ECE 661 Computer Vision (2018 Fall)
Homework 6

Runzhe Zhang

October 22, 2018

1. Introduction

The homework deals with image segmentation. In particular, we are provided with three images and you need to separate out the foreground from the background in these images. Subsequently, we will extract the contour of the segmented output.

For each image we need to know what is the desired foreground that we would like you to segment out of the image. The foreground for the first image is the red lighthouse. The foreground for the second image is the head and body of the cute baby. The foreground for the third image is the whole body of the jumping man.

In this homework, we apply Otsu's method to perform image segmentation. The RGB segmentation is achieved by applying the Otsu's algorithm to the three RGB color channel separately, and then combine the segmentation results to get the final image segmentation.

The texture-based segmentation is achieved by first computing the texture features with three different windows, apply the Otsu's algorithm to the three texture features separately, and then combine the segmentation results to get the final image segmentation.

After the segmentation is done, we use contour extraction techniques to get the boundary of the foreground.

2. OTSU Method

Otsu's method is used to automatically perform clustering-based image thresholding or, the reduction of a graylevel image to a binary image. The algorithm assumes that the image contains two classes of pixels following bi-modal histogram (foreground pixels and background pixels), it then calculates the optimum threshold separating the two classes so that their combined spread (intra-class variance) is minimal, or equivalently (because the sum of pairwise squared distances is constant), so that their inter-class variance is maximal (Reference: Wikipedia).

In Otsu's method we exhaustively search for the threshold that minimizes the intra-class variance (the variance within the class), defined as a weighted sum of variances of the two classes:

$$\sigma_{\omega}^2(t) = \omega_0(t)\sigma_0^2(t) + \omega_1(t)\sigma_1^2(t)$$

Weights ω_0 andω_1 are the probabilities of the two classes separated by a threshold t ,and σ_0^2 and σ_1^2 are variances of these two classes. The class probability $\omega_{0,1}(t)$ is computed from the L bins of the histogram:

$$\omega_0(t) = \sum_{i=0}^{t-1} p(i)$$

$$\omega_1(t) = \sum_{i=t}^{L-1} p(i)$$

Otsu shows that minimizing the intra-class variance is the same as maximizing inter-class variance:

$$\sigma_b^2(t) = \sigma^2 - \sigma_{\omega}^2(t) = \omega_0(t)\omega_1(t)[\mu_0(t) - \mu_1(t)]^2$$

which is expressed in terms of class probabilities ω and class means μ, while the class mean $\mu_{0,1,T}(t)$ is:

$$\mu_0(t) = \frac{\sum_{i=0}^{t-1} i\,p(i)}{\omega_0(t)}$$

$$\mu_1(t) = \frac{\sum_{i=t}^{L-1} i\,p(i)}{\omega_1(t)}$$

$$\mu_T = \sum_{i=0}^{L-1} i\,p(i)$$

The following relations can be easily verified:

$$\omega_0\mu_0 + \omega_1\mu_1 = \mu_T$$

$$\omega_0 + \omega_1 = 1$$

The class probabilities and class means can be computed iteratively. This idea yields an effective algorithm.

Algorithm:

1. Compute histogram and probabilities of each intensity level;
2. Set up initial$\omega_i(0)$ and $\mu_i(0)$
3. Step through all possible thresholds $t = 1$ maximum intensity
   (a) Update ω_i and μ_i;
   (b) Compute $\sigma_b^2(t)$;
4. Desired threshold corresponds to the maximum $\sigma_b^2(t)$

After these process, we can get the OTSU threshold to segment the image.

3. RGB Image Segmentation

The image segmentation can be achieved by applying Otsu's method to the three RGB color channels separately, and then combine the segmentation results to get final segmentation of the image. The procedure is described as follows.

1. Separate the three RGB color channels and convert them into three gray- scale images;
2. Construct the mask for each channel using Otsu's method;
3. Combine the three masks by logical operator AND. To get a better segmentation result, the combination logic is chosen depending on the image.

4. Texture-based Image Segmentation

The texture-based segmentation method is similar to what we have done in Section 3. The only difference is that we use three texture feature channels as the input to Otsu's method instead of using the three RGB channels. The procedure is described as follows.

1. Convert the image to gray-scale image;
2. Generate a new gray-scale image whose pixel value is the variance of the gray-scale values in a $N \times N$ window around the corresponding pixels of the original gray-scale image.
3. Do Step 1 to Step 2 for three different window sizes $N = 3, N = 5, and N = 7$. These three gray-scale images are considered to be the texture features of the original image.
4. Treat the three texture features as the three channels of the original image, apply Otsu's method separately to get the three masks.
5. Combine the three masks by logical operator AND. To get a better segmentation result, the combination logic is chosen depending on the image. It should be noted that the values in the texture feature represent the variance.

After these process, we can get the texture-based image segmentation result.

5. Contour Extraction

After the segmentation is done, the contour can be extract for better visualization. My contour extraction algorithm is implemented based on 8-neighbors. The foreground corresponds to the pixel values equal to 1 in the overall mask, while the background corresponds to the pixel values equal to 0 in the overall mask. For each pixel in the overall mask:

1. If the pixel value is 0, then it is not selected as thecontourpoint;
2. If the pixel value is 1, and all its 8-neighbors are 1, then it is not selected as the contour point;
3. If the pixel value is 1, and not all of its 8-neighbors are 1, then it is selected as the contour point.

6. Observations

According to the results,the texture-based method works worse in the smooth objective image, while the RGB segmentation method works better in the smooth images. It is reasonable since the texture-based method is suitable when our foreground image contains more textures than the background image.

The contour extraction method could also influence the result. Right now my contour extraction algorithm would depict the boundary of those tiny foreground regions. Use other contour extraction algorithms may improve the results.

