Rehana Mahfuz (rmahfuz@purdue.edu)
ECE 661 HW06

Otsu's algorithm

Given a two-dimensional image with L gray levels, Otsu's algorithm finds the gray level k that separates the background from the foreground.

Let the number of pixels belonging to gray level i be n_i, for all i belonging to [1,L].

The probability mass function p_i is calculated for all gray levels as $p_i = n_i/N$, for all i belonging to [1,L]. N is the total number of pixels, which is the height*width of the image.

Let C_0 and C_1 be the two classes (foreground and background, or vice-versa) that are separated by k. All pixels with gray levels lesser than or equal to k belong to C_0, and the rest belong to C_1.

We want to find the k that maximizes the between-class variance, given by

$\boldsymbol{\sigma}^2_B = \boldsymbol{\omega}_0 \, \boldsymbol{\omega}_1(\mu_1 - \mu_0)^2$

Where $\boldsymbol{\omega}_0$ is the sum of all p_i for all 1 <= i <= k, and $\boldsymbol{\omega}_1 = 1 - \boldsymbol{\omega}_0$,

$\mu_0 = \sum_{i=1}^{k} i.\frac{p_i}{\omega_0}$

and $\mu_1 = \sum_{i=k+1}^{L} i.\frac{p_i}{\omega_1}$

The range of k over which the maximum is searched for can be reduced to all k for which $\boldsymbol{\omega}_0$ is in the interval (0,1). Once this range of k is found, the between-class variance is calculated for all k's within this range to find the maximum. The k corresponding to the maximum $\boldsymbol{\sigma}^2_B$ is the gray level that separates the background and foreground.

Whether C_0 is background or foreground is a decision to be made based on the image. By default C_0 was foreground, but that did not work for the ski image. So only for the ski image, C_0 was background.

Image segmentation using RGB values

We separate the red, green and blue channels as individual 2-dimensional images. Then we apply Otsu's algorithm to each channel separately to find the foreground pixels. Finally, we create another mask where we consider a pixel to belong to the foreground only if it belongs to the foreground in all the three channels. This approach may be modified based on the image. For example, for the ski image, segmentation using only blue yielded superior results than segmentation using all three colors, because segmentation using other colors were excluding part of the human from the foreground.

If part of the background is also included in the foreground the first time this segmentation is performed, then the segmentation is performed again taking as input the foreground that was output by the first run of the algorithm. This process is continued till a satisfactory foreground is achieved. Since this is a repetitive process, each performance of the algorithm is often termed as an iteration.

Image segmentation using texture-based features

First the image is converted into grayscale using the cvtColor function from OpenCV using the code BGR2GRAY. Then, in another image which is a copy of the original image, each pixel is replaced by the variance of the all pixels in an NxN window surrounding the image. If the pixel is on the edge of the image such that the NxN window exceeds the bounds of the image, then this replacement is omitted. This is done for window sizes of N = 3, N = 5, and N = 7. Now this three-dimensional characterization of the image is treated the same way as the RGB channels were treated. The foreground is calculated for each of the three channels separately. A pixel is considered to belong to the foreground only if it belongs to the foreground (as indicated by Otsu's algorithm) in each of the three channels.

If a satisfactory foreground is not achieved, the algorithm may be repeated multiple times.

Contour extraction algorithm

Each pixel of the segmented image was looked at (except the border pixels), and the pixel was classified as being part of the contour if it met the following criteria:

1) The pixel is non-zero (part of foreground)
2) At least one of the pixel's 8 neighbors is zero (part of background), where the 8 neighbors are top, top right, right, bottom right, bottom, bottom left, left, top left.

Performance of segmentation algorithm and usefulness of features used

Lighthouse image

For the lighthouse, color-based segmentation was much more effective than texture-based segmentation. One iteration of RGB segmentation yielded a clear red foreground. This is because the foreground is very conspicuously separated from the background by color. But the texture of the lighthouse (foreground) and texture of the sky (part of background) are both smooth, so there is no way of segmenting the lighthouse from the sky by texture. Even after segmenting by texture, both the sky and lighthouse remained. By the third iteration, the grass was mostly cleared out of the foreground because of its texture, so that was the optimal iteration to stop at before losing too much of the foreground as well. After the third iteration of texture-based segmentation, I simply removed the sky from the image by removing any pixel for which the blue is greater than 100, but red is less than 190. That was the only way to remove the sky if I use texture-based segmentation.

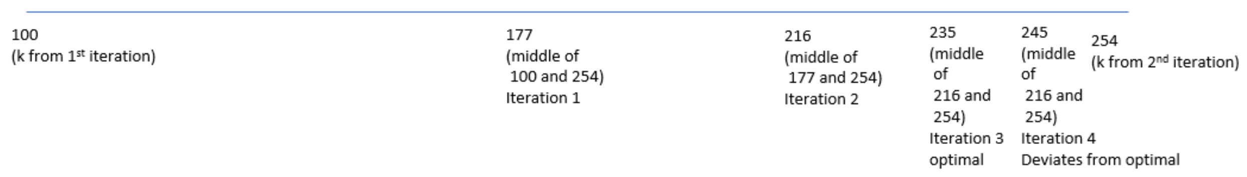
Ski image

For the ski image, the first iteration of RGB segmentation was not able to remove the blue background at the top of the image because it is quite dark: its darkness is comparable to the darkness of the foreground. The second iteration removed almost all of the background, but also removed certain parts of the human light in color. Letting go certain parts of the human is the price I had to pay to exclude much of the blue background.

For texture-based segmentation, the second iteration yielded the best result: got rid of much of the background, without sacrificing too much of the foreground. However, the RGB segmentation was better.

Baby image

RGB segmentation of the baby image got rid of only the extreme white portion of the background during the first iteration, but got rid of significant parts of the foreground as well, during the second iteration. Otsu's algorithm was not able to find a k at the sweet spot in between. So I did a 'binary search' for that optimal k, where I found the k in the middle of the two k's returned by the first and second iterations. If that k was too high, next I considered the middle k between that and the result of first iteration. If that k was too low, I considered the k in the middle of that and the result of the second iteration. I continued this process till I found a k that separated the background and the foreground satisfactorily. As shown in the figure below, a k of 235 found at the third iteration of this method was found to be optimal. However, this segmentation is not very good either, since certain parts of the baby (right side of head) are lighter than the background. Parts of the blanket are also included in the foreground here since the blanket is as dark as parts of the baby are. But it is better than the extreme k levels that Otsu's algorithm was finding.


100 (k from 1st iteration)
177 (middle of 100 and 254) Iteration 1
216 (middle of 177 and 254) Iteration 2
235 (middle of 216 and 254) Iteration 3 optimal
245 (middle of 216 and 254) Iteration 4 Deviates from optimal
254 (k from 2nd iteration)

Performing texture-based segmentation on the baby image just got rid of the boundary of the baby and parts of the blanket, since that is where the texture is. More iterations just get rid of more textured parts, essentially more of the boundary. Using the texture based method, there is no way to get rid of just the background while keeping the entire foreground, since both the background and foreground have a smooth texture. What is different about the background is that it is white. So the only way I found to get rid of the background after the texture-based segmentation was to simply remove pixels that are white enough: pixels whose red, green and blue values are all above 210. Note that applying this white-removing filter on the segmented image gives a better result than applying it on the original image, because the segmentation gets rid of a part of the blanket, which the white-removing filter cannot.

