ECE 661 Fall 2018
Homework #6

Ye Shi
shi349@purdue.edu

October 23, 2018

Contents

1 Methodology **2**
1.1 Otsu Algorithm . . . 2
1.2 Contour Extraction . . . 2
1.3 Texture Based Segmentation (TBS) . . . 3

2 Obesrvations **3**

3 Outputs and Results **3**
3.1 lighthouse.jpg . . . 4
3.1.1 Otsu Segmentation . . . 4
3.1.2 Texture Based Segmentation . . . 9
3.2 baby.jpg . . . 14
3.2.1 Otsu Segmentation . . . 14
3.2.2 Texture Based Segmentation . . . 16
3.3 ski.jpg . . . 19
3.3.1 Otsu Segmentation . . . 19
3.3.2 Texture Based Segmentation . . . 24

4 Source Code **29**
4.1 Main Script . . . 29
4.2 Segmentation Call Function . . . 30
4.2.1 Otsu Algorithm . . . 30
4.2.2 Texture Based Algorithm . . . 31
4.2.3 Segmentation Plot Function . . . 32
4.2.4 Contour Extraction . . . 32

1 Methodology

1.1 Otsu Algorithm

Otsu's method, named after Nobuyuki Otsu, is used to automatically perform clustering-based image thresholding, or, the reduction of a graylevel image to a binary image. Assume the imput image is in 8-bit grayscale, $[0, 255]^3$. The algorithm is presented as follows.

1. The probability mass function (PMF) by the image pixel intensity level as

$$p[i] = \frac{n[i]}{N},$$

where $i \in [0, 255]$, $n[i]$ is the total number with i intensity, and N is the total pixel number.

2. Then we set a threshold $k \in [0, 255]$ and find the cumulative distribution function (CDF) of $p[i]$ as

$$F[k] = \sum_{i=0}^{k} p[i].$$

By k, we divide the image into two classes, where

$$\omega_0 = F[k]$$
$$\omega_1 = 1 - F[k]'$$

ω_0 represents the probability that a pixel intensity is lower or equal than k; ω_1 represents the probability that a pixel intensity is greater than k.

3. The average pixel intensity is calculated as follows,

$$\mu_0 = \frac{\sum_{i=0}^{k} i}{\sum_{i=0}^{k} n[i]}$$
$$\mu_1 = \frac{\sum_{i=k+1}^{255} i}{\sum_{i=k+1}^{255} n[i]}.$$

4. Then the between class variance is given as

$$\sigma_B^2 = \omega_0 \omega_1 (\mu_0 - \mu_1)^2.$$

5. Change the threshold k and find the maxima k^* of σ_B^2. It can be done either by exhausting search all $k \in [0, 255]$ by repeating Step 2 to 4, or use a optimization algorithm.

6. The foreground is all pixels with intensity lower than k^*; the background is all pixels with intensity grater equal than k^*.

7. For better result, you can also repeat Step 1 to 6 in the result foreground image. As the homework instruction said, it may give superior results from the previous invocation.

Please note, sometimes the background and foreground may be exchanged due to the intensity of object.
In this homework implementation, we use a RGB-color image as 3 grayscale images. Then we combine the 3 images by AND logic to find the foreground.

1.2 Contour Extraction

In this homework, I applied a simple d_8 contour extraction for the generated binary mask. This method check the summation of a 3×3 window, if it is not fully occupied or empty, where the summation is not 9 or 0, the center of the window will be marked as a point on contour.

1.3 Texture Based Segmentation (TBS)

We generate the texture images based on overall variance of each square window centered at each pixel of a transformed grayscale image. In this homework, I used 3×3, 5×5, 7×7. We then applied Otsu algorithm on those generated variance images. Please note, in practical, I normalize the variance images to grayscale, [0,255], by uniform distribution before applying Otsu algorithm to save the computation time.

2 Obesrvations

1. In the step output in Section 2.1.1, we notice that due to the color channel density, the foreground and background can be switched as shown in Fig.4. Due to this, it is unable to present an acceptable result as an object than a contour.

2. From the results, it is obvious that the TBS works better in cases that the object has stable texture, than simply applied Otsu in RGB. For example, in ski.jpg, Fig.15a only keeps the person than Fig.12a still keeps the sky. In contrast, while the object has noncontinuous texture, as the baby face in Fig.10a, the result seems to be scaring due to the texture changes while there is a nose or eye. In this case, we may apply a noise filter like dilation and erosion. However, it demands test for finding the best window size.

3. It is also obvious that the TBS method takes more computation power than simple Otsu.

4. In Section 1, we mentioned by applying Otsu algorithm several iterations of the segmented object may lead better results. However, it does not grantee. For example, Fig.15 is the second iteration of ski.jpg by TBS, which eliminates many pixels.

In conclusion, the performance of the two methods depend on the target object we want to segment. If it has more texture information, then TBS is better; If it has even color distribution, than the RGB is better.