The performance of the segmentation results depends highly on the characteristic of the original image. We should select the appropriate segmentation method based on the input image.

7. Result

7.1 Image: baby

Figure 1: The original input image

Figure 2: The R layer of the original RGB image

Figure 3: The G layer of the original RGB image

Figure 4: The B layer of the original RGB image

Figure 5: The R layer OTSU result

Figure 6: The G layer OTSU result

Figure 7: The B layer OTSU result

Figure 8: The combination of the RGB three layers OTSU result

Figure 9: The contour result based on the OTSU method(RED line is the contour)

Figure 10: The foreground based on the contour detection

Figure 11: The OTSU result for 3*3 window texture method

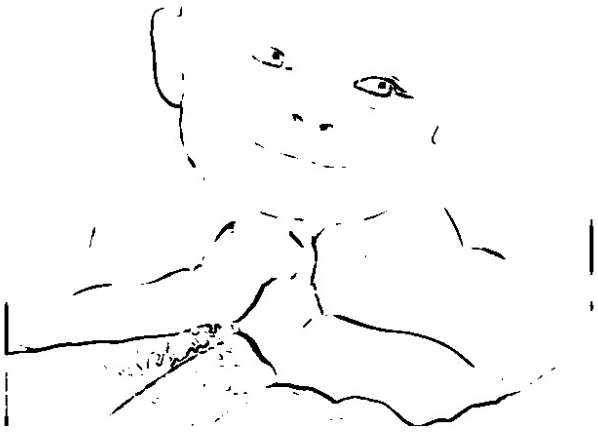

Figure 12: The OTSU result for 5*5 window texture method

Figure 13: The OTSU result for 7*7 window texture method

Figure 14: The combination of texture OTSU result

7.2 Image: Light House

Figure 15: The original input image

Figure 16: The R layer of the original RGB image

Figure 17: The G layer of the original RGB image

Figure 18: The B layer of the original RGB image

Figure 19: The R layer OTSU result

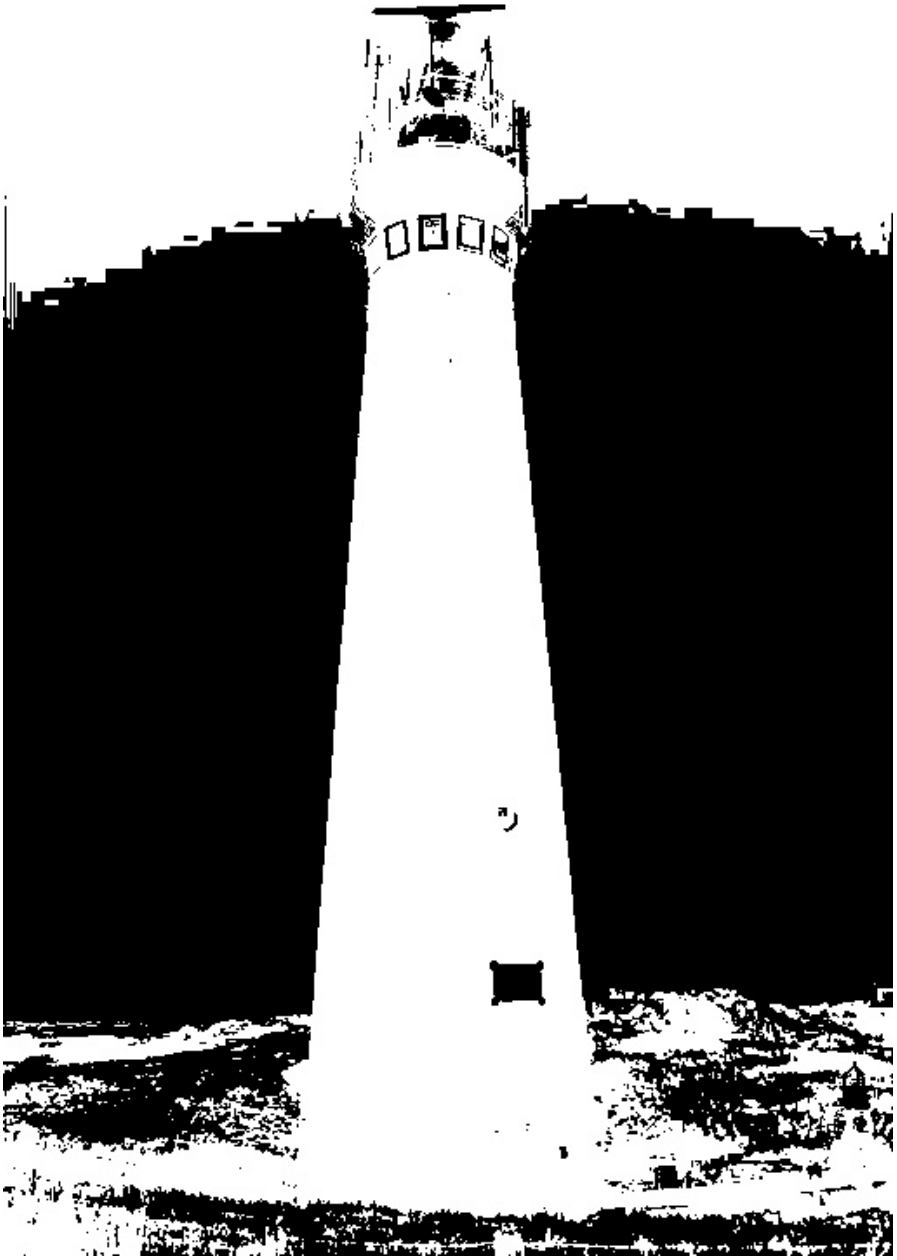

Figure 20: The G layer OTSU result

Figure 21: The B layer OTSU result

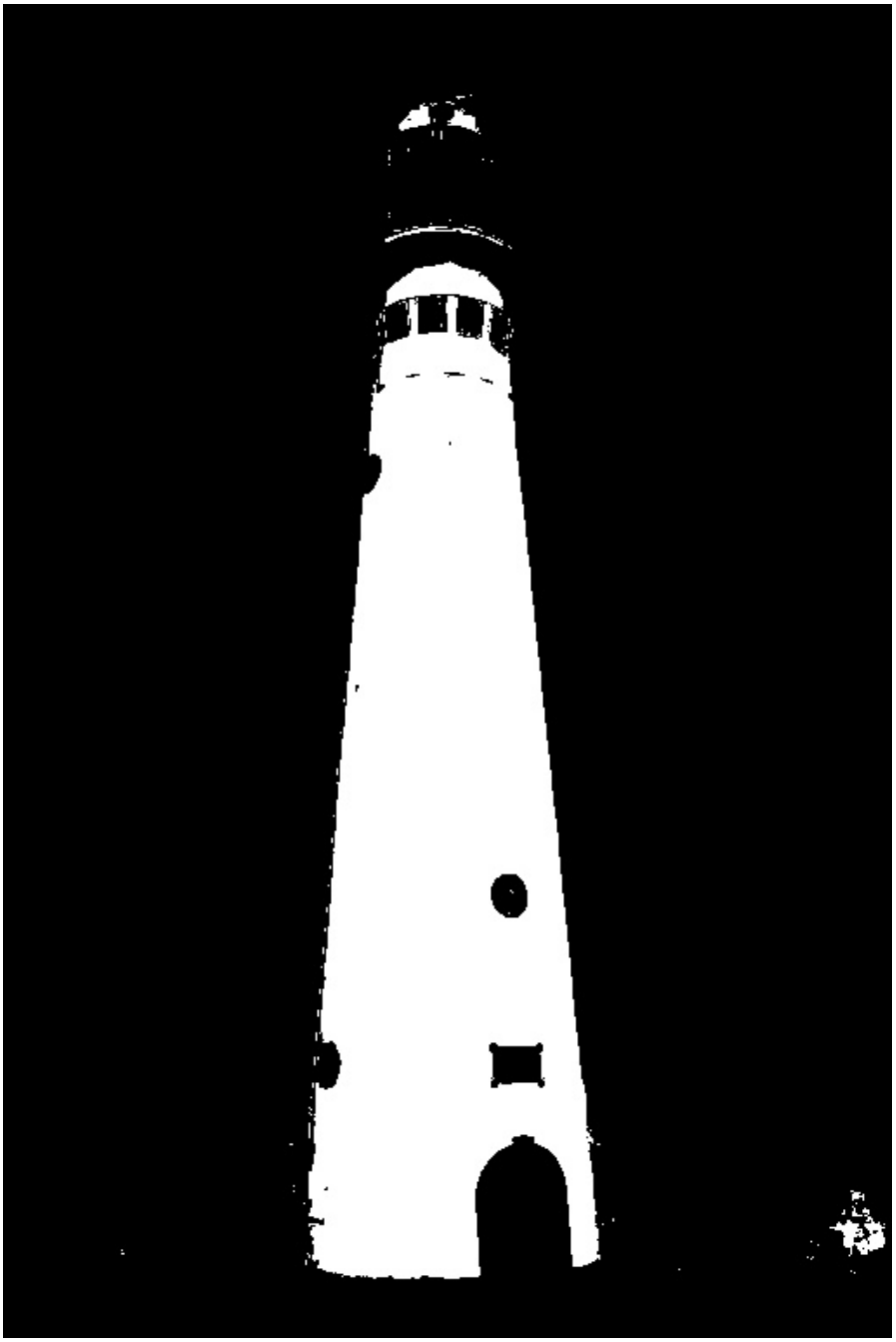

Figure 22: The combination of the RGB three layers OTSU result

Figure 23: The contour result based on the OTSU method(RED line is the contour)