Conclusion

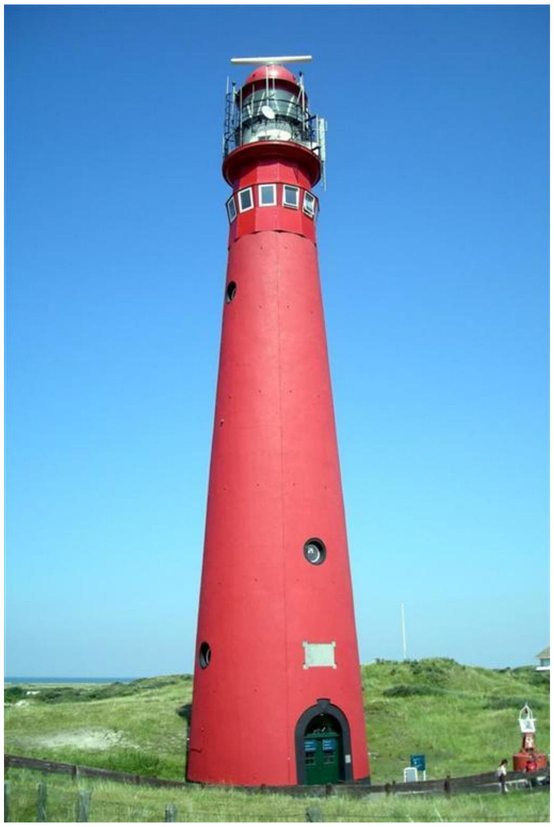
The catch is to find the iteration which gets rid of as much of the background as possible, while keeping as much of the foreground as possible. This works well when the background and foreground are well-separated by the property that is being used to segment. For example, the lighthouse was easy to segment using color because it is well-separated by color, but not easy to separate by texture since both foreground and background have similar textures. Some images are possible to segment using both color and texture, such as the ski image, though color segmentation works better than texture. Some images are simply not easy to segment using either color or texture, such as the baby image. Color segmentation again works better than texture in this case since the background is much lighter than the foreground.

A disadvantage of Otsu's algorithm is that there are no parameters that one can tweak to adjust the result. The algorithm uses its own judgement to find what it thinks is the optimal gray level. For example, in the baby picture, the first iteration by color only got rid of the extreme white, while the second iteration excluded colors that were part of the foreground. There is no way to adjust the algorithm to move the k-value to a slightly higher value for the first iteration, which may have served our purpose. Of course, one can do a 'binary search' for the optimal k-value between what the first and the second iterations yield, but that is extra work.