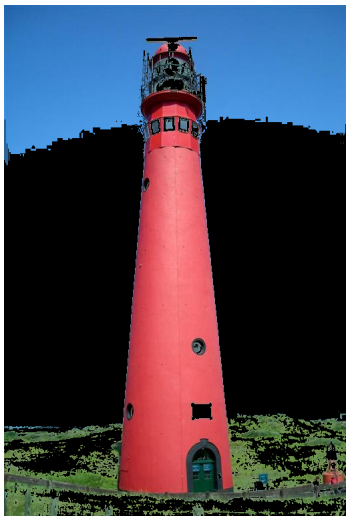
3 Outputs and Results

In this section, I display the output of the process and the 1st iteration result in foreground segmentation, background segmentation and contour in green lines.

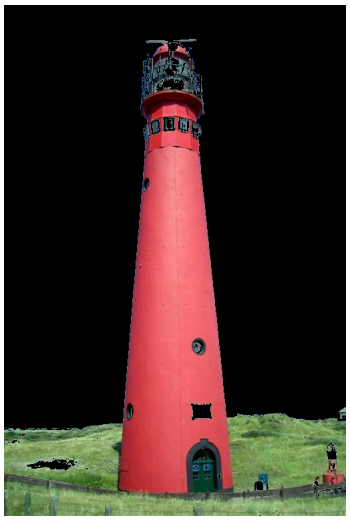
3.1 lighthouse.jpg

3.1.1 Otsu Segmentation

(a) R channel foreground segmentation



(b) G channel foreground segmentation



(c) B channel foreground segmentation

(d) R channel background segmentation

(e) G channel background segmentation

(f) B channel background segmentation

(g) R channel contour segmentation

