FALL 22 ECE 661 – COMPUTER VISION

Homework 6

Sahithi Kodali – 34789866
kodali1@purdue.edu

THEORY QUESTION

Otsu Algorithm:

Otsu Algorithm works well for images with bimodal histograms (two peaks) and is simple to compute. However, it yields noisy results despite being simple. When there is a large difference between the background and foreground images the image segmentation doesn't perform as accurately as estimated. This is due to the overlap in the grey scale images - which are sensitive to noise – used in the Otsu algorithm. Even in iterative Otsu method, it is required for the foreground image in each iteration to be bimodal.

Watershed Algorithm:

Watershed Algorithm is very fast compared to Otsu and produces complete boundaries usually. Hence, it produces less noisy results, however the markers given initially need to be good enough for the method to work as desired. It has a disadvantage of giving over segmented results due to the own area made by many estimated local minima points in an image, giving dense small areas. A good thing is that new methods are being introduced to tackle the over segmentation in watershed algorithm.

TASK – 1

Two input images shown below are provided to perform image segmentation based on RGB values and Textures using Otsu Algorithm, followed by Contour extraction for the images.

Input image (a)

Input image (b)

1.1 Otsu Algorithm

Otsu algorithm works by diving an image into 2 classes, namely, foreground and background whose probabilities are given by w_0 and w_1 and, means by u_0 and u_1. The probability distribution function (histogram frequencies) i.e., p_i at a gray scale pixel 'i' are computed using the gray pixel values at 'i' level by the total number of pixels in the image (N). The value of u_T is given as the mean of the entire image pixels.

$$w_0 = \sum_{k=0}^{i-1} p_i = \sum_{k=0}^{i-1} \frac{n_i}{N} \qquad and \qquad w_1 = 1 - w_0$$

$$u_0 = \frac{1}{w_0} \sum_{k=0}^{i-1} ip_i \qquad and \qquad u_1 = \frac{1}{w_0}(u_T - w_0 u_0)$$

The goal is the maximize the between class variance to obtain the threshold value given as σ_b^2 .

$$\sigma_b^2 = w_0 w_1 (u_0 - u_1)^2$$

All the threshold values are computed at each level and the threshold that gives the maximum between class-variance is considered for the foreground and background masking.

1.2 Image segmentation using RGB values

To perform Image segmentation, Otsu algorithm is run on the three RGB channels of the images separately. A masking parameter is given which assist in keeping track of the foreground class above or below the threshold computed.

All the three RGB channels results are combined to get the image segmentation of foreground from the background of the image. The following aer the parameters chosen fro each type of inpout image a, b, c, d.

a. For the car image, the number of iterations chosen are [1,1,1] and the type mask is [0,0,0]. It is optimized on all the channels.

b. For the cat image, the number of iterations chosen are [1,3,1] and the type mask is [1,1,1]. It is optimized more on the green channel with three iterations and inverted with respected to all channels.

c. For the bird image, the number of iterations chosen are [1,0,1] and the type mask is [1,1,1]. It is optimized more on the red and blue channels and inverted with respect to first channel.

d. For the dog image, the number of iterations chosen are [0,0,2] and the type mask is [0,0,0]. It is optimized more on the blue channel and iterated twice.

1.3 Texture - based segmentation

A sliding window approach is used to compute the texture values of pixel in the given image. Three window sizes in the form N X N are considered to slide through every pixel value in the image and compute the variances of their neighbourhood. The three sizes give three channels of variance-based images that can be considered as the texture features for a respective pixel. These three channels images which are variance-based images are used separately to compute the Otsu threshold.

a. For the car image, the number of iterations chosen are [1,3,1] and the type mask is [1,1,1]. It is optimized more on the green channel with three iterations and inverted with respected to all channels.

b. For the cat image, the number of iterations chosen are [1,3,1] and the type mask is [1,1,1]. It is optimized more on the green channel with three iterations and inverted with respected to all channels.

c. For the bird image, the number of iterations chosen are [3,3,6] and the type mask is [1,1,1]. It is optimized more on the blue channel and inverted with respect to all channels.

d. For the dog image, the number of iterations chosen are [6,3,3] and the type mask is [1,1,1]. It is optimized more on the Red channel and inverted with respect to all channels.

1.4 Contour Extraction

Contours are obtained by extracting the pixels that form edges. To connect the edges completely and get rid of the small high value pixels that appear as dots, dilation and erosion operations are performed. If at least one pixel in the 9-pixel neighbourhood of the pixel belongs to the background (i.e., 0), then the pixel is considered as an edge.

RESULTS:

Figure-1: RGB Segmentation - Channel 1 foreground mask of input image (a)

Figure-2: RGB Segmentation - Channel 2 foreground mask of input image (a)

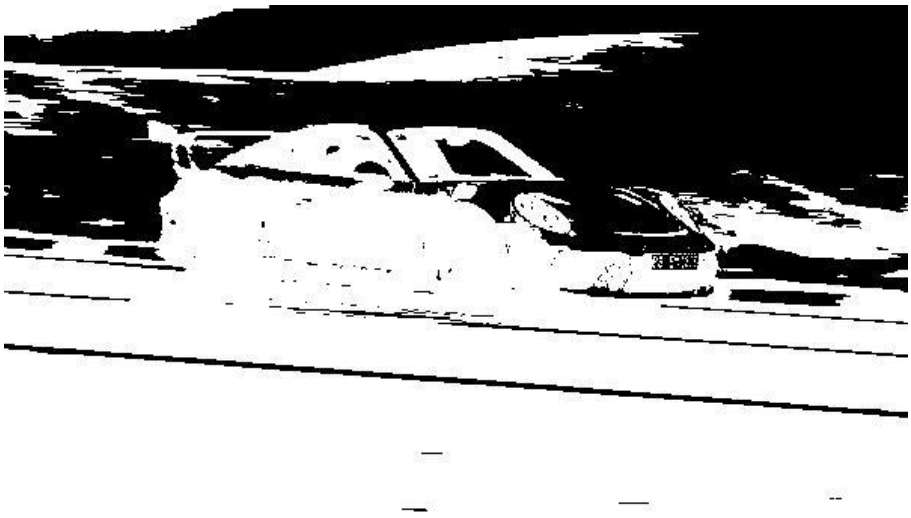

Figure-3: RGB Segmentation - Channel 3 foreground mask of input image (a)