Results

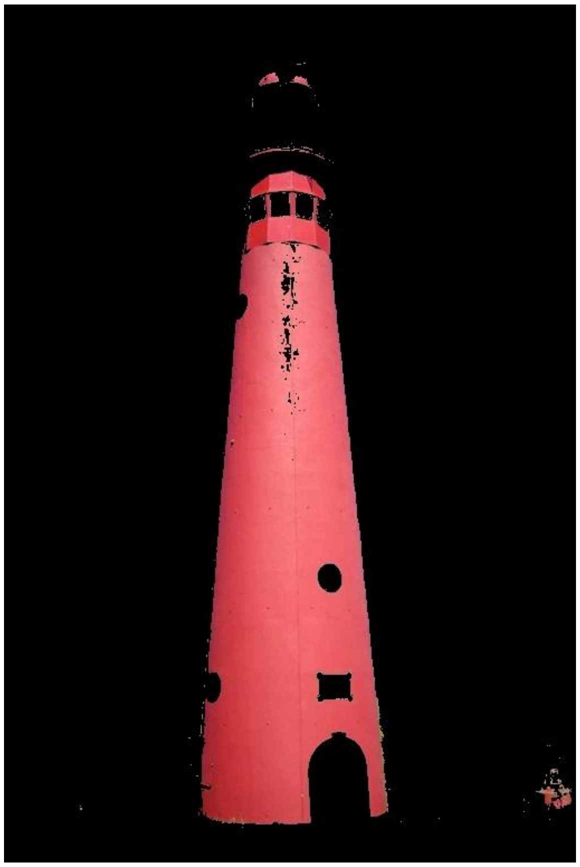
Lighthouse image

Rehana Mahfuz (rmahfuz@purdue.edu)
ECE 661 HW06



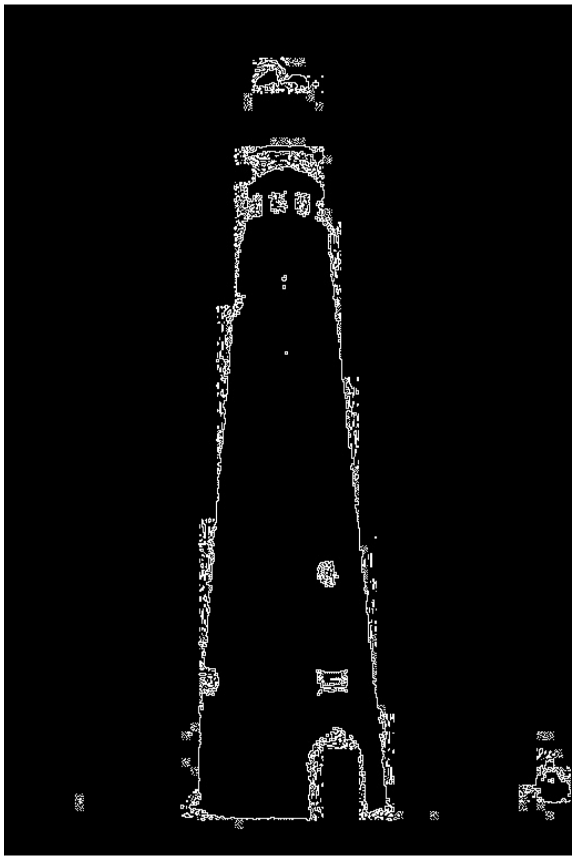
Original image

Rehana Mahfuz (rmahfuz@purdue.edu)
ECE 661 HW06



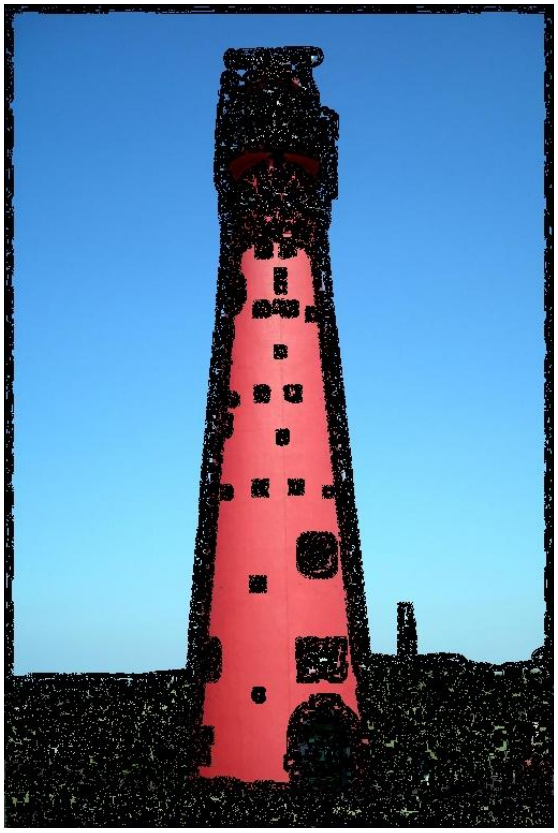
Color-segmented image. 1 iteration used

Rehana Mahfuz (rmahfuz@purdue.edu)
ECE 661 HW06



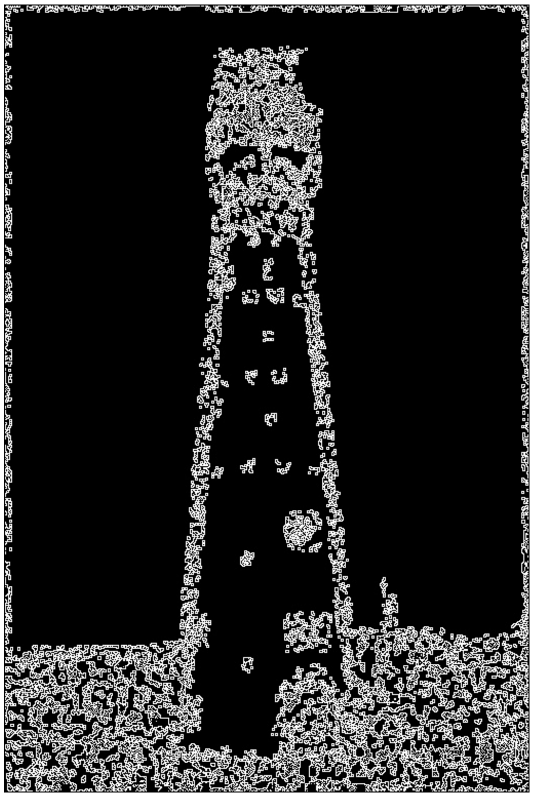
Contour from color-segmented image

Rehana Mahfuz (rmahfuz@purdue.edu)
ECE 661 HW06

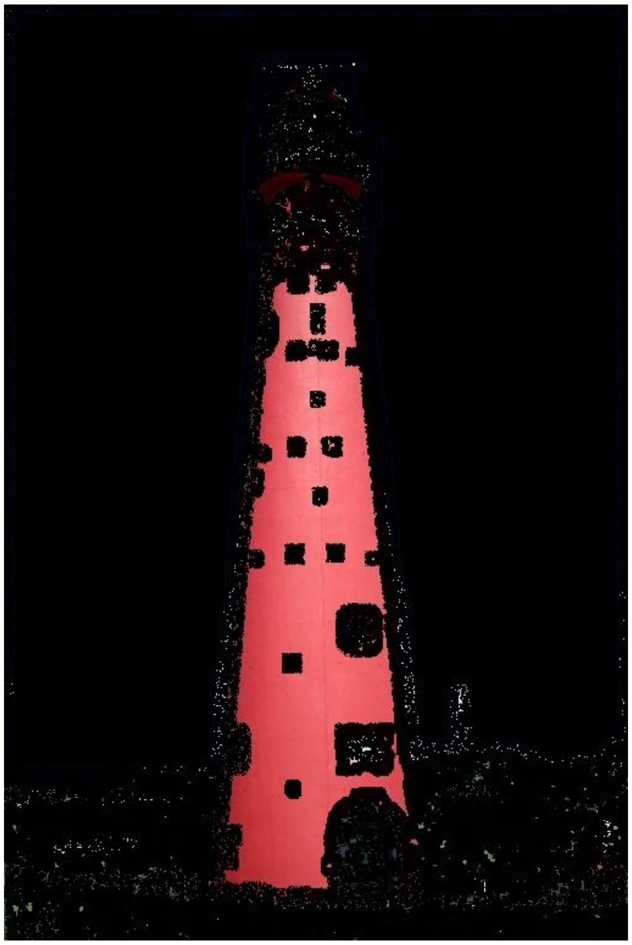


Texture-segmented image. Three iterations used. Earlier iterations did not exclude the grass enough, and more than three iterations exclude more grass but also start excluding more of the lighthouse.

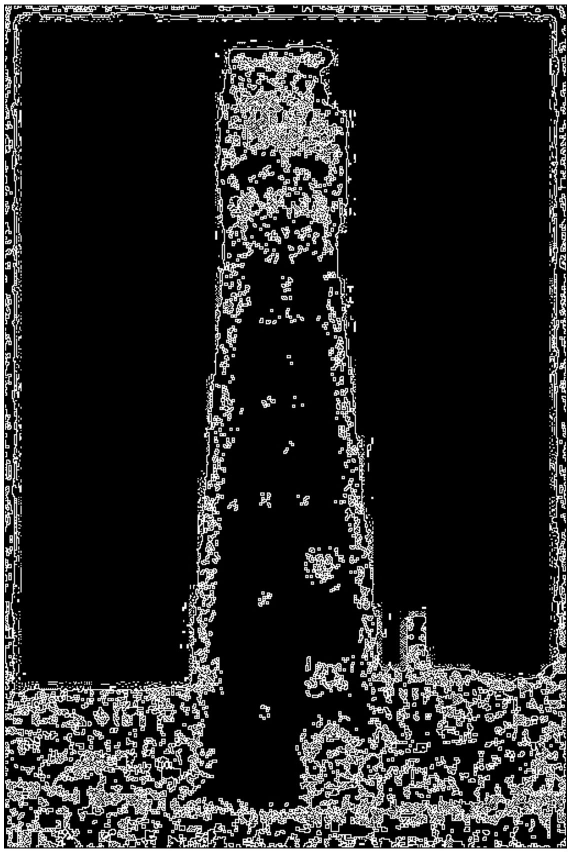
Rehana Mahfuz (rmahfuz@purdue.edu)
ECE 661 HW06


Contour from texture-segmented image

Rehana Mahfuz (rmahfuz@purdue.edu)
ECE 661 HW06



Blue sky removed from third iteration of texture-segmenting the image

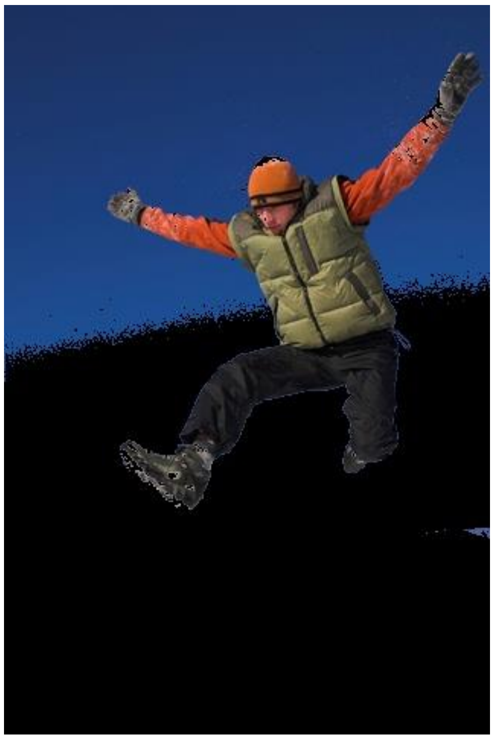


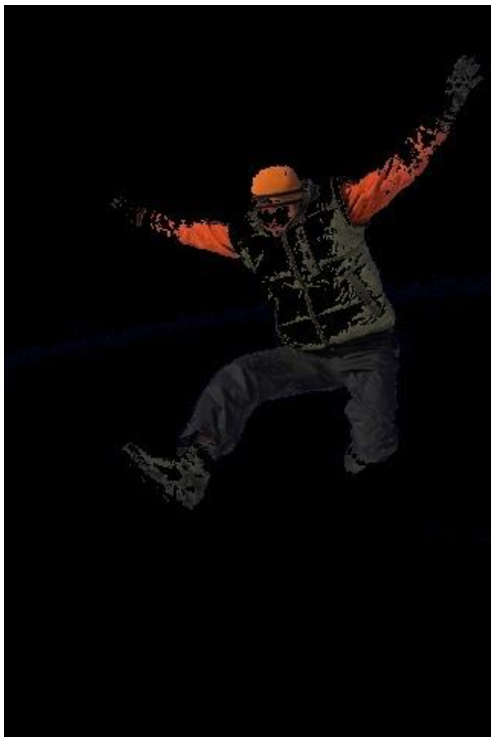
Contour from texture-segmented image with the blue sky removed