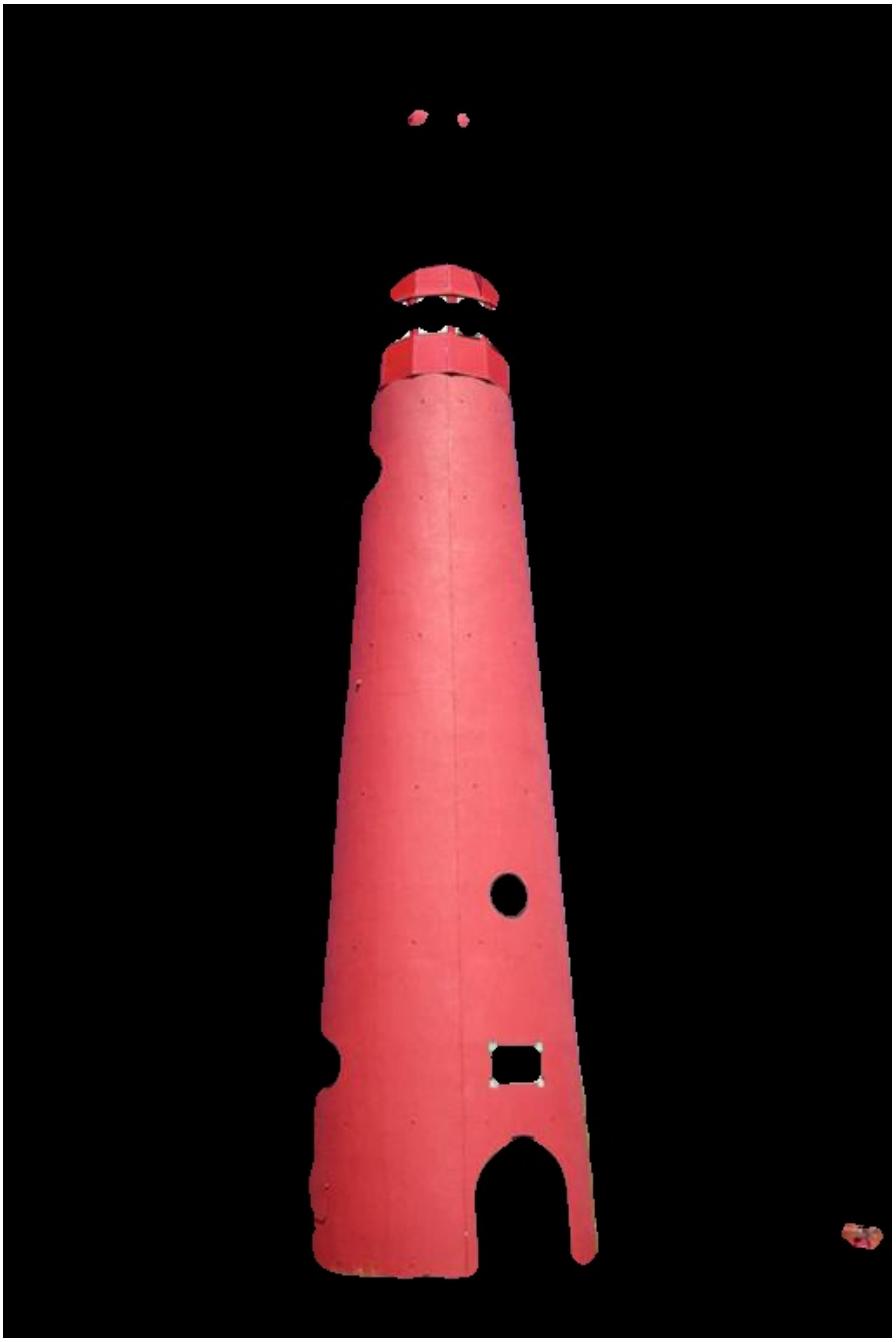

Figure 24: The foreground based on the contour detection

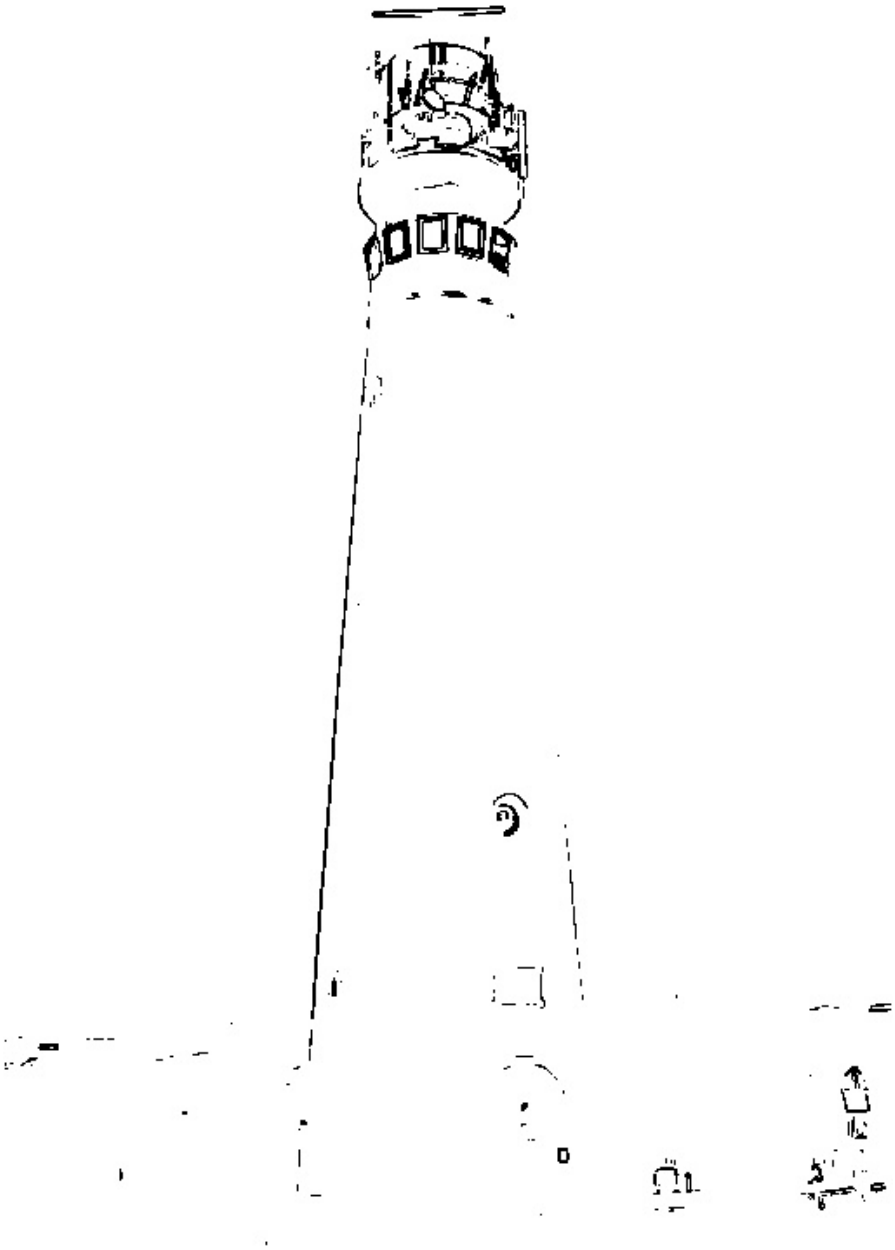

Figure 25: The OTSU result for 3*3 window texture method

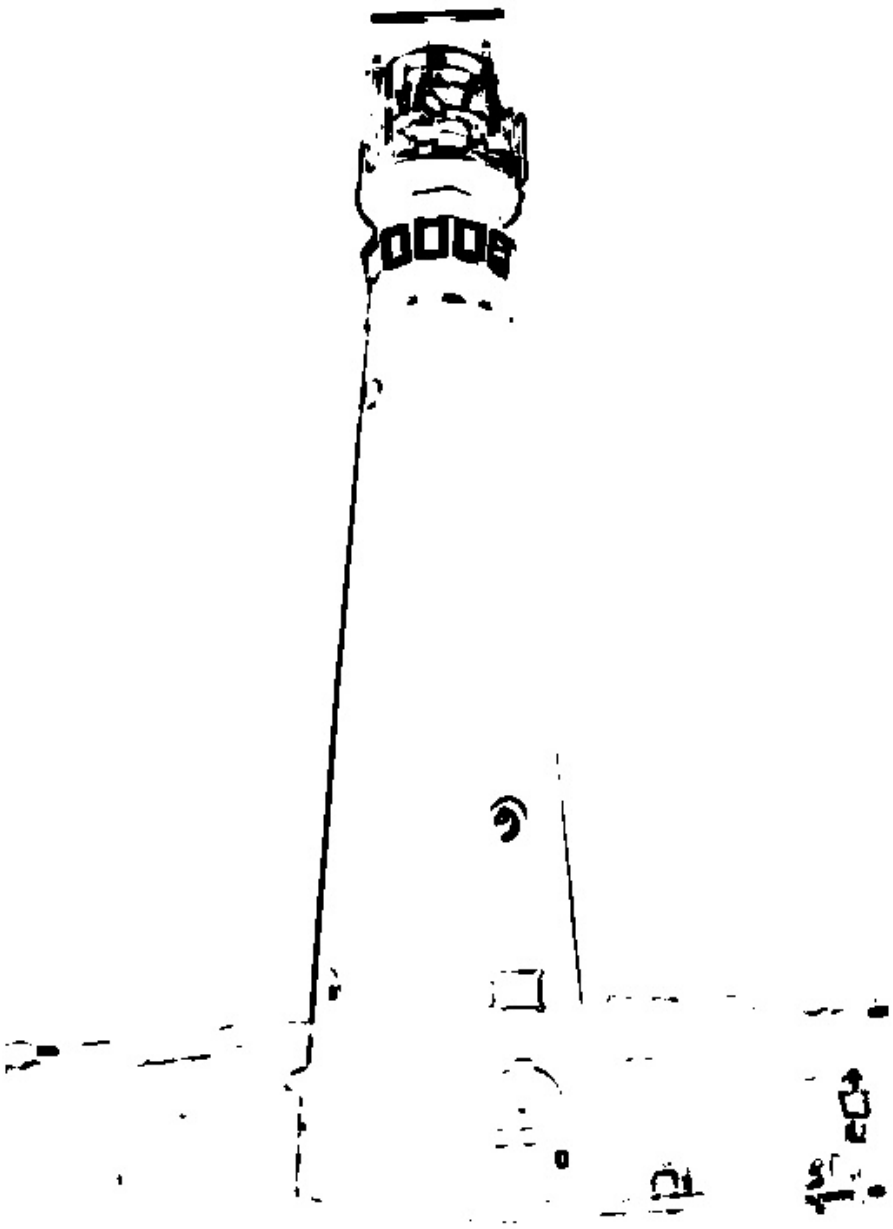

Figure 26: The OTSU result for 5*5 window texture method