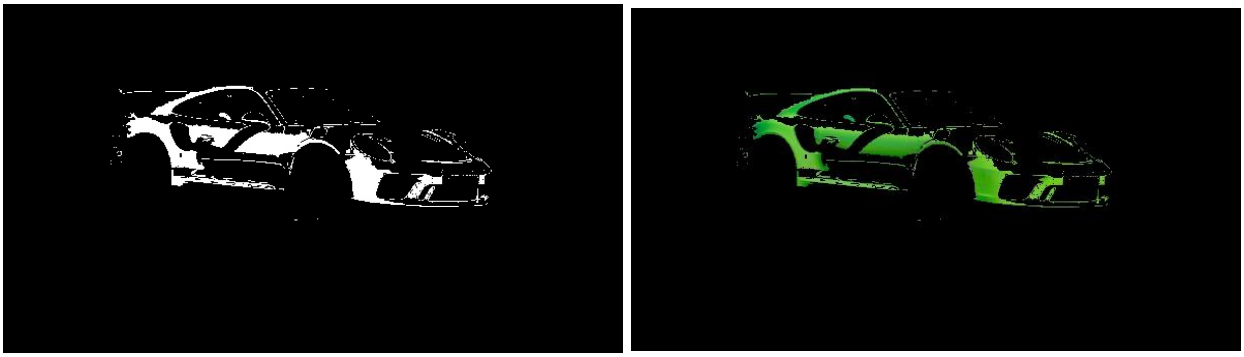

Figure-4: RGB Segmentation – Combined foreground mask and image of input image (a)

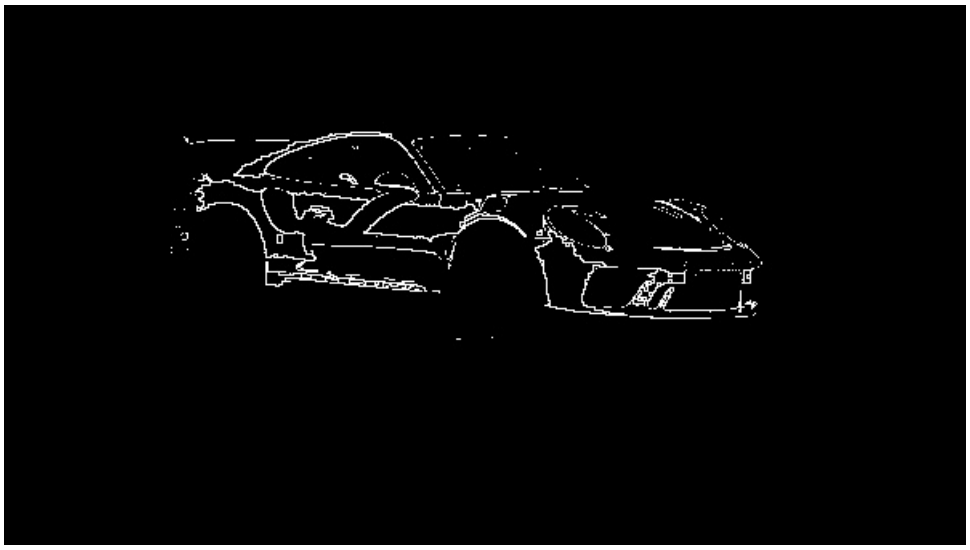

Figure-5: RGB Segmentation – Contour extraction of input image (a)

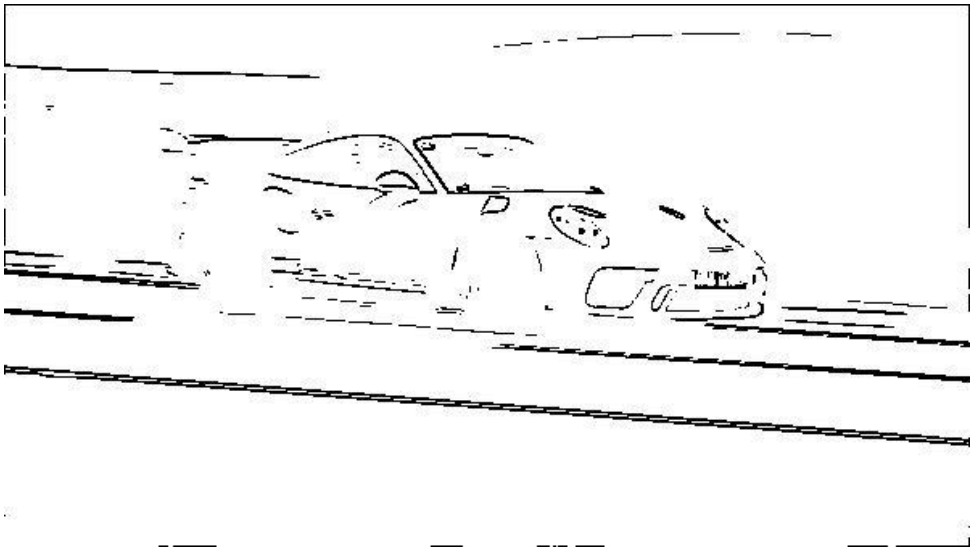

Figure-6: Texture Segmentation - Window 1 foreground mask of input image (a)