Ski image

Original image

Rehana Mahfuz (rmahfuz@purdue.edu)
ECE 661 HW06

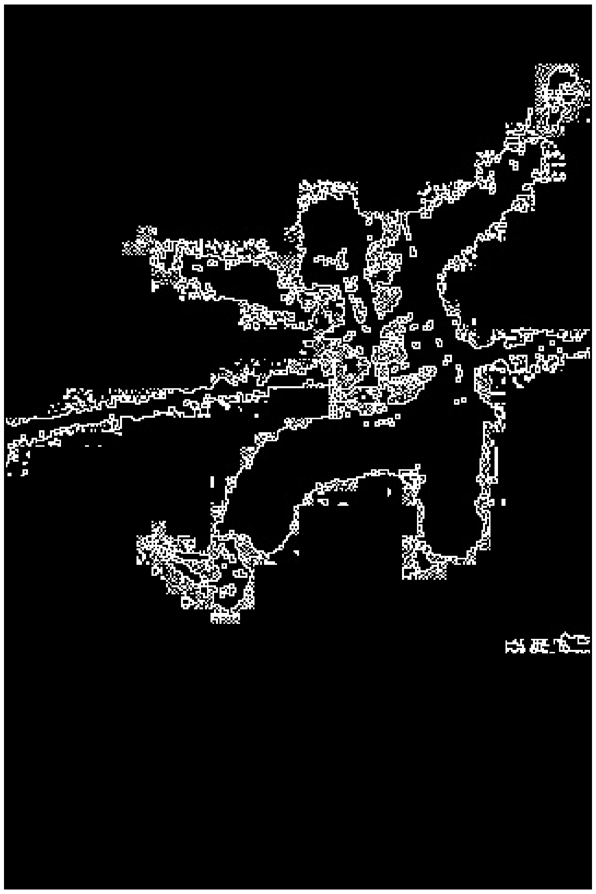




Left: Color-segmented image using only blue. Using other color channels was excluding parts of human from foreground. 1 iteration used. The background is still visible

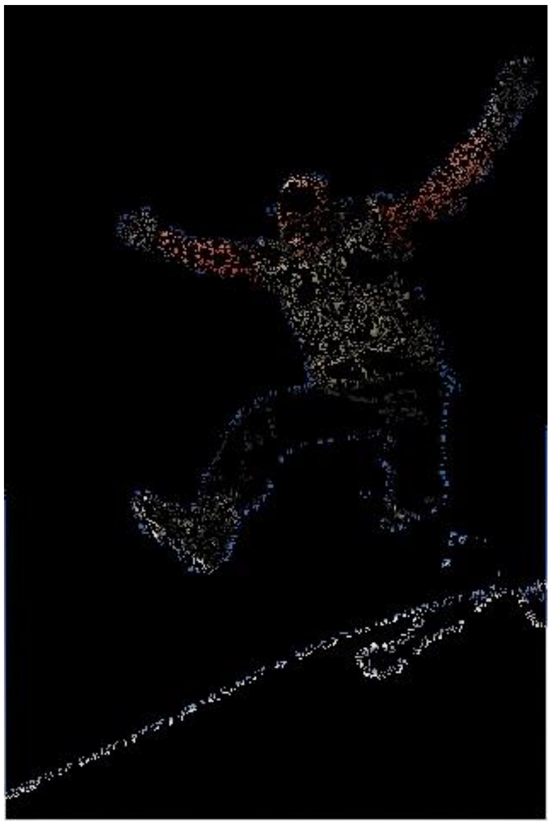
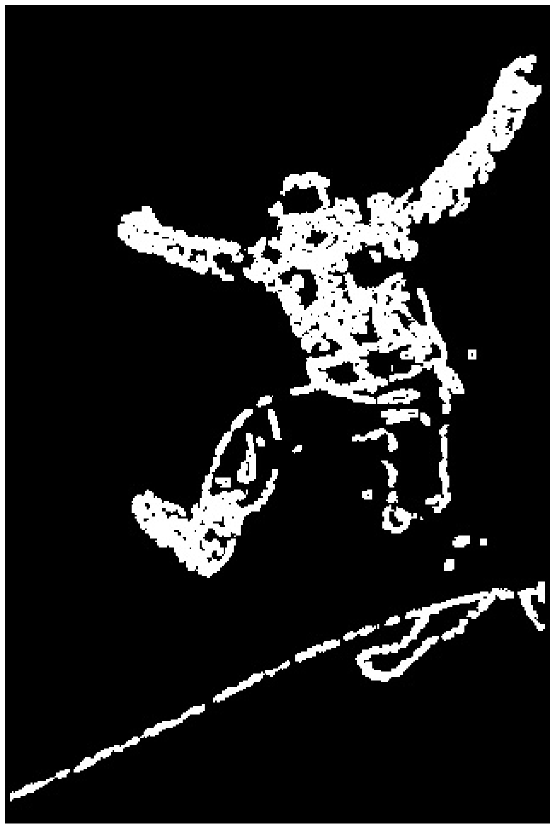
Right: Color-segmented image with two iterations using only blue. Background is excluded, but so is part of the human (foreground). A solution to this may be to use ‘binary search’ to find the optimal gray level, as done with the baby image

Rehana Mahfuz (rmahfuz@purdue.edu)
ECE 661 HW06



Contour of color-segmented ski image with 2 iterations

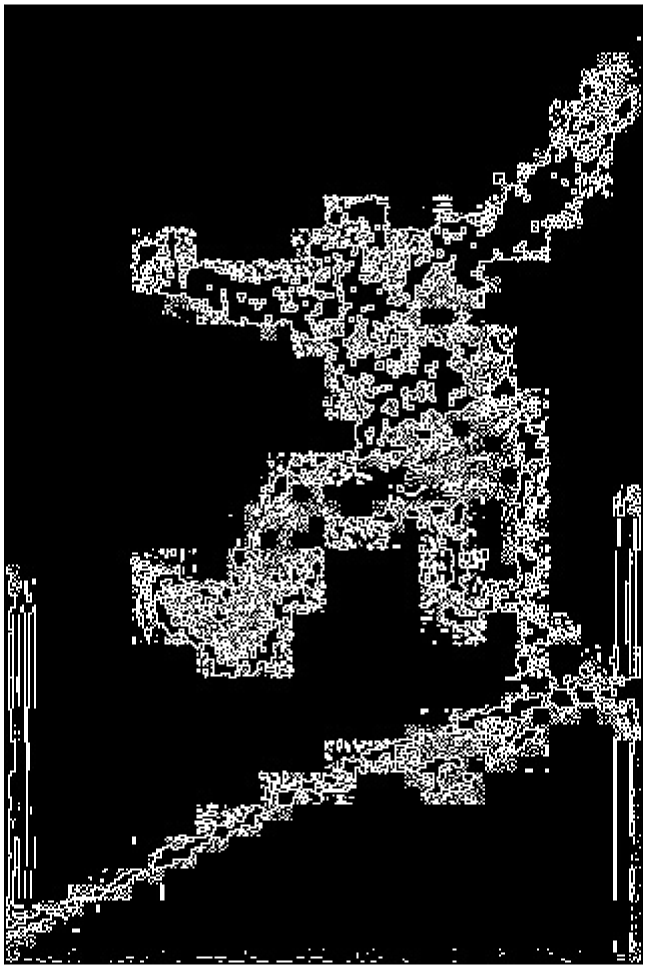
Rehana Mahfuz (rmahfuz@purdue.edu)
ECE 661 HW06

Left: Texture-segmented ski image using 2 iterations

Right: Mask of texture-segmented image (the actual image may be hard to see because of the dark color of the foreground)