(h) G channel contour segmentation

(i) B channel contour segmentation

Figure 1: Step Output of Otsu Algorithm

(a) Combined foreground segmentation

(b) Combined background segmentation

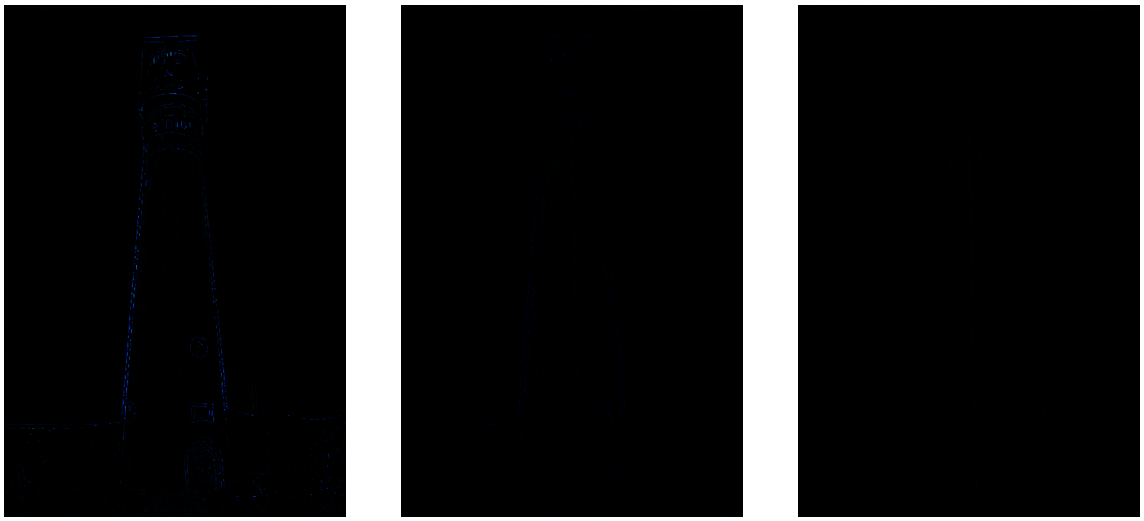
(c) Combined contour segmentation

Figure 2: 1st Iteration Result of Otsu Algorithm

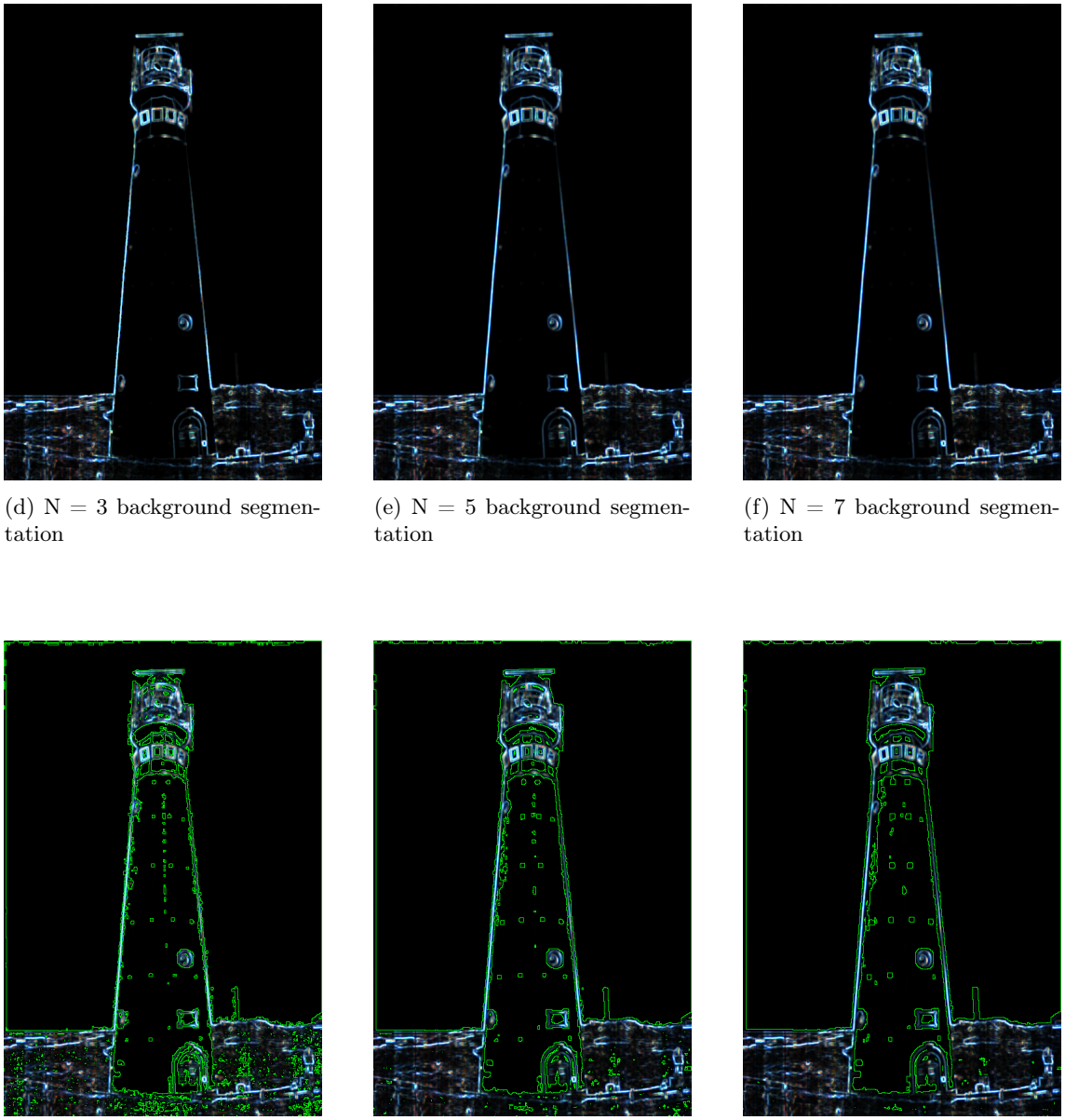
3.1.2 Texture Based Segmentation

(a) N = 3 Texture Image (b) N = 5 Texture Image (c) N = 7 Texture Image

Figure 3: Output of the Variance Image



(a) N = 3 foreground segmentation (b) N = 5 foreground segmentation (c) N = 7 foreground segmentation