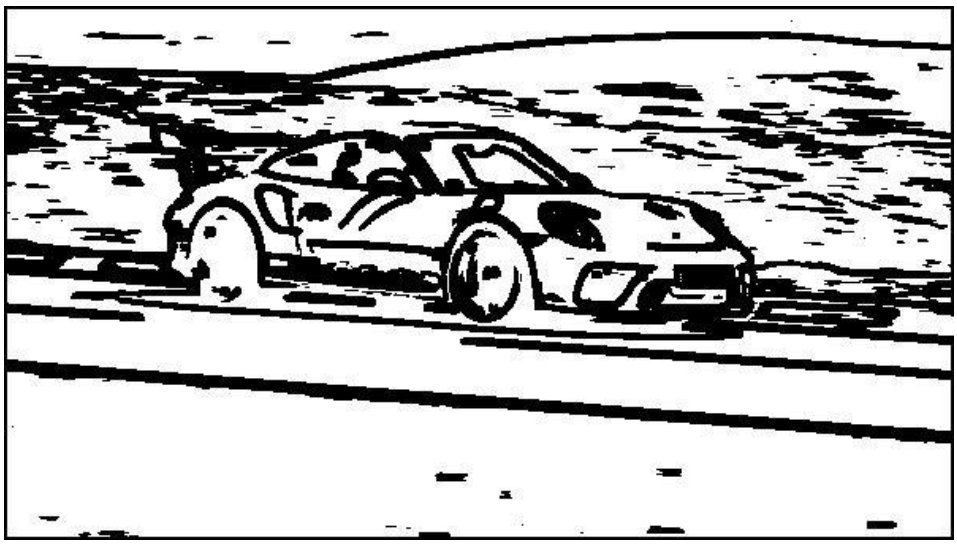

Figure-7: Texture Segmentation - Window 2 foreground mask of input image (a)

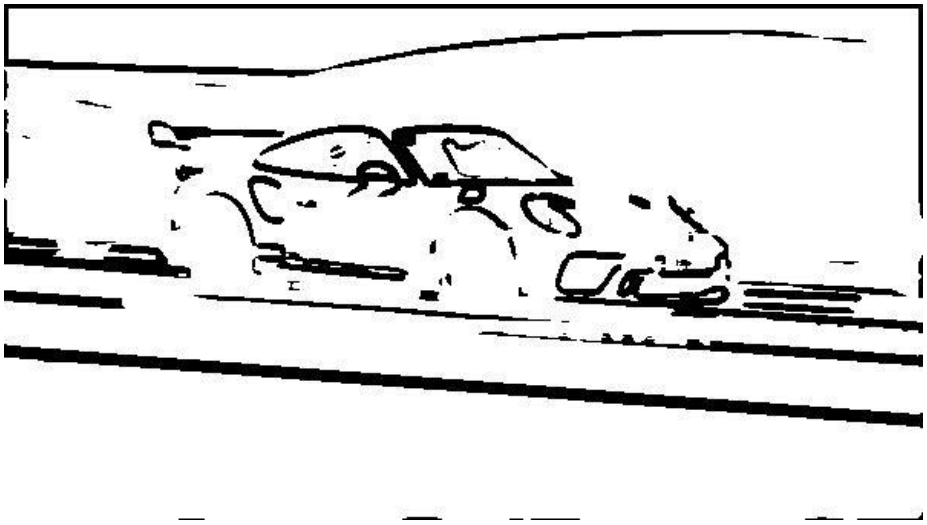

Figure-8: Texture Segmentation - Window 3 foreground mask of input image (a)

Figure-9: Texture Segmentation – All windows combined foreground mask of input image (a)

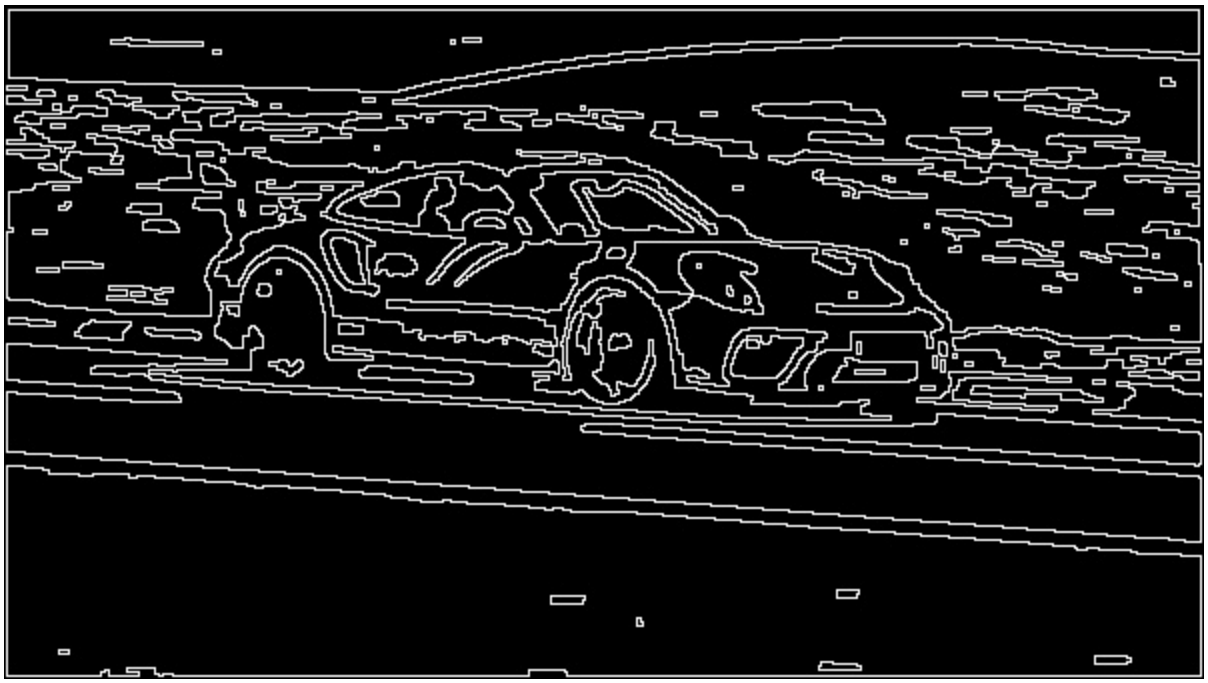

Figure-10: Texture Segmentation – Contour extraction of input image (a)