Rehana Mahfuz (rmahfuz@purdue.edu)

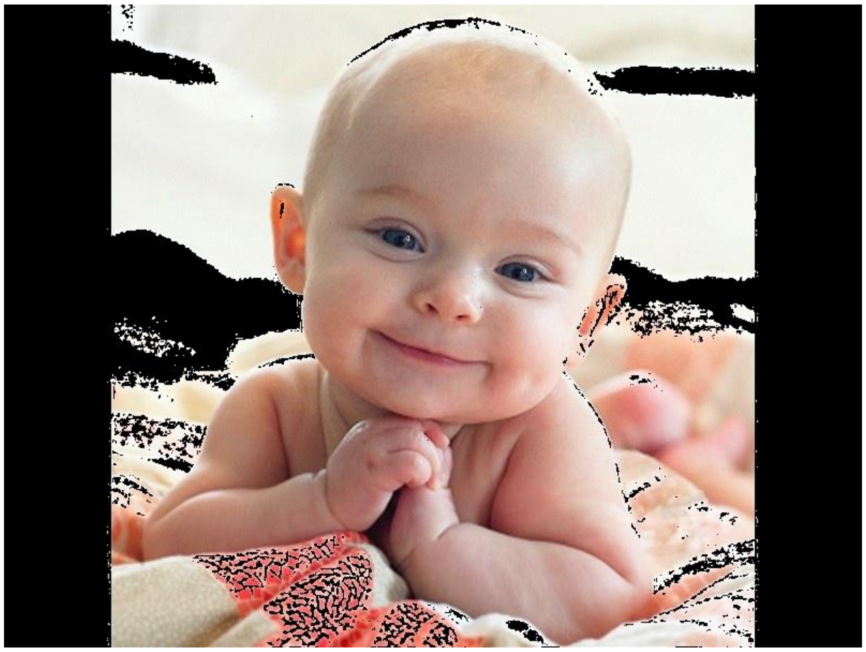


Contour of texture-segmented ski image

<u>Baby image</u>

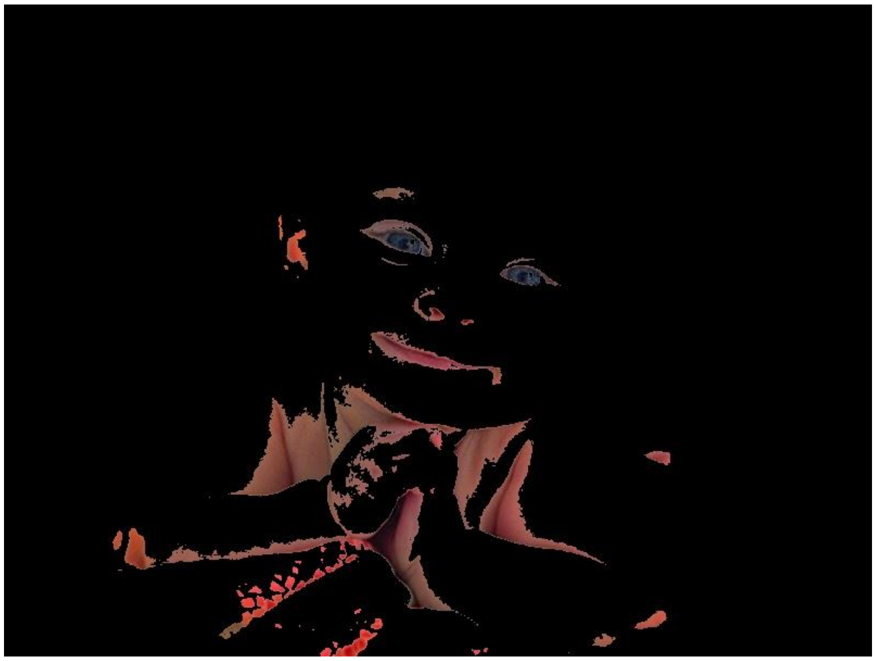
Rehana Mahfuz (rmahfuz@purdue.edu)
ECE 661 HW06

Original image

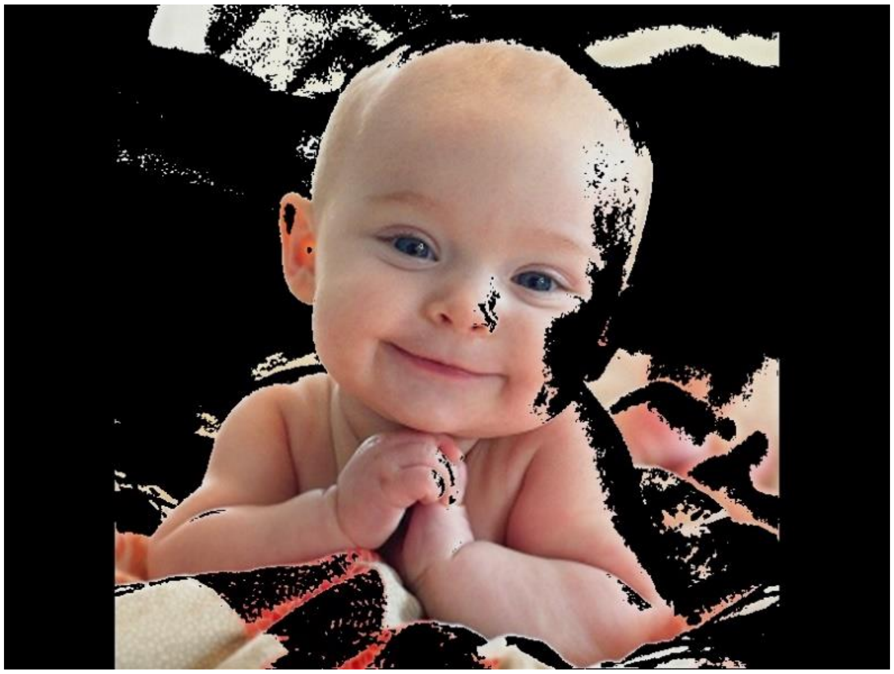


Color-segmented baby image using one iteration. This gets rid of a small part of the background, which is not enough, which is why a second iteration is attempted

Rehana Mahfuz (rmahfuz@purdue.edu)
ECE 661 HW06

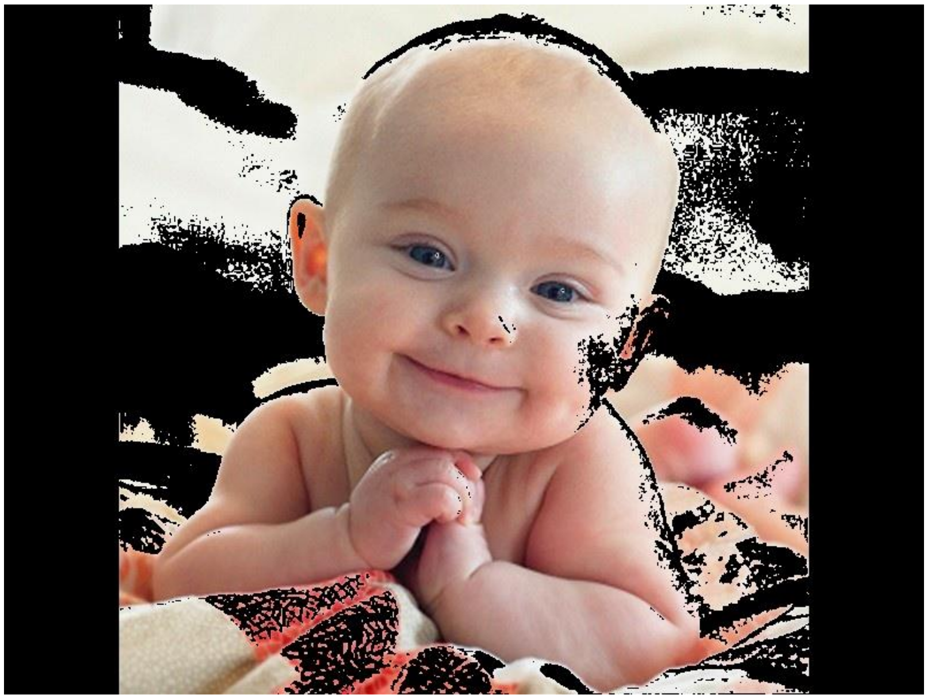


Color-segmented baby image using two iterations gets rid of not only the background, but also a good portion of the foreground. To fix this, I do a ‘binary search’ between the gray level values found using first and second iterations (100 and 254 respectively), till I find the value 235 (at my third attempt) which is a good segregation between the background and foreground. Another iteration of the ‘binary search’ gives me a k-value of 245, which includes too much of the background.

