"""

#----------#

Trabalho 02 - Processamento de Imagens

Flávio Henrique Andrade dos Santos - 201020002040

#----------#

O diretorio img\BloodImageSetS6NucSeg contém todas as imagens que serão utilizadas, além de dois arquivos de texto, os quais são usados para obter as imagens do diretório, o arquivo test.txt contém apenas algumas imagens para teste, já o index.txt possui todas as imagens.

O diretorio tmp\ é utilizado para registrar os resultados das imagens segmentadas

Descompactar a biblioteca de imagens no diretorio img\

Passo a Passo para testar o algoritmo

primeiro passo:

import flavio_henrique as fla

segundo passo:

Para executar um teste com poucas imagens basta chamar o método:

fla.test()

imagens que serão processadas:

BloodImage_00074.jpg

BloodImage_00313.jpg

BloodImage_00000.jpg

BloodImage_00010.jpg

BloodImage_00077.jpg

BloodImage_00407.jpg

BloodImage_00110.jpg

BloodImage_00242.jpg

BloodImage_00041.jpg

BloodImage_00164.jpg

BloodImage_00069.jpg

BloodImage_00054.jpg

no diretório tmp\ será criadas as imagens com os resultados da segmentação

terceiro passo (opcional)

Para executar uma segmentação com todas as imagens basta chamar o método:

fla.main()

quarto passo (processamento de uma única imagem)

Para executar o teste e exibir o resultado da segmentação da imagem BloodImage_00000.jpg

fla.testImg("BloodImage_00000.jpg")

quinto passo (processamento de uma única imagem com o debug ativo)

Para executar o teste e exibir o resultado da segmentação da imagem BloodImage_00000.jpg

com o debug

fla.testImg("BloodImage_00000.jpg",1)

Observações:

Quando realizei o teste em todas as imagens, algumas não apresentaram o resultado esperado,

são elas:

BloodImage_00017.jpg

BloodImage_00045.jpg

BloodImage_00048.jpg

BloodImage_00054.jpg

BloodImage_00059.jpg

BloodImage_00062.jpg

BloodImage_00069.jpg

BloodImage_00106.jpg

BloodImage_00164.jpg

BloodImage_00352.jpg

BloodImage_00034.jpg

BloodImage_00189.jpg

BloodImage_00201.jpg

BloodImage_00239.jpg

BloodImage_00298.jpg

BloodImage_00331.jpg

BloodImage_00392.jpg

BloodImage_00398.jpg

No diretório resultados estou disponibilizando algumas imagens segmentadas por esse algoritmo.