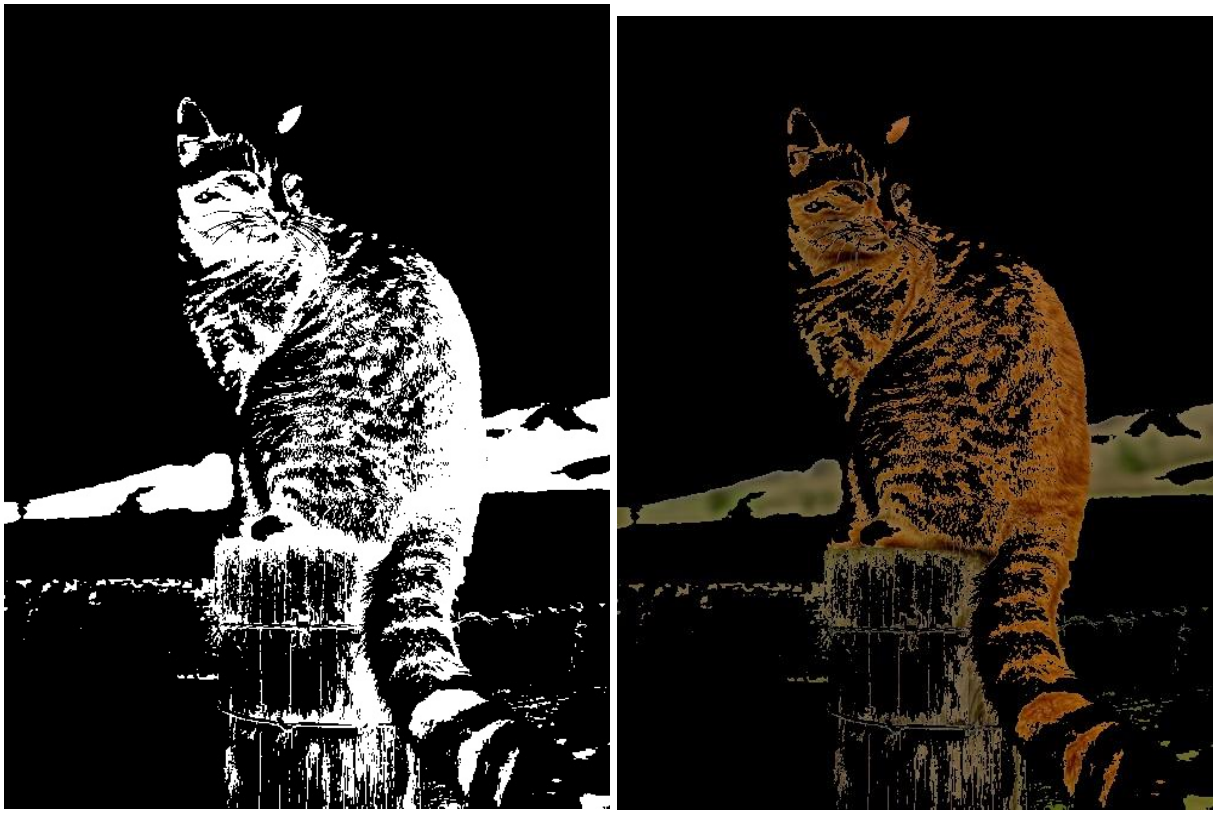

Figure-11: RGB Segmentation – All channels combined foreground mask of input image (b)

Figure-12: RGB Segmentation – Contour extraction of input image (b)

Figure-13: Texture Segmentation – All windows combined foreground mask of input image (b)

Figure-14: Texture Segmentation – Contour extraction of input image (b)

TASK – 2

The same methods as in Task-1 are applied on two own images shown below.

Input Image (c)

Input Image (d)

Figure-15: RGB Segmentation – Combined foreground mask and image of input image (c)

Figure-16: RGB Segmentation – Contour extraction of input image (c)

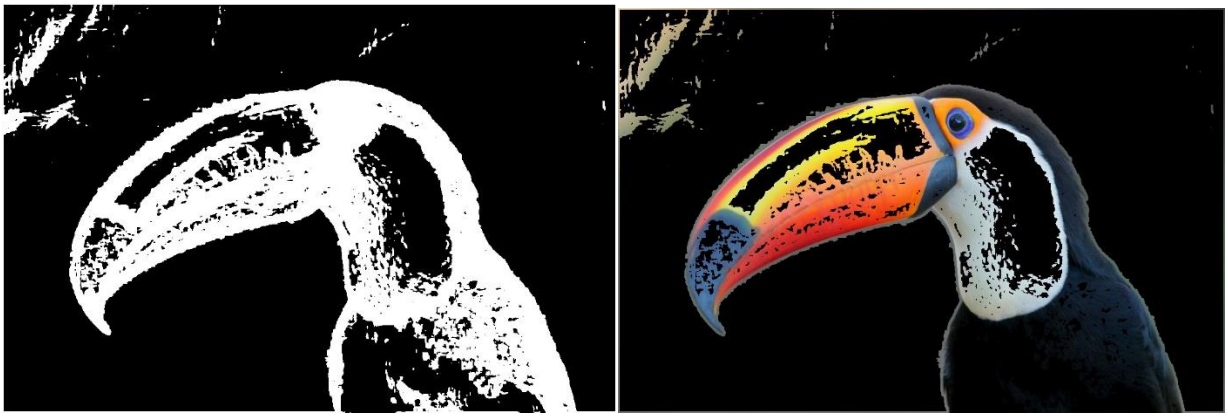

Figure-17: Texture Segmentation – All windows combined foreground mask of input image (c)

Figure-18: Texture Segmentation – Contour extraction of input image (c)

Figure-19: RGB Segmentation – All channels combined foreground mask of input image (d)