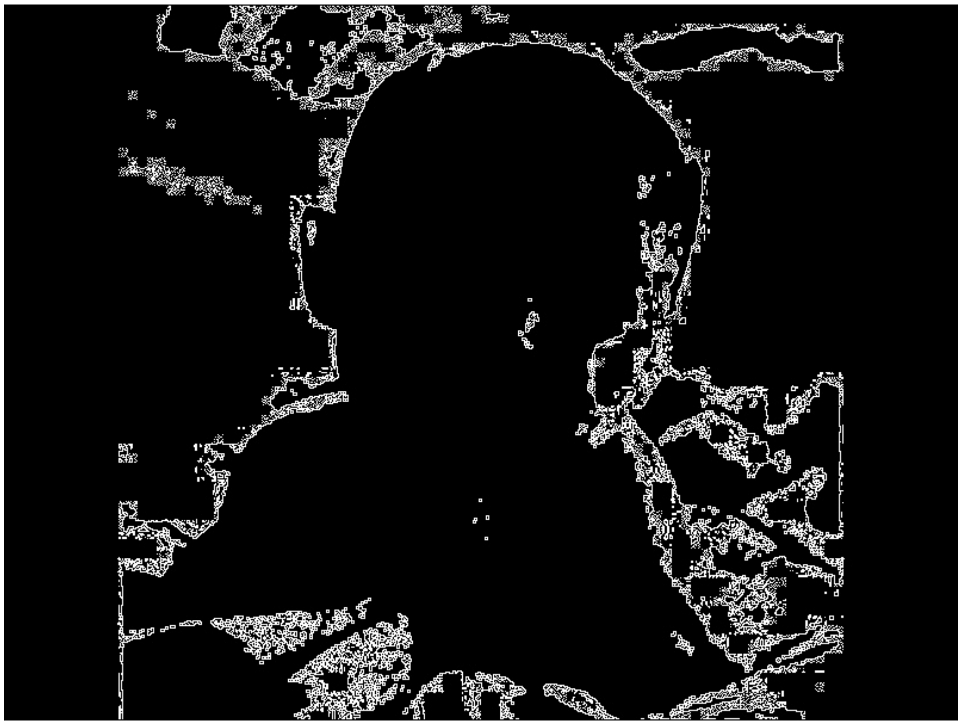


Color-segmented baby image using k = 235, as found using ‘binary search’, which is the optimum value

Rehana Mahfuz (rmahfuz@purdue.edu)
ECE 661 HW06

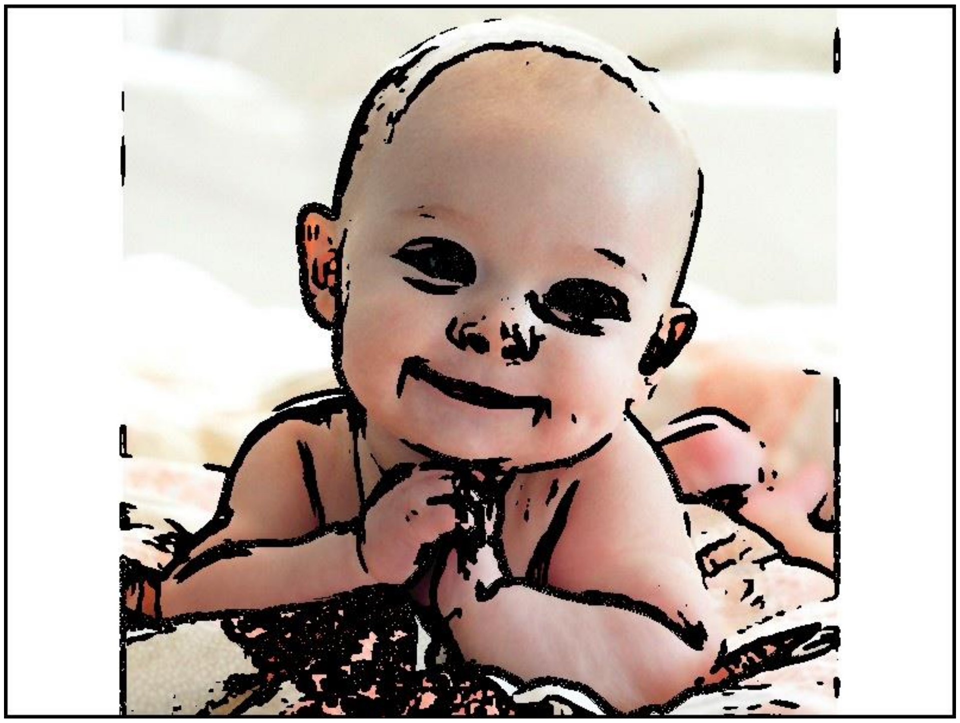


Color-segmented baby image using k = 245, as found using ‘binary search’. This k-value is too large as it includes too much of the background as well