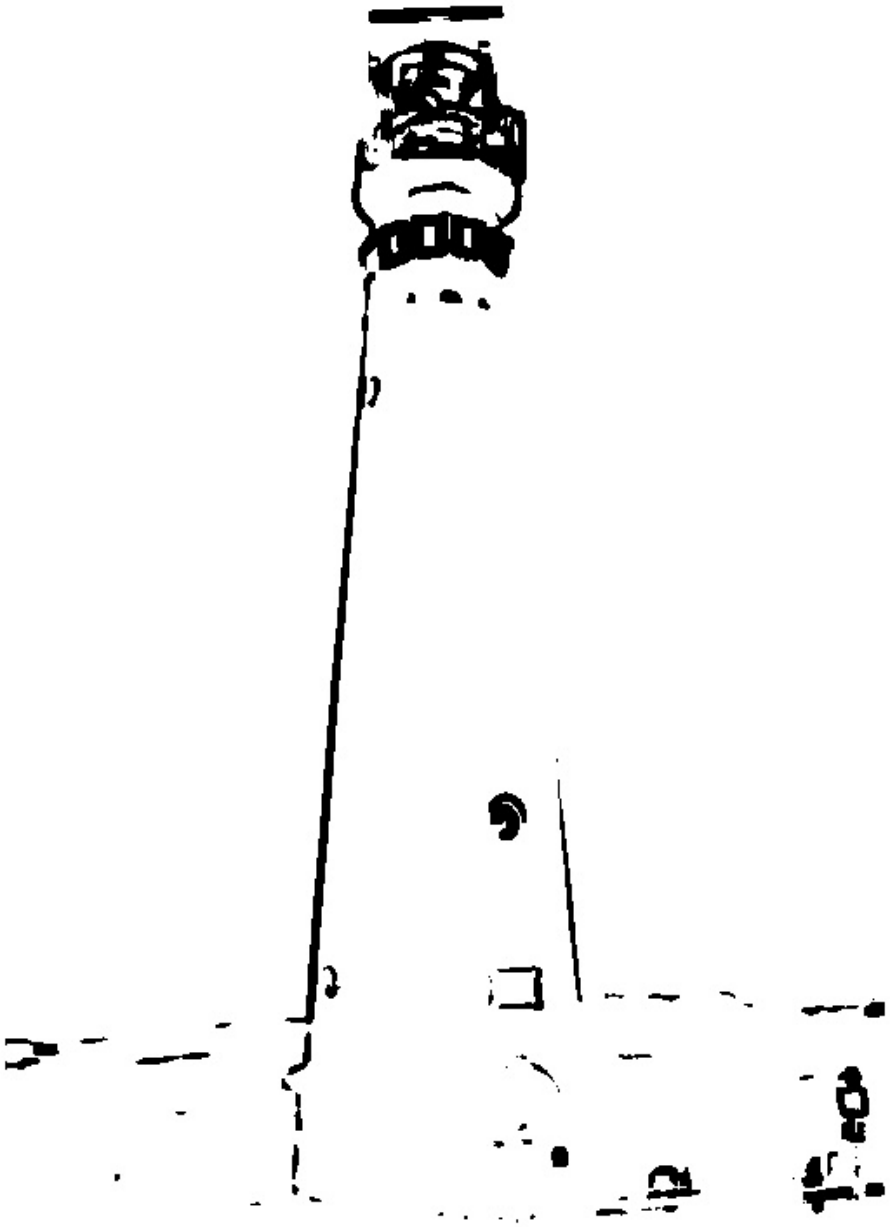

Figure 27: The OTSU result for 7*7 window texture method

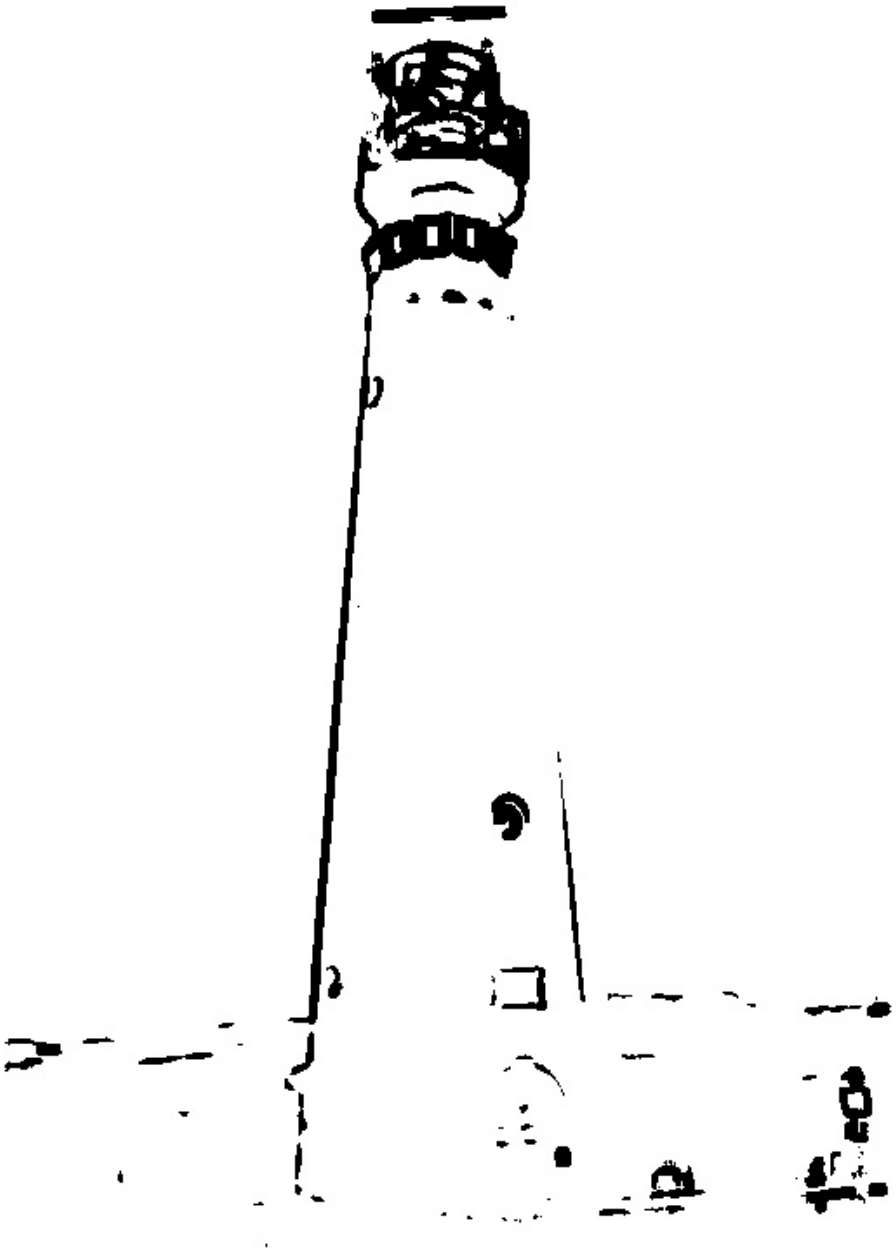

Figure 28: The combination of texture OTSU result

7.3 Image: Ski

Figure 29: The original input image

Figure 30: The R layer of the original RGB image

Figure 31: The G layer of the original RGB image

Figure 32: The B layer of the original RGB image

Figure 33: The R layer OTSU result

Figure 34: The G layer OTSU result

Figure 35: The B layer OTSU result

Figure 36: The combination of the RGB three layers OTSU result

Figure 37: The contour result based on the OTSU method(RED line is the contour)