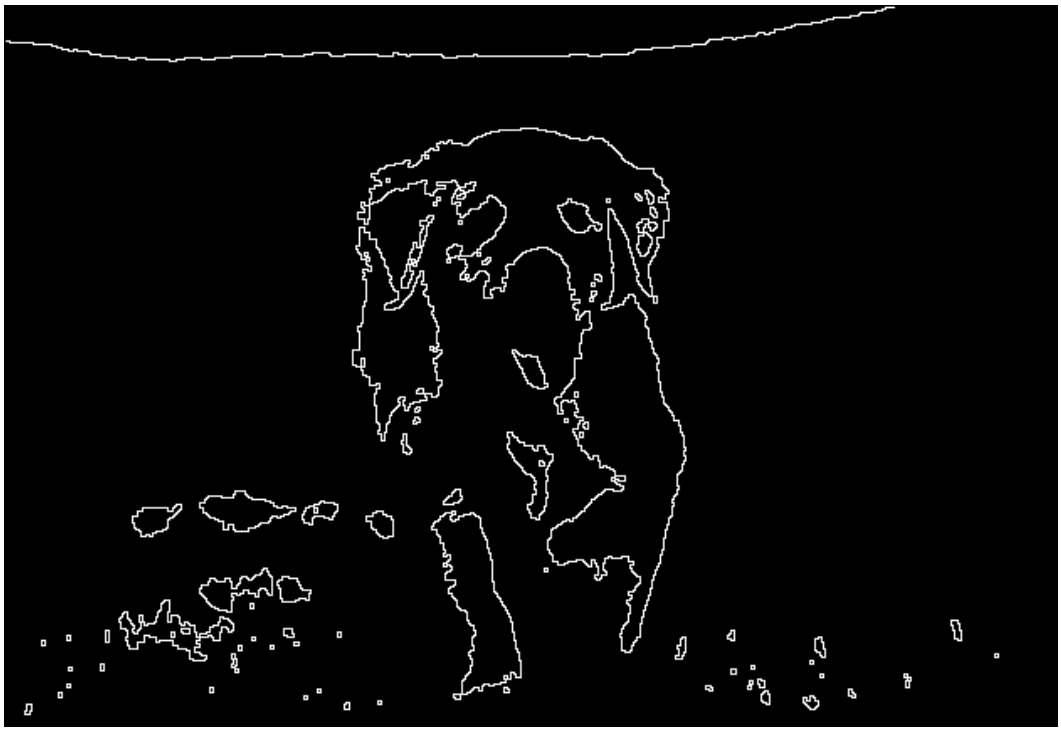

Figure-20: RGB Segmentation – Contour extraction of input image (d)

Figure-21: Texture Segmentation – All windows combined foreground mask of input image (d)

Figure-22: Texture Segmentation – Contour extraction of input image (d)