(d) N = 3 background segmentation

(e) N = 5 background segmentation

(f) N = 7 background segmentation

(g) N = 3 contour segmentation

(h) N = 5 contour segmentation

(i) N = 7 contour segmentation

Figure 4: Step Output of TBS Algorithm

(a) Combined foreground segmentation

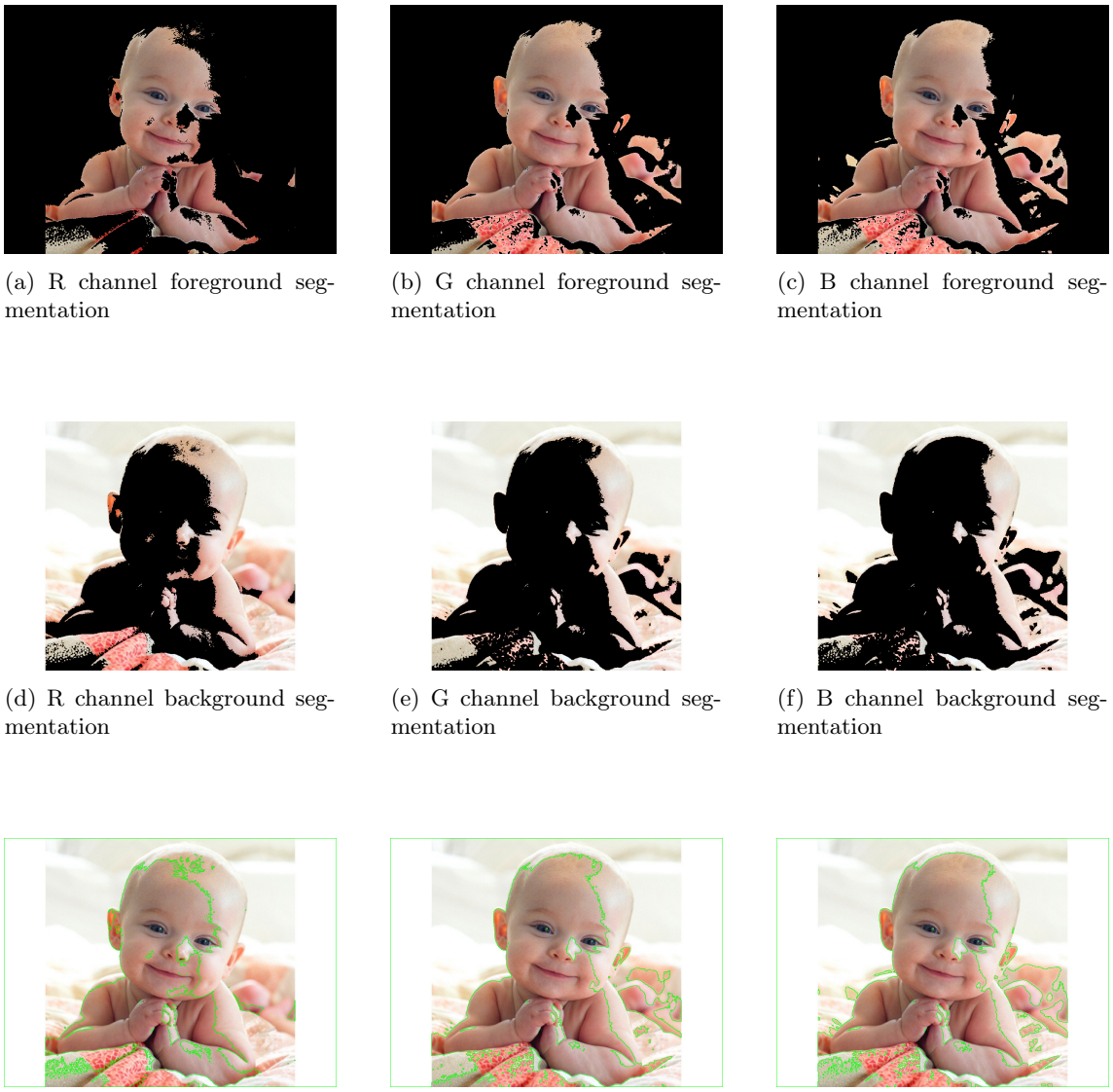
(b) Combined background segmentation

(c) Combined contour segmentation

Figure 5: 1st Iteration Result of TBS Algorithm

3.2 baby.jpg

3.2.1 Otsu Segmentation



(a) R channel foreground segmentation

(b) G channel foreground segmentation

(c) B channel foreground segmentation

(d) R channel background segmentation

(e) G channel background segmentation

(f) B channel background segmentation

(g) R channel contour segmentation

(h) G channel contour segmentation

(i) B channel contour segmentation

Figure 6: Step Output of Otsu Algorithm

(a) Combined foreground segmentation

(b) Combined background segmentation

(c) Combined contour segmentation

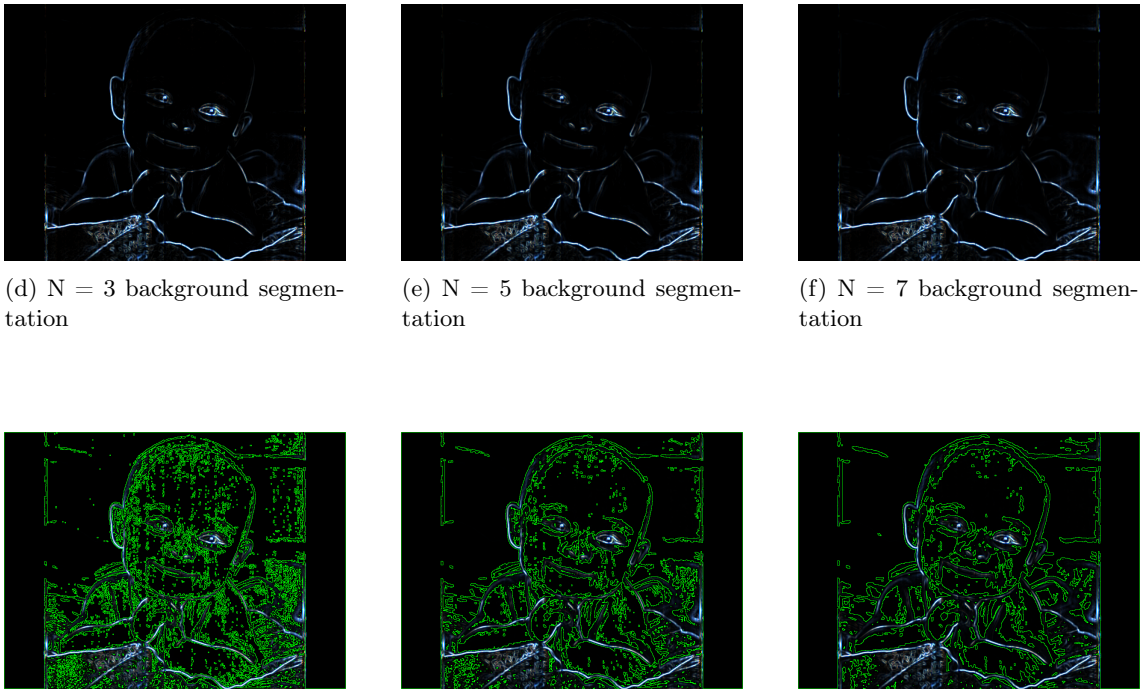
Figure 7: 1st Iteration Result of Otsu Algorithm

3.2.2 Texture Based Segmentation

(a) N = 3 Texture Image (b) N = 5 Texture Image (c) N = 7 Texture Image

Figure 8: Output of the Variance Image

(a) N = 3 foreground segmentation (b) N = 5 foreground segmentation (c) N = 7 foreground segmentation