Figure 38: The foreground based on the contour detection

Figure 39: The OTSU result for 3*3 window texture method

Figure 40: The OTSU result for 5*5 window texture method

Figure 41: The OTSU result for 7*7 window texture method

Figure 42: The combination of texture OTSU result

8. Code

```
#####
#####    ECE 661 Computer Vision(2018 Fall) Homework 6
#####         Oct 11 2018      Runzhe Zhang
#####
#########################################################

import cv2
import numpy as np
from matplotlib import pyplot as plt

"""
   Description:
       implementation of the otsu algorthm
   Input:
       imggray --- a grayscale image to which find threshold on
       minval, maxval --- and minimum and maxmim of grascale values to find
    threshold in
   Output:
       otsu_thresh --- the found threshold
   Usage:
       otsu_thresh =  otsumethod(imggray, minval, maxval)
"""

def otsumethod(imggray, minval, maxval):

    size1, size2 = imggray.shape
    numpixel = size1 * size2
    L = maxval - minval
    hist = cv2.calcHist([np.uint8(imggray)], [0], None, [L], [minval, maxval])

    # for displaying his only using plt
    # plt.hist(imggray.ravel(),L,[minval,maxval]);plt.show()

    propt = hist/ numpixel
    var_b = np.zeros((L,1))
    w0 = graysum0 = 0
    graylev = np.reshape(np.add(range(L), minval + 1), (L, 1))
    graylev_propt = np.multiply(graylev, propt)
    mean_total = np.sum(graylev_propt)
    var_b_max = 0
    otsu_thresh = 0

    for i in range(L):
        w0 += propt[i]
        w1 = 1 - w0
        graysum0 += graylev_propt[i]
        mean0 = graysum0 / w0

        if w1 == 0:
            var_b[i] = 0
        else:
```

```
            mean1 = (mean_total - graysum0) / w1
            var_b[i] = w0 * w1 * ( mean0 - mean1 ) ** 2

        if var_b[i] > var_b_max:
            var_b_max = var_b[i]
            otsu_thresh = i + minval
    print("Found OTSU threshold: %i" %(otsu_thresh))

    return otsu_thresh


def textureimage(img_gray, blocksize):

    size1, size2 = img_gray.shape
    texture_result = np.zeros((img_gray.shape))
    half_block = np.uint8((blocksize - 1)/2)

    for i in range(half_block, size1-half_block):
        for j in range(half_block, size2-half_block):
            kernal = img_gray[i-half_block:i+half_block+1,j-half_block:j+half_block
    +1]
            texture_result[i,j] = np.mean((kernal-np.mean(kernal))**2)

    texture_resultf = np.uint8(texture_result/texture_result.max()*255)
    # hist = cv2.calcHist([texture_resultf], [0], None, [256-0], [0, 256])
    # plt.hist(texture_resultf.ravel(), 256-0, [0, 256]);
    # plt.show()

    return texture_resultf


def find_countour(img, connectivity = 4):
    if (connectivity != 4) and (connectivity != 8):
        print('connectivity is not 4 or 8')
        return -1
    h,w = img.shape
    contour_mask = np.zeros((h,w))
    for i in range(1,w-1):
        for j in range(1,h-1):
            if img[j,i] == 255:
                if connectivity == 8:
                    for ii in range(3):
                        for jj in range(3):
                            if img[j+jj, i+ii] == 0:
                                contour_mask[j,i] = 255
                elif connectivity == 4:
                    if (img[j+1, i] == 0 or img[j-1, i] == 0 or img[j, i+1] == 0 or
     img[j, i-1] == 0):
                        contour_mask[j,i] = 255
    return contour_mask
```

```
102
103 def main():
104     img = int(input('Please choose image (baby--1; lighthouse--2; ski--3): '))
105     method = int(input('Please choose method (RGB_OTSU--1; Texture_OTSU--2): '))
106
107     if img == 1:
108         imgBGR = cv2.imread('baby.jpg', cv2.IMREAD_COLOR)
109         img_gray = cv2.imread('baby.jpg', cv2.IMREAD_GRAYSCALE)
110     elif img == 2:
111         imgBGR = cv2.imread('lighthouse.jpg', cv2.IMREAD_COLOR)
112         img_gray = cv2.imread('lighthouse.jpg', cv2.IMREAD_GRAYSCALE)
113     elif img == 3:
114         imgBGR = cv2.imread('ski.jpg', cv2.IMREAD_COLOR)
115         img_gray = cv2.imread('ski.jpg', cv2.IMREAD_GRAYSCALE)
116     else:
117         print('You input the wrong number for choosing image, please input the real
      number (1, 2 ,or 3) and run again.')
118
119     # cv2.imshow('imgRGB', img_gray)
120     # cv2.waitKey()
121
122     if method != 1 and method !=2:
123         print('Please choose the right method(RGB_OTSU--1; Texture_OTSU--2) and run
      again.')
124
125     if method == 1:
126
127         mask11 = np.zeros((imgBGR.shape))
128         mask12 = np.zeros((imgBGR.shape))
129
130         for iteration in range(1,3):
131
132             # First iteration to get foreground
133             for i in range(3):
134                 img_layer = imgBGR[:, :, i]
135                 img_layer_blur = cv2.GaussianBlur(img_layer, (5, 5), 0)
136                 cv2.imwrite(str(img) + 'layer' + str(i) + 'itration' + str(
    iteration) + '.jpg', img_layer_blur)
137
138                 # OTSU iteration 1
139                 otsu_thresh1 = otsumethod(img_layer_blur, 0, 256)
140                 mask11[:, :, i] = img_layer <= otsu_thresh1
141                 otsucv_thresh1, otsucv_result1 = cv2.threshold(img_layer_blur, 0,