SOURCE CODE:

```
#!/usr/bin/env python
# coding: utf-8

# <h2><center>ECE661 COMPUTER VISION</center></h2>
# <h3><center>Homework - 6 </center></h3>
# <h3><center>Sahithi Kodali - 34789866</center></h3>
# <h3><center>kodali1@purdue.edu</center></h3>

# In[1]:


# Import libraries needed

import numpy as np
import matplotlib.pyplot as plt
import skimage.io as sk
import cv2
import math
import copy
```

```
# In[427]:


# Read the images from uplaoded files/directly by giving path in the computer

car_img = cv2.imread('car.jpg')
cat_img = cv2.imread('cat.jpg')

own1_img = cv2.imread('bird.jpg')
own2_img = cv2.imread('dog.png')


# In[285]:


#perform Otsu thresholding
def otsu_threshMasking(img, iterations, type_mask):

    output_mask = np.ones(img.shape)
    print(output_mask.shape)

    #compute total pixels
    for j in range(iterations):
        total_pixels = img[output_mask != 0]

        #Get the Pixels for each grey scale level i and compute the probability
density function
        pixels_perThresh, thresh = np.histogram(total_pixels, bins =
np.unique(total_pixels))
        pdf = pixels_perThresh/float(len(total_pixels))

        #initialize variables to compute the Otsu method variables
        w_0 = 0
        u_0 = 0
        u_T = np.mean(total_pixels)
        u0_num = 0

        max_sigmaSqr = 0

        max_ThreshLevel = -math.inf
        pixels_i = 0

        #Iterate through each gray scale level i and compute the maximum
threshold
```

```
        for i in thresh[:-1]:

            w_0 = w_0 + pdf[pixels_i]
            w_1 = 1 - w_0

            if w_1 == 0:
                break

            u0_num = u0_num + (i * pdf[pixels_i])
            u_0 = u0_num/w_0
            u_1 = (u_T - u0_num)/w_1

            sigmaSqr = w_0 * w_1 * np.square(u_0 - u_1)

            pixels_i = pixels_i + 1

            if sigmaSqr > max_sigmaSqr:
                max_sigmaSqr = sigmaSqr
                max_ThreshLevel = i

        print(max_ThreshLevel)

        #check the threshold with respect to foreground image
        if type_mask == 0:
            get_foreGround = img > max_ThreshLevel
            output_mask = get_foreGround
        else:
            get_backGround = img < max_ThreshLevel
            output_mask = get_backGround

    return output_mask


# In[286]:


#get the contour in the image
def get_Contours_cv(mask):

    contours = np.zeros(mask.shape).astype(np.uint8)

    #check if any neighbour has 0 in it, if so make it an edge and return the
contour
    for i in range(1, mask.shape[0] - 1):
        for j in range(1, mask.shape[1] - 1):
```

```
            if mask[i][j] == 0:
                continue
            if np.min(mask[i-1:i+2, j-1:j+2]) == 0:
                contours[i,j] = 255
    return contours


# ### Car_Img

# In[287]:


#Initialize parameters
iterations = [1,1,1]
type_mask = [1,0,1]


# In[288]:


# determine each channel mask and the mask combined

BGR_masks = np.zeros(car_img.shape)

for channel in range(car_img.shape[-1]):

    car_copy = copy.deepcopy(car_img[:,:,channel])

    mask_channel = otsu_threshMasking(car_copy.flatten(), iterations[channel],
type_mask[channel])
    BGR_masks[:,:,channel] = mask_channel.reshape((car_img.shape[0],
car_img.shape[1]))

    filename = 'D:\Purdue\ECE661_CV\HW6\HW6_outputs' + '\car_channel_' +
str(channel) +'.jpg'
    plt.imsave(filename, BGR_masks[:,:,channel], cmap = 'gray' )

foreGround_mask = np.multiply(np.multiply(
BGR_masks[:,:,0],  BGR_masks[:,:,1]),  BGR_masks[:,:,2])
filename1 = 'D:\Purdue\ECE661_CV\HW6\HW6_outputs' + '\car_foregroundMask' +
'.jpg'
cv2.imwrite(filename1, foreGround_mask*255)


# In[289]:
```

```
#Get the combined mask image displayed
foreGround_img = np.zeros(car_img.shape)
for channel in range(car_img.shape[-1]):
    foreGround_img[:, :, channel] = np.multiply(car_img[:,:,channel],
foreGround_mask)

filename2 = 'D:\Purdue\ECE661_CV\HW6\HW6_outputs' + '\car_foregroundImg' + '.jpg'
cv2.imwrite(filename2, foreGround_img)


# In[290]:


#Get the contours
contourImg = get_Contours_cv(foreGround_mask)
filename2 = 'D:\Purdue\ECE661_CV\HW6\HW6_outputs' + '\car_contourImg_mask' +
'.jpg'
cv2.imwrite(filename2, contourImg)


# In[291]:


#Perform erosion followed by dilation
kernel = np.ones((3,3))
eroded_img = cv2.erode(np.float32(foreGround_mask), kernel, iterations = 1)
dilated_img = cv2.dilate(eroded_img, kernel, iterations = 1)


# In[292]:


#get the countor image after the dilation
contourImg = get_Contours_cv(dilated_img)
filename2 = 'D:\Purdue\ECE661_CV\HW6\HW6_outputs' + '\car_contourImg_dilated' +
'.jpg'
cv2.imwrite(filename2, contourImg)


# ### cat_img

# In[293]:


#Initialize parameters
```

```
iterations = [1,3,1]
type_mask = [1,1,1]


# In[294]:


# determine each channel mask and the mask combined

BGR_masks = np.zeros(cat_img.shape)

for channel in range(cat_img.shape[-1]):

    cat_copy = copy.deepcopy(cat_img[:,:,channel])

    mask_channel = otsu_threshMasking(cat_copy.flatten(), iterations[channel],
type_mask[channel])
    BGR_masks[:,:,channel] = mask_channel.reshape((cat_img.shape[0],
cat_img.shape[1]))

    filename = 'D:\Purdue\ECE661_CV\HW6\HW6_outputs' + '\cat_channel_' +
str(channel) +'.jpg'
    plt.imsave(filename, BGR_masks[:,:,channel], cmap = 'gray' )

foreGround_mask = np.multiply(np.multiply(
BGR_masks[:,:,0],  BGR_masks[:,:,1]),  BGR_masks[:,:,2])
filename1 = 'D:\Purdue\ECE661_CV\HW6\HW6_outputs' + '\cat_foregroundMask' +
'.jpg'
cv2.imwrite(filename1, foreGround_mask*255)


# In[295]:


#Get the combined mask image displayed

foreGround_img = np.zeros(cat_img.shape)
for channel in range(cat_img.shape[-1]):
    foreGround_img[:, :, channel] = np.multiply(cat_img[:,:,channel],
foreGround_mask)

filename2 = 'D:\Purdue\ECE661_CV\HW6\HW6_outputs' + '\cat_foregroundImg' + '.jpg'
cv2.imwrite(filename2, foreGround_img)


# In[296]:
```

```
#Get the contours

contourImg = get_Contours_cv(foreGround_mask)
filename2 = 'D:\Purdue\ECE661_CV\HW6\HW6_outputs' + '\cat_contourImg_mask' +
'.jpg'