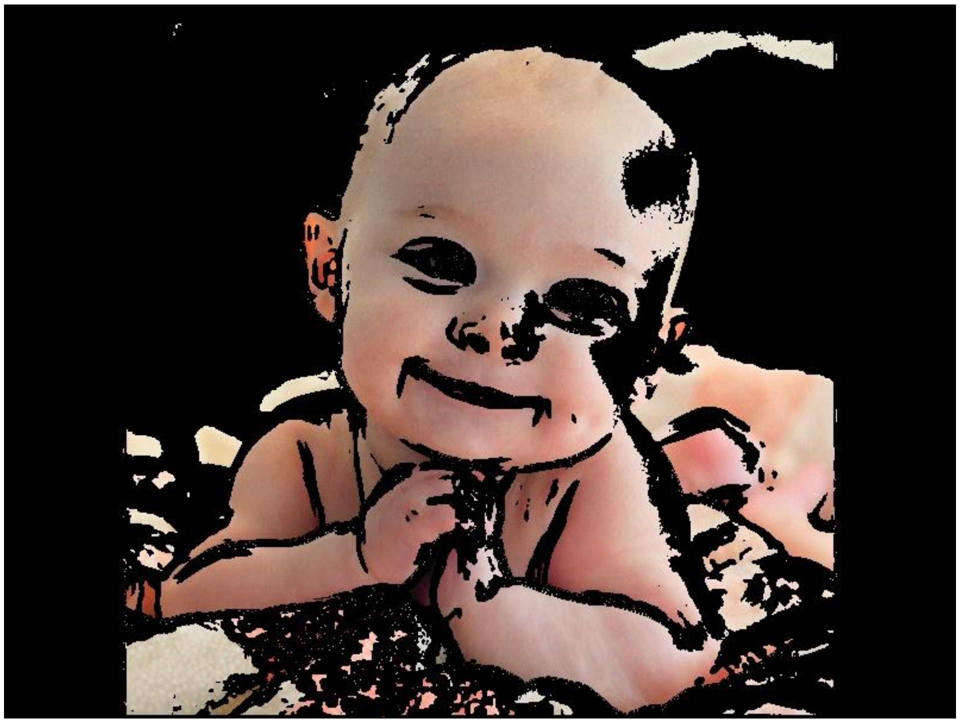


Contour from output of segmentation with k = 235

Rehana Mahfuz (rmahfuz@purdue.edu)
ECE 661 HW06

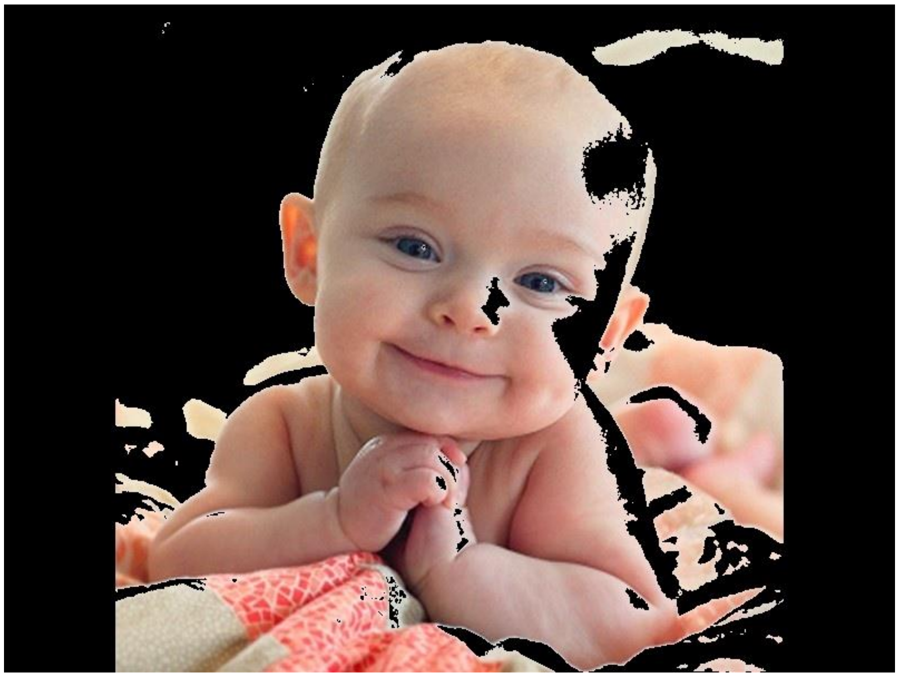


Texture-segmented image using one iteration. It does a good job of removing a little bit of the blanket, but does not remove the rest of the background. This is not surprising because just like the foreground, the background is also quite smooth in texture. A solution is to explicitly remove the white from the image

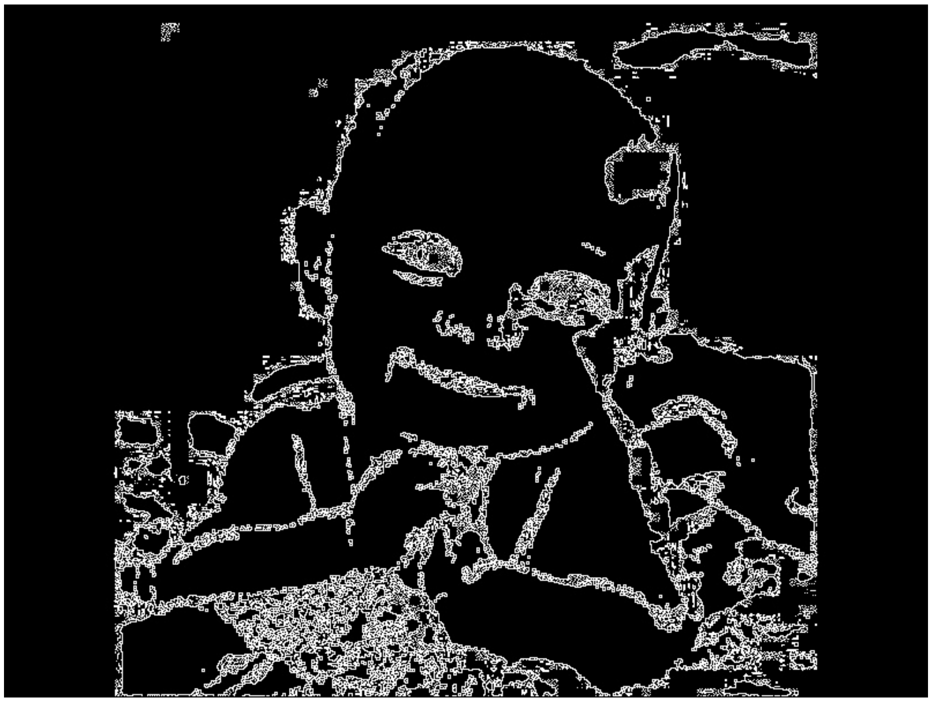


Texture-segmented image after removing the white (background)

Rehana Mahfuz (rmahfuz@purdue.edu)
ECE 661 HW06



Removing white background from original image (shown for the purpose of illustration). It is beneficial to apply this white-removing filter to the texture-segmented image than to the original image, since the texture-segmented image removes a part of the blanket.