(d) N = 3 background segmentation

(e) N = 5 background segmentation

(f) N = 7 background segmentation

(g) N = 3 contour segmentation

(h) N = 5 contour segmentation

(i) N = 7 contour segmentation

Figure 9: Step Output of TBS Algorithm

(a) Combined foreground segmentation

(b) Combined background segmentation

(c) Combined contour segmentation

Figure 10: 1st Iteration Result of TBS Algorithm

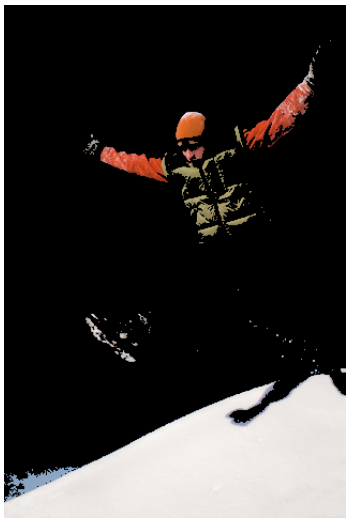
3.3 ski.jpg

3.3.1 Otsu Segmentation

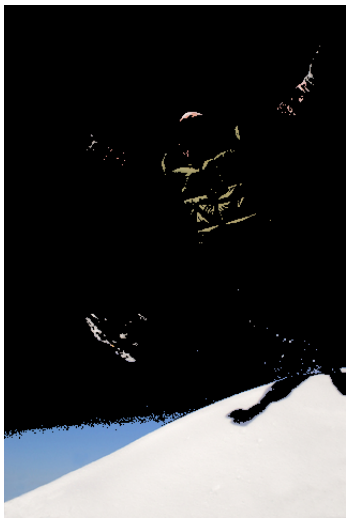
(a) R channel foreground segmentation

(b) G channel foreground segmentation

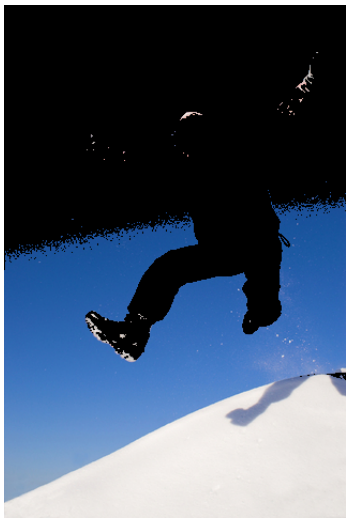
(c) B channel foreground segmentation



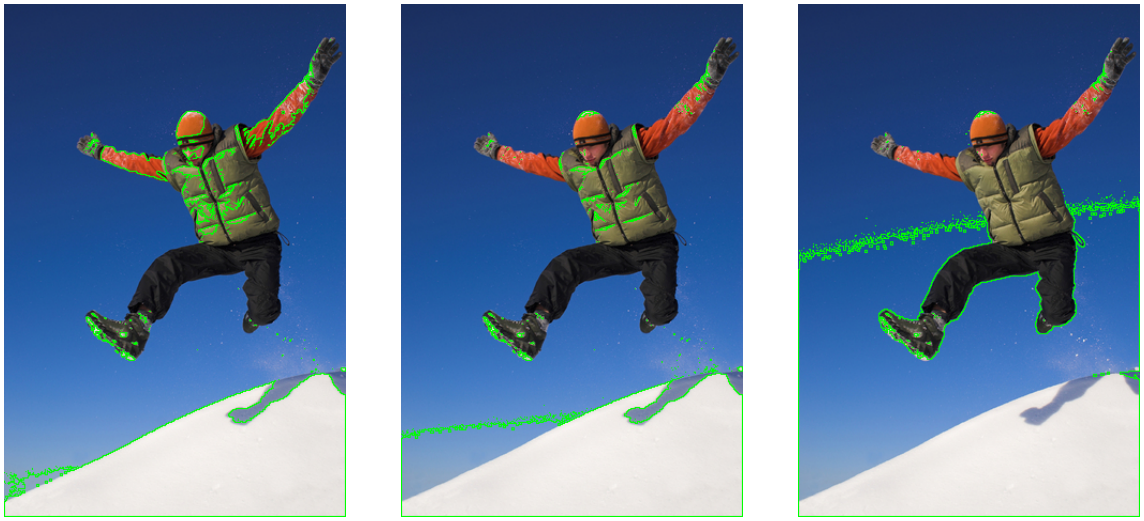
(d) R channel background segmentation



