initial Contour
Active Contour