Contour from output of texture-based segmentation, after removing the white

Rehana Mahfuz (rmahfuz@purdue.edu)
ECE 661 HW06

Source code

```
import numpy as np

import cv2, sys

import matplotlib.pyplot as plt


path = '../Users/rmahfuz/Desktop/661/HW06/'

#==========================================================================
====================================

def remove_sky_from_lighthouse(img):

    '''special function to remove sky from lighthouse after texture-based segmentation'''

    if img is None:

        img = cv2.imread(path + 'pictures/lighthouse/txt_segmented_color_iter2.jpg')

    for i in range(img.shape[0]):

        for j in range(img.shape[1]):

            if img[i][j][0] > 100 and img[i][j][2] < 190:

                img[i][j] = [0,0,0]

    cv2.imwrite(path + 'pictures/lighthouse/removed_sky.jpg', img)

    return img

#==========================================================================
====================================

def process_baby(img = None):

    '''special function to remove the white background after texture-based segmentation'''

    if img is None:

        img = cv2.imread(path + 'pictures/baby.jpg')

    for i in range(img.shape[0]):

        for j in range(img.shape[1]):

            if img[i][j][0] > 210 and img[i][j][2] > 210 and img[i][j][2] > 210:

                img[i][j] = [0,0,0]

    cv2.imwrite(path + 'pictures/baby/refined.jpg', img)

    return img
```

```
#====================================================================================
====================================

def otsu(img):
    '''accepts a grayscale image. returns k'''
    assert len(img.shape) == 2
    L = np.max(img)
    N = img.shape[0]*img.shape[1]
    n = np.array(list(map(lambda i: len(img[img==i]), range(L))))
    p = n/N
    def omega(k):
        return np.sum(p[:int(k)])
    def mu(k):
        result = 0
        for i in range(int(k)):
            result += (i* p[i])
        return result
    mu_T = mu(L)
    def mu0(k):
        return mu(k)/omega(k)
    def mu1(k):
        return (mu_T - mu(k))/(1 - omega(k))
    def sigma_b_sq(k):
        return omega(k) * (1 - omega(k)) * np.square(mu1(k) - mu0(k))
    #Find all k's for which 0 < omega(k) < 1
    #k_range = list(range(L))[1:]
    k_range = np.array([])
    for k in range(L):
        if omega(k) > 0 and omega(k) < 1:
            k_range = np.append(k_range, k)
    #Within that range of k, find the k for which sigma_b_sq is maximum
    k_star = k_range[0]
    max_sigma = sigma_b_sq(k_star)
    for k in k_range:
        if sigma_b_sq(k) > max_sigma:
            max_sigma = sigma_b_sq(k)
            k_star = k
    #print('k_star = ', k_star)
    return k_star
#=====================================================================================================
def segment(name, iteration = 0, img = None):
    '''segments by color. accepts name: a string. optionally accepts an image: img'''
    if img is None:
        img = cv2.imread(path + 'pictures/' + name +'.jpg')
    def segment_given_k(actual_image, k, order = 0):
        image = actual_image
        for i in range(image.shape[0]):
            for j in range(image.shape[1]):
                if order == 0:
                    cmp = image[i,j] >= k
                else:
                    cmp = image[i,j] < k
                if cmp == True:
                    image[i,j] = 0
        return image
    k_blue = otsu(img[:,:,0]) #blue
    k_green = otsu(img[:,:,1]) #green
    k_red = otsu(img[:,:,2]) #red
    print('k_blue = {}, k_green = {}, k_red = {}'.format(k_blue, k_green, k_red))
    segmented_blue = segment_given_k(img[:,:,0].copy(), k_blue)
    segmented_green = segment_given_k(img[:,:,1].copy(), k_green)
    segmented_red = segment_given_k(img[:,:,2].copy(), k_red)
    new_img = img.copy()
    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            if not(segmented_blue[i][j] != 0 and segmented_green[i][j] != 0 and segmented_red[i][j] != 0):
            #if segmented_blue[i][j] == 0 and segmented_green[i][j] == 0 and segmented_red[i][j] == 0:
                new_img[i][j] = 0

    cv2.imwrite(path + 'pictures/' + name + '/segmented_iter' + str(iteration) + '.jpg', new_img)
    #cv2.imwrite(path + 'pictures/' + name + '/exp2.jpg', new_img)

    return new_img
#====================================================================================================================
def segment_by_texture(name, iteration = 0, img = None):
    '''accepts name: a string. optionally accepts an image: img'''
    if img is None:
        img = cv2.imread(path + 'pictures/' + name +'.jpg')
    if len(img.shape) == 3: #convert to grayscale if img is bgr
        img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    def get_channel(N): #returns img. variance from NxN channel
        image = img.copy()
        win = int(N/2)
        for i in range(win, image.shape[0] - win):
            for j in range(win, image.shape[1] - win):
                patch = img[i-win:i+win+1, j-win:j+win+1]
                image[i,j] = np.var(patch)
        image = (image / np.max(image)) * 255
        return image.astype(int)
```

```
def segment_given_k(actual_image, k, order = 0):
    image = np.zeros((actual_image.shape[0], actual_image.shape[1]))
    for i in range(image.shape[0]):
        for j in range(image.shape[1]):
            if order == 0:
                cmp = actual_image[i,j] >= k
            else:
                cmp = actual_image[i,j] < k
            if cmp == False:
                image[i,j] = 255
    return image
one = get_channel(3); two = get_channel(5); three = get_channel(7)
k_one = otsu(one)
k_two = otsu(two)
k_three = otsu(three)
print('k_one = {}, k_two = {}, k_three = {}'.format(k_one, k_two, k_three))
segmented_one = segment_given_k(one, k_one)
segmented_two = segment_given_k(two, k_two)
segmented_three = segment_given_k(three, k_three)
new_img = cv2.imread(path + 'pictures/' + name +'.jpg')
#new_img = np.ones((img.shape[0], img.shape[1])); new_img = new_img * 255
for i in range(img.shape[0]):
    for j in range(img.shape[1]):
        if not(segmented_one[i][j] != 0 and segmented_two[i][j] != 0 and segmented_three[i][j] != 0):
        #if segmented_blue[i][j] == 0 and segmented_green[i][j] == 0 and segmented_red[i][j] == 0:
            new_img[i][j] = [0,0,0]

cv2.imwrite(path + 'pictures/' + name + '/txt_segmented_color_iter' + str(iteration) + '.jpg', new_img)
return new_img
```

```
#===========================================================================
===================================

def contour(name, img):
    if len(img.shape) == 3:
        img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    mask = np.zeros((img.shape[0], img.shape[1]))
    for i in range(1, img.shape[0] - 1):
        for j in range(1, img.shape[1] - 1):
            if img[i][j] != 0:
                #if any of the 8 neighbors is 0:
                if img[i-1,j-1] == 0 or img[i-1,j] == 0 or img[i-1,j+1] == 0 or img[i,j+1] == 0\
                or img[i+1,j+1] == 0 or img[i+1,j] == 0 or img[i+1,j-1] == 0 or img[i,j-1] == 0:
                    mask[i][j] = 255
    cv2.imwrite(path + 'pictures/' + name, mask)
    return mask
#===========================================================================
===================================

def apply_k(name, k, img):
    def segment_given_k(actual_image, k, order = 0):
        image = actual_image
        for i in range(image.shape[0]):
            for j in range(image.shape[1]):
                if order == 0:
                    cmp = image[i,j] >= k
                else:
                    cmp = image[i,j] < k
                if cmp == True:
                    image[i,j] = 0
        return image
    segmented_blue = segment_given_k(img[:,:,0].copy(), k)
    segmented_green = segment_given_k(img[:,:,1].copy(), k)
    segmented_red = segment_given_k(img[:,:,2].copy(), k)
    new_img = img.copy()
    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            if not(segmented_blue[i][j] != 0 and segmented_green[i][j] != 0 and segmented_red[i][j] != 0):
                new_img[i][j] = 0

    cv2.imwrite(path + name, new_img)
    return new_img
#===================================================================================================================
def main():
    #-----Lighthouse-------------------------------------------------------------------------------------------------

    img = segment('lighthouse')
    contour('lighthouse/contour_rgb.jpg', img)
    img = segment_texture('lighthouse')
    for i in range(3):
       img = segment_texture('lighthouse', i+1, img)
    contour('lighthouse/contour_txt.jpg', img)
    remove_sky_from_lighthouse(img)
    contour('lighthouse/contour_txt_removed_sky.jpg', img)

    #-----Ski--------------------------------------------------------------------------------------------------------
    img = segment('ski')
    img = segment('ski', 1, img)
    contour('ski/contour_rgb_2iter.jpg', img)

    contour('lighthouse/contour_rgb.jpg', img)
    img = segment_texture('ski')
    img = segment_texture('ski', 1, img)
    contour('ski/contour_txt_1iter.jpg', img)


    #-----Baby----------------------------------------------------------------------------------------------------------
    img = segment('baby')
    contour('baby/contour_segmented_color_iter0.jpg', img)
    img = segment('baby', 1, img)
    contour('baby/contour_segmented_color_iter1.jpg', img)


    #binary search
    apply_k('pictures/baby/binary_search177.jpg', 177, cv2.imread(path + 'pictures/baby.jpg'))
    apply_k('pictures/baby/binary_search216.jpg', 216, cv2.imread(path + 'pictures/baby.jpg'))
    img = apply_k('pictures/baby/binary_search235.jpg', 235, cv2.imread(path + 'pictures/baby.jpg'))
    contour('baby/contour_binary_search235.jpg', img)
    apply_k('pictures/baby/binary_search245.jpg', 245, cv2.imread(path + 'pictures/baby.jpg'))


    img = segment_texture('baby') #removed some of blanket
    contour('baby/contour_segmented_txt.jpg', img)
    img = process_baby(img)
    contour('baby/contour_segmented_txt_refined.jpg', img)
#===============================================================================
===================================
if __name__ == '__main__':
    main()
```