(e) G channel background segmentation



(f) B channel background segmentation



(g) R channel contour segmentation

(h) G channel contour segmentation

(i) B channel contour segmentation

Figure 11: Step Output of Otsu Algorithm

(a) Combined foreground segmentation

(b) Combined background segmentation

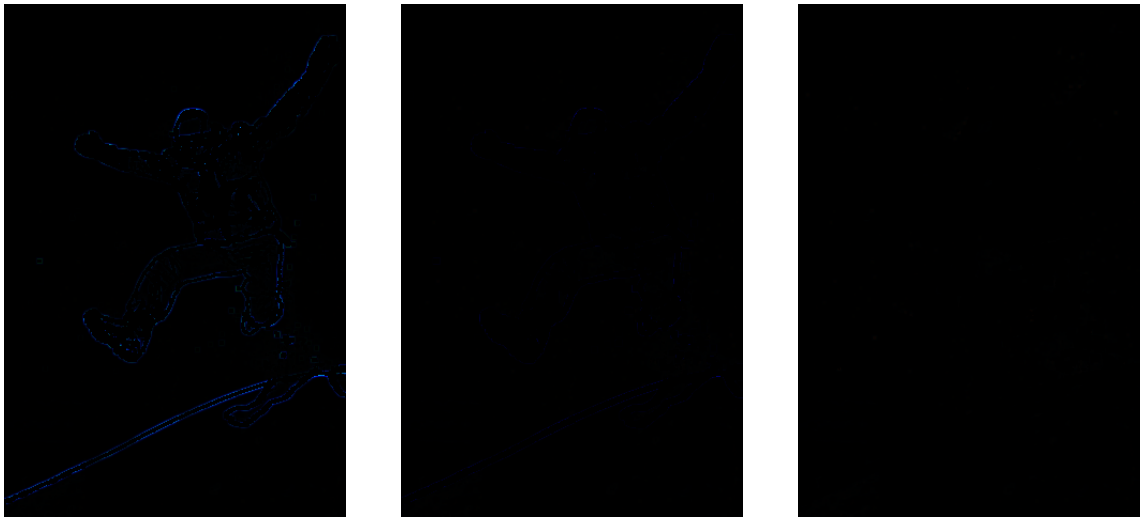
(c) Combined contour segmentation

Figure 12: 1st Iteration of Otsu Algorithm

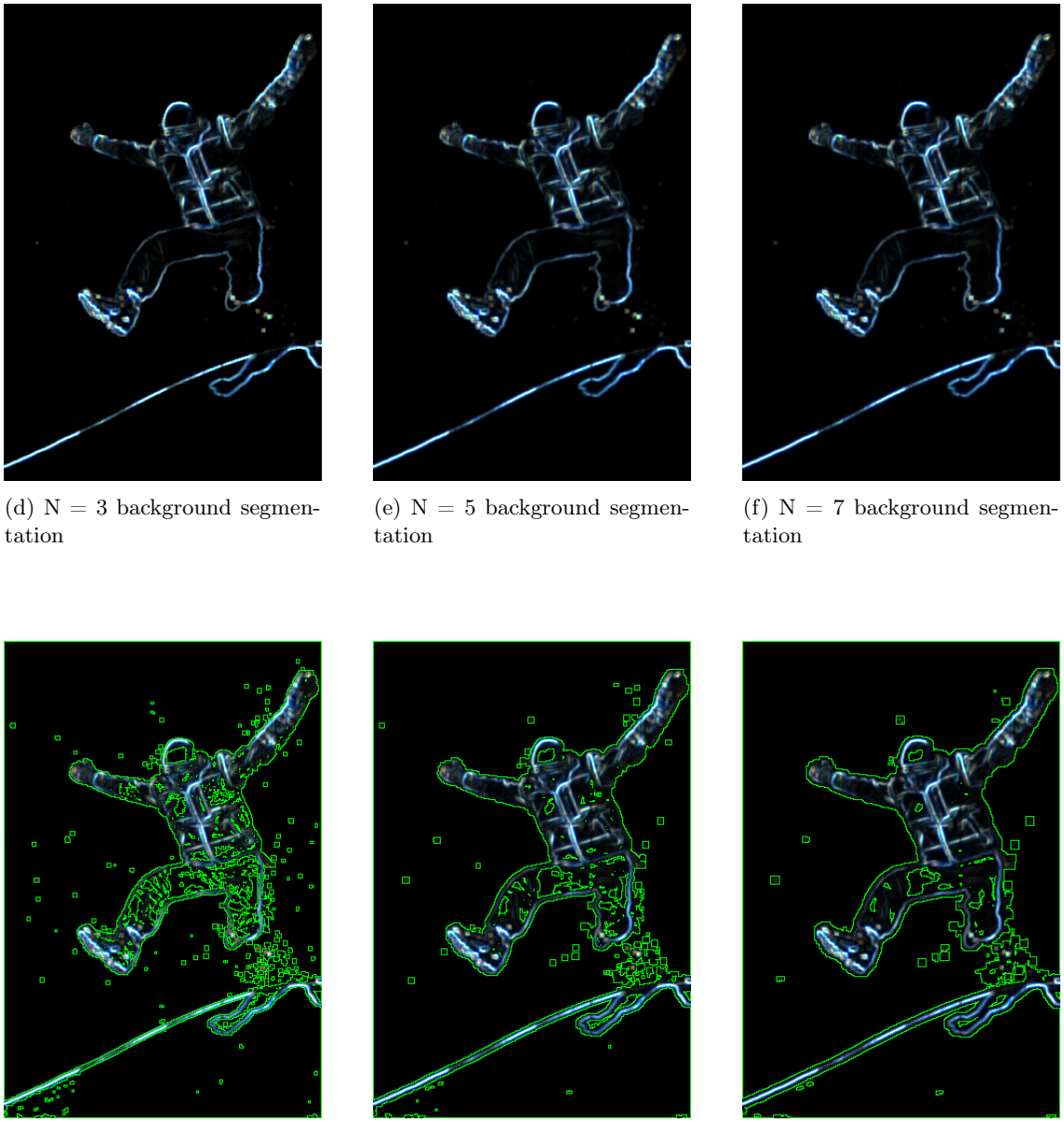
3.3.2 Texture Based Segmentation

(a) N = 3 Texture Image (b) N = 5 Texture Image (c) N = 7 Texture Image

Figure 13: Output of the Variance Image



(a) N = 3 foreground segmentation (b) N = 5 foreground segmentation (c) N = 7 foreground segmentation