    255,
142                                                                cv2.THRESH_BINARY +
    cv2.THRESH_OTSU)
143                 cv2.imwrite(str(img) + 'mask' + str(i) + 'itration' + str(iteration
    ) + '.jpg', mask11[:, :, i] * 255)
144
145                 # high_img = np.array(mask1[:, :, i] * img_layer, np.uint8)
146                 # low_img = np.logical_not(mask1[:, :, i]) * img_layer
147                 # cv2.imshow('subimage, high value', high_img)
148                 # cv2.imshow('subimage, low value', low_img)
```

```
149                 # cv2.waitKey(0)
150
151             # combined_mask = np.array(mask[:,:,0] * 255, np.uint8)
152             combined_mask11 = np.array(
153                 np.logical_and(np.logical_and(mask11[:, :, 0], mask11[:, :, 1]),
    mask11[:, :, 2]) * 255, np.uint8)
154             # cv2.imshow('combined_mask', combined_mask)
155             cv2.imwrite(str(img) + 'RGBcombinedmaskIteration' + str(iteration) + '.
    jpg', combined_mask11)
156             # cv2.waitKey(0)
157
158             # combined_mask = cv2.imread('combined_mask.jpg', 0)
159             combined_mask112 = cv2.medianBlur(combined_mask11, 13)
160             # cv2.imshow('combined_mask', combined_mask)
161             # cv2.imshow('combined_mask2', combined_mask2)
162             mask11 = combined_mask112 > 240
163             clean_mask11 = np.array(mask11 * 255, np.uint8)
164             # cv2.imshow('clean_mask', clean_mask)
165             # cv2.waitKey(0)
166
167             contour11 = find_countour(clean_mask11)
168             # cv2.imshow('contour', contour)
169             # cv2.imwrite('contour.jpg', contour)
170             # cv2.waitKey(0)
171
172             b, g, r = cv2.split(imgBGR)
173             mask_logcial111 = np.logical_and(np.logical_not(clean_mask11), 1)
174             b[mask_logcial111] = 0
175             g[mask_logcial111] = 0
176             r[mask_logcial111] = 0
177             imgBGR2 = cv2.merge([b, g, r])
178             # cv2.imshow('foreground', imgBGR2)
179             # cv2.waitKey(0)
180             cv2.imwrite(str(img) + 'RGBforegroundIteration' + str(iteration) + '.
    jpg', imgBGR2)
181
182             b, g, r = cv2.split(imgBGR)
183             mask_logcial112 = np.logical_and(contour11, 1)
184             b[mask_logcial112] = 0
185             g[mask_logcial112] = 255
186             r[mask_logcial112] = 0
187             imgBGR2 = cv2.merge([b, g, r])
188             # cv2.imshow('contour', imgBGR2)
189             # cv2.waitKey(0)
190             cv2.imwrite(str(img) + 'RGBcontourIteration' + str(iteration) + '.jpg',
     imgBGR2)
191
192
193     elif method == 2:
194
195         mask21 = np.zeros((imgBGR.shape))
196         imgtexture = np.zeros((imgBGR.shape))
197
```

```
198         for i in range(3):
199             imgtexture[: ,: , i] = textureimage(img_gray, (i+1)*2+1)
200             otsu_thresh1 = otsumethod(imgtexture[: ,: , i], 0, 256)
201             mask21[:, :, i] = imgtexture[: ,: , i] <= otsu_thresh1
202             # otsucv_thresh1, otsucv_result1 = cv2.threshold(imgtexture[: ,: , i],
    0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
203             cv2.imwrite(str(img) + 'mask' + str(i) + 'texture.jpg',  mask21[:, :, i
    ] * 255)
204
205
206         # combined_mask = np.array(mask[:,:,0] * 255, np.uint8)
207         combined_mask21 = np.array(np.logical_and(np.logical_and(mask21[:,:,0],
    mask21[:,:,1]), mask21[:,:,2]) * 255, np.uint8)
208         # cv2.imshow('combined_mask', combined_mask)
209         cv2.imwrite(str(img) + 'texturecombinedmask.jpg', combined_mask21)
210         # cv2.waitKey(0)
211
212         # combined_mask = cv2.imread('combined_mask.jpg', 0)
213         combined_mask2 = cv2.medianBlur(combined_mask21, 13)
214         # cv2.imshow('combined_mask', combined_mask)
215         # cv2.imshow('combined_mask2', combined_mask2)
216         mask = combined_mask2 > 240
217         clean_mask = np.array(mask * 255, np.uint8)
218         # cv2.imshow('clean_mask', clean_mask)
219         # cv2.waitKey(0)
220
221         contour = find_countour(clean_mask)
222         # cv2.imshow('contour', contour)
223         # cv2.imwrite('contour.jpg', contour)
224         # cv2.waitKey(0)
225
226         # imgBGR = cv2.imread('pic1.jpg', 1)
227         b, g, r = cv2.split(imgBGR)
228         mask_logcial = np.logical_and(np.logical_not(clean_mask), 1)
229         b[mask_logcial] = 0
230         g[mask_logcial] = 0
231         r[mask_logcial] = 0
232         imgBGR2 = cv2.merge([b, g, r])
233         # cv2.imshow('foreground', imgBGR2)
234         # cv2.waitKey(0)
235         cv2.imwrite(str(img) +'textureforeground.jpg', imgBGR2)
236
237         b, g, r = cv2.split(imgBGR)
238         mask_logcial = np.logical_and(contour, 1)
239         b[mask_logcial] = 0
240         g[mask_logcial] = 0
241         r[mask_logcial] = 255
242         imgBGR2 = cv2.merge([b, g, r])
243         # cv2.imshow('contour', imgBGR2)
244         # cv2.waitKey(0)
245         cv2.imwrite(str(img) +'texturecontour.jpg', imgBGR2)
246
247
```

```
248
249 if __name__ == '__main__':
250     debug = 1
251     main()
```