**Lab 08**

|  |  |
| --- | --- |
| Name: | 易峻葦 |
| Student ID: | B08611031 |
| Total Score: |  |

**Note:**

Most of the explanations in this lab is mandatory, However, giving reasonable explanations to your answer or programs will earn you partial credits when your answer is incorrect.

1. **Multiple Choice (20 points)**

|  |  |  |  |
| --- | --- | --- | --- |
| # | Answer | Explanation (Please write codes to verify it.) | Score |
| 1 | b | 1. 建立出題目 A:   A=np.ones((10,10))  A[1,2:7]=0  A[2,1:3]=0  A[2,6:9]=0  A[3,1:3]=0  A[3,7:9]=0  A[4,1:5]=0  A[4,7]=0  A[5,1]=0  A[5,4:8]=0  A[6,1]=0  A[6,7:9]=0  A[7,1:4]=0  A[7,6:9]=0  A[8,3:8]=0   1. 建立kernel B:   B=np.ones((2,2))   1. 在A的周圍加上一層1，避免在erode 的過程出現錯誤訊息   E=np.ones((11,11))  E[1:,1:]=A  EA=E.copy()   1. 進行erode:   for r in range(10,0,-1):  for c in range(10,0,-1):  fit=E[r,c]+E[r-1,c]+E[r-1,c-1]+E[r,c-1]  if fit !=4: #if fit than retain，otherwise =0  EA[r,c]=0   1. 結果: |  |

1. **Find the Contour (20 points)**

|  |  |  |
| --- | --- | --- |
| # | Description | Score |
| 1 | 1. 讀取影像後先轉灰階後進行高斯模糊:   gray=cv2.cvtColor(rice,cv2.COLOR\_BGR2GRAY)  blur=cv2.GaussianBlur(gray, (13,13),10)    進行高斯模糊的目的是讓之後做threshold的時候可以成功讓毛被去掉   1. 進行threshold:   ret,thresh\_img=cv2.threshold(blur,90,255,cv2.THRESH\_BINARY)     1. 接著開始抓取圖片的contours，並且從中找尋邊長最大的那一個就是米粒的外圍:   cnts,hierarchy=cv2.findContours(thresh\_img,cv2.RETR\_TREE,cv2.CHAIN\_APPROX\_NONE)  length=0  for i in cnts:  l=cv2.arcLength(i,True)  if l>length:  length=cv2.arcLength(i,True)  points=i   1. 在原圖中畫上(3)找到的最大邊長的資料點:   cv2.drawContours(rice,points,-1,(0,0,255),5)  cv2.imshow('rice',rice) |  |

1. **Image Enhancement (25 points)**

|  |  |  |
| --- | --- | --- |
| # | Description | Score |
| - | Paste your result from (a) to (g) here.    (a)laplace\_filter=np.array([[0,1,0],[1,-4,1],[0,1,0]])  laplace=convolve2d(image,laplace\_filter)  (b)sharpen\_image=image-laplace[2:413,2:259]  (c)sobel= filters.sobel(sharpen\_image)#\_v(sharpen\_image)+filters.sobel\_h(sharpen\_image)  (d)avg\_filter=np.ones((5,5))/25  avg\_sobel= convolve2d(sobel,avg\_filter)  (e)multiply= avg\_sobel[2:413,2:259]\*sharpen\_image  (f)add=image+multiply  (g)Power\_law=add\*\*0.5  #因為前面做Laplace filter 的時候會有負數的產生，導致(g)步驟會產生需數不能夠顯示出來，因此之後才有一個找到虛數的index並將其改為0的步驟  Power\_law[np.isnan(Power\_law)]=0 |  |

1. **Fingerprint Analysis (35 points)**

|  |  |  |
| --- | --- | --- |
| # | Description | Score |
| - | Paste your result here and briefly describe your image processing procedure and approach. How do you think your results are?    步驟:   1. 將圖片normalize:   fingerprint\_normalize=(fingerprint-fingerprint.min())\*255/(fingerprint.max()-fingerprint.min()))\*255   1. 利用Laplace filter 銳化影像:   laplace\_filter=np.array([[0,1,0],[1,-4,1],[0,1,0]])  laplace=convolve2d(fingerprint\_normalize,laplace\_filter)  fingerprint\_normalize=fingerprint\_normalize-laplace[1:481,1:401]   1. 利用Gabor filter:   kernel = cv2.getGaborKernel((15, 15), 5, 1, 10, 0.5, 0, cv2.CV\_32F)  kernel /= math.sqrt((kernel \* kernel).sum())  fingerprint\_normalize== cv2.filter2D(fingerprint\_normalize, -1, kernel)     1. 利用otsu 演算法得到threshold後得到黑白圖:   otsu\_thresh= filters.threshold\_otsu(fingerprint\_normalize)  otsu\_local= fingerprint\_normalize>otsu\_thresh     1. 利用remove small hole、dilation、erosion 修飾指紋:   otsu\_local= morphology.remove\_small\_holes(otsu\_local,30)  otsu\_local= morphology.dilation(otsu\_local,morphology.square(2))  otsu\_local= morphology.erosion(otsu\_local,morphology.square(2))  fingerprint2=otsu\_local         1. 最後 skeletonizec後，再以cross-number 演算法提取特徵，原理如下:   (參考論文來源: <http://bura.brunel.ac.uk/handle/2438/7473>  )  設定一個3X3矩陣P，其樣式如下:    P為圖片skeletonize後圖片中的第(I,j)個像素點，P1~P8為中心像素點周圍的像素，  定義該點的cross number CN 為:  未提供說明。  而不同的指紋特徵會有不同的CN，像是本次作業所要求的 ridge-ending  以及bifurcation分別為1、3(如下圖):  未提供說明。  程式碼:  def cross\_number(image):  w=image.shape[0]+2  h=image.shape[1]+2  CN\_image=np.zeros((w,h))  im=np.zeros((w,h))  im[1:w-1,1:h-1]=image  for i in range(1,image.shape[0]+1):  for j in range(1,image.shape[1]+1):  if im[i,j]==1:  P=np.zeros((9))  P[0]= P[8]= im[i,j+1]  P[1]= im[i-1,j+1]  P[2]= im[i-1,j]  P[3]= im[i-1,j-1]  P[4]= im[i,j-1]  P[5]= im[i+1,j-1]  P[6]= im[i+1,j]  P[7]= im[i+1,j+1]  sum=0  for k in range(0,8):  sum+=abs(P[k]-P[k+1])  CN\_image[i,j]=0.5\*sum  CN\_image=CN\_image[1:w-1,1:h-1]  return CN\_image   1. 得到指紋的CN marix後，再根據CN matrix 中的數字去提取特徵:     但從我的結果中可知效果不太理想，很明顯是因為在影像處理的時候做得不太完全，在我參考CN演算法的那篇論文中，他們的預處理並不是像dilation、erosion 之類的那麼簡單，而是包含了 Orientation estimation、Ridge frequency estimation 等較複雜的處理，並且他們只寫出數學概念，並無提供程式範碼，同時也包含了我使用到的Gabor filter，礙於時間關係，我只簡單大概利用了Gabor filter，也因此最後圖片出來的雜訊出現那麼多。 |  |