(d) N = 3 background segmentation

(e) N = 5 background segmentation

(f) N = 7 background segmentation

(g) N = 3 contour segmentation

(h) N = 5 contour segmentation

(i) N = 7 contour segmentation

Figure 14: Step Output of TBS Algorithm

(a) Combined foreground segmentation

(b) Combined background segmentation

(c) Combined contour segmentation

Figure 15: 1st Iteration of TBS Algorithm

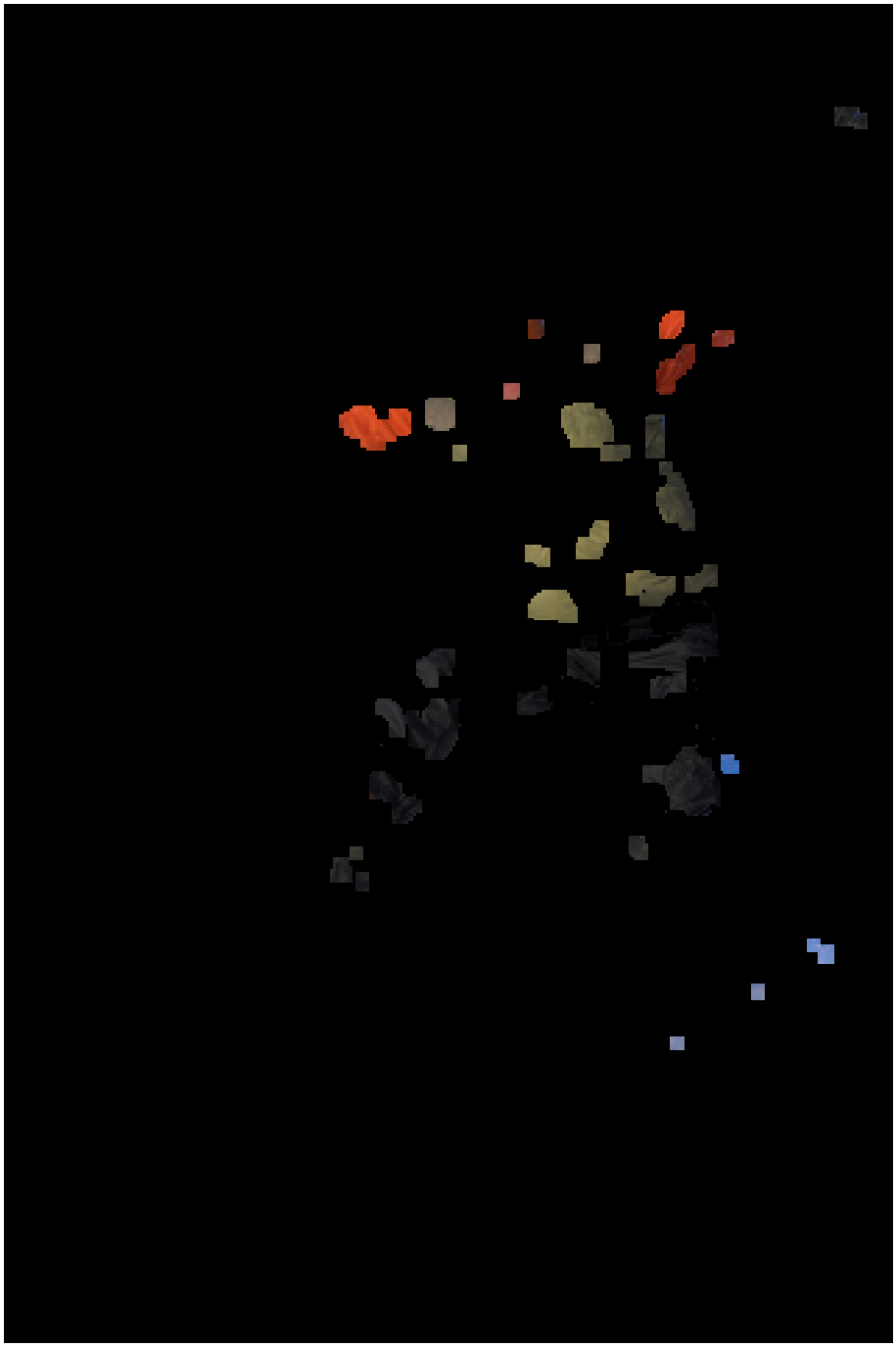


Figure 15: 2nd Iteration Result of ski.jpg of TBS Algorithm

4 Source Code

The code is written in Matlab 2018b and Windows 10.