cv2.imwrite(filename2, contourImg)


# In[297]:


#Perform erosion followed by dilation

kernel = np.ones((3,3))
eroded_img = cv2.erode(np.float32(foreGround_mask), kernel, iterations = 2)
dilated_img = cv2.dilate(eroded_img, kernel, iterations = 2)


# In[298]:


#get the countor image after the dilation

contourImg = get_Contours_cv(dilated_img)
filename2 = 'D:\Purdue\ECE661_CV\HW6\HW6_outputs' + '\cat_contourImg_dilated' +
'.jpg'
cv2.imwrite(filename2, contourImg)


# ### bird_img

# In[299]:


#Initialize parameters

iterations = [1,0,1]
type_mask = [1,0,0]


# In[300]:


# determine each channel mask and the mask combined
```

```
BGR_masks = np.zeros(own1_img.shape)

for channel in range(own1_img.shape[-1]):

    own1_copy = copy.deepcopy(own1_img[:,:,channel])

    mask_channel = otsu_threshMasking(own1_copy.flatten(), iterations[channel],
type_mask[channel])
    BGR_masks[:,:,channel] = mask_channel.reshape((own1_img.shape[0],
own1_img.shape[1]))

    filename = 'D:\Purdue\ECE661_CV\HW6\HW6_outputs' + '\own1_channel_' +
str(channel) +'.jpg'
    plt.imsave(filename, BGR_masks[:,:,channel], cmap = 'gray' )

foreGround_mask = np.multiply(np.multiply(
BGR_masks[:,:,0],  BGR_masks[:,:,1]),  BGR_masks[:,:,2])
filename1 = 'D:\Purdue\ECE661_CV\HW6\HW6_outputs' + '\own1_foregroundMask' +
'.jpg'
cv2.imwrite(filename1, foreGround_mask*255)


# In[301]:


#Get the combined mask image displayed

foreGround_img = np.zeros(own1_img.shape)
for channel in range(own1_img.shape[-1]):
    foreGround_img[:, :, channel] = np.multiply(own1_img[:,:,channel],
foreGround_mask)

filename2 = 'D:\Purdue\ECE661_CV\HW6\HW6_outputs' + '\own1_foregroundImg' +
'.jpg'
cv2.imwrite(filename2, foreGround_img)


# In[302]:


#Get the contours

contourImg = get_Contours_cv(foreGround_mask)
filename2 = 'D:\Purdue\ECE661_CV\HW6\HW6_outputs' + '\own1_contourImg_mask' +
'.jpg'
```

```
cv2.imwrite(filename2, contourImg)


# In[303]:


#Perform erosion followed by dilation

kernel = np.ones((3,3))
eroded_img = cv2.erode(np.float32(foreGround_mask), kernel, iterations = 1)
dilated_img = cv2.dilate(eroded_img, kernel, iterations = 1)


# In[304]:


#get the countor image after the dilation

contourImg = get_Contours_cv(dilated_img)
filename2 = 'D:\Purdue\ECE661_CV\HW6\HW6_outputs' + '\own1_contourImg_dilated' +
'.jpg'
cv2.imwrite(filename2, contourImg)


# ### dog_img

# In[308]:


#Initialize parameters

iterations = [0,0,2]
type_mask = [0,0,0]


# In[309]:


# determine each channel mask and the mask combined

BGR_masks = np.zeros(own2_img.shape)

for channel in range(own2_img.shape[-1]):

    own2_copy = copy.deepcopy(own2_img[:,:,channel])
```

```
    mask_channel = otsu_threshMasking(own2_copy.flatten(), iterations[channel],
type_mask[channel])
    BGR_masks[:,:,channel] = mask_channel.reshape((own2_img.shape[0],
own2_img.shape[1]))

    filename = 'D:\Purdue\ECE661_CV\HW6\HW6_outputs' + '\own2_channel_' +
str(channel) +'.jpg'
    plt.imsave(filename, BGR_masks[:,:,channel], cmap = 'gray' )

foreGround_mask = np.multiply(np.multiply(
BGR_masks[:,:,0],  BGR_masks[:,:,1]),  BGR_masks[:,:,2])
filename1 = 'D:\Purdue\ECE661_CV\HW6\HW6_outputs' + '\own2_foregroundMask' +
'.jpg'
cv2.imwrite(filename1, foreGround_mask*255)


# In[310]:


#Get the combined mask image displayed

foreGround_img = np.zeros(own2_img.shape)
for channel in range(own2_img.shape[-1]):
    foreGround_img[:, :, channel] = np.multiply(own2_img[:,:,channel],
foreGround_mask)

filename2 = 'D:\Purdue\ECE661_CV\HW6\HW6_outputs' + '\own2_foregroundImg' +
'.jpg'
cv2.imwrite(filename2, foreGround_img)


# In[311]:


#Get the contours

contourImg = get_Contours_cv(foreGround_mask)
filename2 = 'D:\Purdue\ECE661_CV\HW6\HW6_outputs' + '\own2_contourImg_mask' +
'.jpg'
cv2.imwrite(filename2, contourImg)


# In[312]:


#Perform erosion followed by dilation
```

```
kernel = np.ones((3,3))
eroded_img = cv2.erode(np.float32(foreGround_mask), kernel, iterations = 1)
dilated_img = cv2.dilate(eroded_img, kernel, iterations = 1)


# In[313]:


#get the countor image after the dilation

contourImg = get_Contours_cv(dilated_img)
filename2 = 'D:\Purdue\ECE661_CV\HW6\HW6_outputs' + '\own2_contourImg_dilated' +
'.jpg'
cv2.imwrite(filename2, contourImg)


# ## Texture- based segmentation

# In[341]:


#Compute variance for each window size with the neighbouring pixel values

def compute_varImg(gray_img, window_size):

    neighbours = []

    k = int(window_size/2)

    h = gray_img.shape[0]
    w = gray_img.shape[1]

    k_img = np.zeros((h + 2*k, w + 2*k))
    k_img[k : h+k, k : w+k] = gray_img


    for i in range(k, h+k):
        for j in range(k, w+k):
            neighbours.append(k_img[i-k : i+k+1, j-k : j+k+1].flatten())


    var_img = np.var(np.array(neighbours, dtype = np.float32), axis = 1)
    var_img = var_img.reshape(h,w)

    return var_img
```

```
# In[367]:


#Get the otsu threshold using sliding window based variances determined

def otsu_texture(img, iterations, type_mask, N = [3,5,7]):

    gray_img = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
    BGR_masks = np.zeros(img.shape)

    for i in range(len(N)):

        var_img = compute_varImg(gray_img, N[i])
        mask_channel = otsu_threshMasking(var_img.flatten(), iterations[i],
type_mask[i])
        BGR_masks[:,:,i] = mask_channel.reshape((img.shape[0], img.shape[1]))

        filename = 'D:\Purdue\ECE661_CV\HW6\HW6_outputs' + '\car_texture_window_'
+ str(i) +'.jpg'
        plt.imsave(filename, BGR_masks[:,:,i], cmap = 'gray' )

    texture_mask = np.multiply(np.multiply(
BGR_masks[:,:,0],  BGR_masks[:,:,1]),  BGR_masks[:,:,2])
    texture_mask = (texture_mask!=0)

    return texture_mask


# ### car_img

# In[491]:


#Initialize parameters

iterations = [1,3,1]
type_mask = [1,1,1]
N = [3,5,7]
invert_mask = 1


# In[492]:
```

```
# determine each channel mask and the mask combined

car_img_copy = copy.deepcopy(car_img)

texture_mask = otsu_texture(car_img_copy, iterations, type_mask, N)