"""

```
import cv2
import numpy as np
from PIL import Image,ImageOps,ImageChops
import os
#limpa tela
os.system('clear')
total = 0
cont = 0
def segmentation(path, name):
    global total,cont
    path_tmp = os.getcwd() + "\\tmp"
    remove(path_tmp)
    path_full = os.getcwd() + path + name
    if os.path.exists(path_full):
        f = open(path_full)
        text = f.read()
```

```
        arr = text.split("\n")
        for line in arr:
            if os.path.exists(os.getcwd() + path + line):
                tmp = process(os.getcwd() + path + line)
                cv2.imwrite(path_tmp+"\\"+line.split(".")[0]+"_tmp."+line.split(".")[1],tmp)
                #cont += 1
        #print total/cont
    else:
        print "arquivo não encontrado."
def process(img,debug=0):
    imgDebug = []
    A = cv2.imread(img)
    imgDebug.append(A.copy())
    B = np.asarray(A.dot([1.900, -0.301, -0.599]),dtype="uint8") # passo 1
    imgDebug.append(B.copy())
    L = np.asarray(ImageOps.autocontrast(Image.fromarray(np.uint8(B)))) # passo 2
    imgDebug.append(L.copy())
    #L = cv2.multiply(B,np.asarray([2.0]))
    H = cv2.equalizeHist(B) # passo 3
    imgDebug.append(H.copy())
    R1 = cv2.add(L,H) # passo 4
    imgDebug.append(R1.copy())
    R2 = cv2.subtract(L,H) # passo 5
    imgDebug.append(R2.copy())
    R3 = cv2.add(R1,R2) # passo 6
    imgDebug.append(R3.copy())
    for i in range(3): #passo 7
```

```
        R3 = cv2.blur(R3,(3,3))
    imgDebug.append(R3.copy())
    #passos 8 e 9
    thresh = method_otsu(R3)
    R3 = Image.fromarray(np.uint8(R3))
    result = R3.point(lambda i: i >= thresh and 255)
    imgDebug.append(np.asarray(result.copy(),dtype="uint8"))
    kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(19,19)) #passo 10
    opening =
cv2.morphologyEx(np.asarray(result.copy(),dtype="uint8"),cv2.MORPH_OPEN,kernel)
    #opening = cv2.morphologyEx(otsu,cv2.MORPH_OPEN,kernel)
    imgDebug.append(opening.copy())
    closing = cv2.morphologyEx(opening.copy(),cv2.MORPH_CLOSE,kernel) #passo 11
    imgDebug.append(closing.copy())
    im = closing
    calc_area(im)   #passo 12
    imgDebug.append(im.copy())
    #area_media(im) #teste realizado para determinar uma média do núcleo
    im = cv2.bitwise_and(A,cv2.cvtColor(im,cv2.COLOR_GRAY2BGR)) #aplicando a mascara na
imagem original
    if(debug):
        show(imgDebug,debug)
    return im
def area_media(im):
    global total
    maxArea = -np.inf
    contours, hierarchy =
cv2.findContours(im.copy(),cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
    for cnt in contours:
```

```
        if cv2.contourArea(cnt) > maxArea:
            maxArea = cv2.contourArea(cnt)
    total += maxArea
def calc_area(im):
    cnt, h = cv2.findContours(im.copy(),cv2.RETR_TREE,cv2.CHAIN_APPROX_NONE)
    cnt = [c for c in cnt if cv2.contourArea(c) < 4000.0]
    cv2.drawContours(im, cnt, 0, 255, -1)
def method_otsu(img):
    blur = cv2.GaussianBlur(img,(5,5),0)
    # encontrar normalized_histogram, e a sua função de distribuição cumulativa
    hist = cv2.calcHist([blur],[0],None,[256],[0,256])
    hist_norm = hist.ravel()/hist.max()
    Q = hist_norm.cumsum()
    bins = np.arange(256)
    fn_min = np.inf
    thresh = -1
    for i in xrange(1,256):
        p1,p2 = np.hsplit(hist_norm,[i]) # probabilidades
        q1,q2 = Q[i],Q[255]-Q[i] # função de distribuição cumulativa das classes
        b1,b2 = np.hsplit(bins,[i]) # pesos
        if(q1 == 0):
            continue
        if(q2 == 0):
            break
        # encontrando médias e variâncias
        m1,m2 = np.sum(p1*b1)/q1, np.sum(p2*b2)/q2
        v1,v2 = np.sum(((b1-m1)**2)*p1)/q1,np.sum(((b2-m2)**2)*p2)/q2
```

```
        # calcula a função de minimização
        fn = v1*q1 + v2*q2
        if fn < fn_min:
            fn_min = fn
            thresh = i
    return thresh;
def otsu2( hist, total ):
    no_of_bins = len( hist )
    sum_total = 0
    for x in range( 0, no_of_bins ):
        sum_total += x * hist[x]
    weight_background     = 0.0
    sum_background        = 0.0
    inter_class_variances = []
    for threshold in range( 0, no_of_bins ):
        weight_background += hist[threshold]
        if weight_background == 0 :
            continue
        weight_foreground = total - weight_background
        if weight_foreground == 0 :
            break
        sum_background += threshold * hist[threshold]
        mean_background = sum_background / weight_background
        mean_foreground = (sum_total - sum_background) / weight_foreground
        inter_class_variances.append( weight_background * weight_foreground *
(mean_background - mean_foreground)**2 )
    return np.argmax(inter_class_variances)
def show(i,debug=0):
```

```
    if(debug):
        cont = 0
        for v in i:
            cont += 1
            name = "image" + str(cont)
            cv2.imshow(name,v)
        cv2.waitKey(0)
    else:
        cv2.imshow("default",i)
        cv2.waitKey(0)
def remove(dirPath):
    fileList = os.listdir(dirPath)
    for fileName in fileList:
        os.remove(dirPath+"\\"+fileName)
def main():
    segmentation("\\img\\BloodImageSetS6NucSeg\\","index.txt")
#main()
def test():
    segmentation("\\img\\BloodImageSetS6NucSeg\\","test.txt")
#test()
def testImg(img,debug=0):
    show(process("img\\BloodImageSetS6NucSeg\\"+img,debug))
#testImg()
```