texture_mask.reshape((car_img_copy.shape[0], car_img_copy.shape[1]))

if invert_mask == 1:
    texture_mask = (texture_mask == 0)

filename1 = 'D:\Purdue\ECE661_CV\HW6\HW6_outputs' + '\car_textureMask' + '.jpg'
cv2.imwrite(filename1, texture_mask*255)


# In[348]:


#Get the combined mask image displayed

texture_img = np.zeros(car_img_copy.shape)

for channel in range(car_img_copy.shape[-1]):
    texture_img[:, :, channel] = np.multiply(car_img_copy[:,:,channel],
texture_mask)

filename2 = 'D:\Purdue\ECE661_CV\HW6\HW6_outputs' + '\car_textureImg' + '.jpg'
cv2.imwrite(filename2, texture_img)


# In[349]:


#Get the contours

contourImg_texture = get_Contours_cv(texture_mask)
filename2 = 'D:\Purdue\ECE661_CV\HW6\HW6_outputs' +
'\car_contour_textureImg_maskImg' + '.jpg'
cv2.imwrite(filename2, contourImg_texture)


# In[350]:


#Perform erosion followed by dilation

kernel = np.ones((3,3))
```

```
eroded_img = cv2.erode(np.float32(texture_mask), kernel, iterations = 1)
dilated_img = cv2.dilate(eroded_img, kernel, iterations = 1)


# In[351]:


#get the contour image after the dilation

contourImg = get_Contours_cv(dilated_img)
filename2 = 'D:\Purdue\ECE661_CV\HW6\HW6_outputs' +
'\car_contour_textureImg_dilated' + '.jpg'
cv2.imwrite(filename2, contourImg)


# ### cat_img
#

# In[358]:


#Initialize parameters

iterations = [1,3,1]
type_mask = [1,1,1]
N = [3,5,7]
invert_mask = 1


# In[359]:


# determine each channel mask and the mask combined

cat_img_copy = copy.deepcopy(cat_img)

texture_mask = otsu_texture(cat_img_copy, iterations, type_mask, N)
texture_mask.reshape((cat_img_copy.shape[0], cat_img_copy.shape[1]))

if invert_mask == 1:
    texture_mask = (texture_mask == 0)

filename1 = 'D:\Purdue\ECE661_CV\HW6\HW6_outputs' + '\cat_textureMask' + '.jpg'
cv2.imwrite(filename1, texture_mask*255)
```

```
# In[360]:


#Get the combined mask image displayed

texture_img = np.zeros(cat_img_copy.shape)

for channel in range(cat_img_copy.shape[-1]):
    texture_img[:, :, channel] = np.multiply(cat_img_copy[:,:,channel],
texture_mask)

filename2 = 'D:\Purdue\ECE661_CV\HW6\HW6_outputs' + '\cat_textureImg' + '.jpg'
cv2.imwrite(filename2, texture_img)


# In[361]:


#Get the contours

contourImg_texture = get_Contours_cv(texture_mask)
filename2 = 'D:\Purdue\ECE661_CV\HW6\HW6_outputs' +
'\cat_contour_textureImg_maskImg' + '.jpg'
cv2.imwrite(filename2, contourImg_texture)


# In[362]:


#Perform erosion followed by dilation

kernel = np.ones((3,3))
eroded_img = cv2.erode(np.float32(texture_mask), kernel, iterations = 1)
dilated_img = cv2.dilate(eroded_img, kernel, iterations = 1)


# In[363]:


#get the contour image after the dilation

contourImg = get_Contours_cv(dilated_img)
filename2 = 'D:\Purdue\ECE661_CV\HW6\HW6_outputs' +
'\cat_contour_textureImg_dilated' + '.jpg'
cv2.imwrite(filename2, contourImg)
```

```
# ### bird_img

# In[485]:


#Initialize parameters

iterations = [3,3,6]
type_mask = [1,1,1]
N = [3,5,7]
invert_mask = 1


# In[486]:


# determine each channel mask and the mask combined

own1_img_copy = copy.deepcopy(own1_img)

texture_mask = otsu_texture(own1_img_copy, iterations, type_mask, N)
texture_mask.reshape((own1_img_copy.shape[0], own1_img_copy.shape[1]))

if invert_mask == 1:
    texture_mask = (texture_mask == 0)

filename1 = 'D:\Purdue\ECE661_CV\HW6\HW6_outputs' + '\own1_textureMask' + '.jpg'
cv2.imwrite(filename1, texture_mask*255)


# In[487]:


#Get the combined mask image displayed

texture_img = np.zeros(own1_img_copy.shape)

for channel in range(own1_img_copy.shape[-1]):
    texture_img[:, :, channel] = np.multiply(own1_img_copy[:,:,channel],
texture_mask)

filename2 = 'D:\Purdue\ECE661_CV\HW6\HW6_outputs' + '\own1_textureImg' + '.jpg'
cv2.imwrite(filename2, texture_img)


# In[488]:
```

```
#Get the contours

contourImg_texture = get_Contours_cv(texture_mask)
filename2 = 'D:\Purdue\ECE661_CV\HW6\HW6_outputs' +
'\own1_contour_textureImg_maskImg' + '.jpg'
cv2.imwrite(filename2, contourImg_texture)


# In[489]:


#Perform erosion followed by dilation

kernel = np.ones((3,3))
eroded_img = cv2.erode(np.float32(texture_mask), kernel, iterations = 1)
dilated_img = cv2.dilate(eroded_img, kernel, iterations = 1)


# In[490]:


#get the contour image after the dilation

contourImg = get_Contours_cv(dilated_img)
filename2 = 'D:\Purdue\ECE661_CV\HW6\HW6_outputs' +
'\own1_contour_textureImg_dilated' + '.jpg'
cv2.imwrite(filename2, contourImg)


# ### dog_img

# In[440]:


#Initialize parameters

terations = [6,3,3]
type_mask = [1,1,1]
N = [3,5,7]
invert_mask = 1


# In[441]:
```

```
# determine each channel mask and the mask combined

own2_img_copy = copy.deepcopy(own2_img)

texture_mask = otsu_texture(own2_img_copy, iterations, type_mask, N)
texture_mask.reshape((own2_img_copy.shape[0], own2_img_copy.shape[1]))

if invert_mask == 1:
    texture_mask = (texture_mask == 0)

filename1 = 'D:\Purdue\ECE661_CV\HW6\HW6_outputs' + '\own2_textureMask' + '.jpg'
cv2.imwrite(filename1, texture_mask*255)


# In[442]:


#Get the combined mask image displayed

texture_img = np.zeros(own2_img_copy.shape)

for channel in range(own2_img_copy.shape[-1]):
    texture_img[:, :, channel] = np.multiply(own2_img_copy[:,:,channel],
texture_mask)

filename2 = 'D:\Purdue\ECE661_CV\HW6\HW6_outputs' + '\own2_textureImg' + '.jpg'
cv2.imwrite(filename2, texture_img)


# In[443]:


#Get the contours

contourImg_texture = get_Contours_cv(texture_mask)
filename2 = 'D:\Purdue\ECE661_CV\HW6\HW6_outputs' +
'\own2_contour_textureImg_maskImg' + '.jpg'
cv2.imwrite(filename2, contourImg_texture)


# In[444]:


#Perform erosion followed by dilation

kernel = np.ones((3,3))
```

```
eroded_img = cv2.erode(np.float32(texture_mask), kernel, iterations = 1)
dilated_img = cv2.dilate(eroded_img, kernel, iterations = 1)


# In[445]:


#get the contour image after the dilation

contourImg = get_Contours_cv(dilated_img)
filename2 = 'D:\Purdue\ECE661_CV\HW6\HW6_outputs' +
'\own2_contour_textureImg_dilated' + '.jpg'
cv2.imwrite(filename2, contourImg)
```

*****