4.1 Main Script

```
%% ECE 661 2018 Fall Homework 6
% Ye Shi
% shi349@purdue.edu

close all;clear; clc;

%% Import images
```

```
baby = imread("baby.jpg");
lighthouse = imread("lighthouse.jpg");
ski = imread("ski.jpg");

%% Otsu Segmentation using RGB values
FinalPlot(baby,'Otsu','baby')
FinalPlot(lighthouse,'Otsu','lighthouse')
FinalPlot(ski,'Otsu','ski')

%% Otsu Segmentation using RGB values 2nd iteration
babyO2 = imread("baby_Otsu_foreground.png");
lighthouseO2 = imread("lighthouse_Otsu_foreground.png");
skiO2 = imread("ski_Otsu_foreground.png");
FinalPlot(babyO2,'Otsu','baby_2nd')
FinalPlot(lighthouseO2,'Otsu','lighthouse_2nd')
FinalPlot(skiO2,'Otsu','ski_2nd')

%% Otsu Segmentation using RGB values 3rd iteration
babyO3 = imread("baby_2nd_Otsu_foreground.png");
lighthouseO3 = imread("lighthouse_2nd_Otsu_foreground.png");
skiO3 = imread("ski_2nd_Otsu_foreground.png");
FinalPlot(babyO3,'Otsu','baby_3rd')
FinalPlot(lighthouseO3,'Otsu','lighthouse_3rd')
FinalPlot(skiO3,'Otsu','ski_3rd')



%% Texture Based Segmentation using 3 window size (3,5,7)

FinalPlot(baby,'TBS','baby')
FinalPlot(lighthouse,'TBS','lighthouse')
FinalPlot(ski,'TBS','ski')

%% Texture Based Segmentation using 3 window size (3,5,7) 2nd Iteration
babyT2 = imread("baby_TBS_foreground.png");
lighthouseT2 = imread("lighthouse_TBS_foreground.png");
skiT2 = imread("ski_TBS_foreground.png");
FinalPlot(babyT2,'TBS','baby_2nd')
FinalPlot(lighthouseT2,'TBS','lighthouse_2nd')
FinalPlot(skiT2,'TBS','ski_2nd')

%% Texture Based Segmentation using 3 window size (3,5,7) 3rd Iteration
babyT3 = imread("baby_2nd_TBS_foreground.png");
lighthouseT3 = imread("lighthouse_2nd_TBS_foreground.png");
skiT3 = imread("ski_2nd_TBS_foreground.png");
FinalPlot(babyT3,'TBS','baby_3rd')
FinalPlot(lighthouseT3,'TBS','lighthouse_3rd')
FinalPlot(skiT3,'TBS','ski_3rd')
```

4.2 Segmentation Call Function

```
%% ECE 661 2018 Fall Homework 6
% Ye Shi
% shi349@purdue.edu

function FinalPlot(Img,Method,filename)
% Plot the foreground, background and contour images

if strcmp(Method, 'Otsu')

    filename = [filename,'_Otsu'];
    Mask =  Otsu(Img,filename);
elseif strcmp(Method,'TBS')

    filename = [filename,'_TBS'];
    Mask =  TextureSeg(Img,filename);
end
SegPlot(Img,Mask,filename)
end
```

4.2.1 Otsu Algorithm

```
%% ECE 661 2018 Fall Homework 6
```

```
% Ye Shi
% shi349@purdue.edu

function Mask =  Otsu(Img,filename)
% Otsu algorithm
[w,h,c] = size(Img);
K = zeros(1,c);
Masks = cell(1,3); % Foreground Mask
Ch = ['R','G','B'];
for C = 1:c

    P = zeros(1,256); % Probability density for gray lelvel
    D = zeros(1,256); % Between class variance
    I = Img(:,:,C); % Current gray scale image
    N = sum(I ~=0,'all'); % total pixels
    for k = 0:255
        P(k+1) = sum(I==k & I ~=0,'all')/N;
    end
    for k = 0:255
        W0 = sum(P(1:k));
        W1 = sum(P(k+1:256));
        U0 = sum(I(find(I<=k)),'all')/sum(I<=k & I ~=0,'all');
        U1 = sum(I(find(I>k)),'all')/sum(I>k,'all');
        D(k+1) = W0*W1*(U0 - U1)^2;
    end
    [~,K(C)] = max(D);
    Masks{C} = I >= K(C);
    SegPlot(Img,Masks{C},[filename,Ch(C)]);

end

Mask = ones(w,h);


for C = 1:c
    Mask  = Mask & Masks{C};
end

if 1
    Mask = double(Mask);
    % Reduce some noise
    SE = strel('ball',10,10);
    Mask = imdilate(Mask,SE);
%     Mask = imdilate(Mask,SE);
%     Mask = imerode(Mask,SE);
    Mask = imerode(Mask,SE);
    Mask = Mask>0;
end

end
```

4.2.2 Texture Based Algorithm

```
%% ECE 661 2018 Fall Homework 6
% Ye Shi
% shi349@purdue.edu

function mask = TextureSeg(Img,filename)
% Texture based segmentation
N = [3,5,7]; % window sizes
[h,w,c] = size(Img);
if c ==3
GrayI = double(rgb2gray(Img));
end
VarI = zeros(h,w,c);

for n = 1:3
    % Find the variance of each window and normalize it to  [0,255] for
    % Otsu
    VarI(:,:,n) = normalize( stdfilt(GrayI, true(N(n))).^2, 'range').*255;
    imwrite(uint8(VarI(:,:,n)),[filename,'_n',num2str(N(n)),'.png']);
end
mask =  Otsu(Img,filename);
end
```

4.2.3 Segmentation Plot Function

```
%% ECE 661 2018 Fall Homework 6
% Ye Shi
% shi349@purdue.edu

function SegPlot(Img,Mask,filename)
% Plot the foreground, background and contour images
contour = Contour( Mask );
[h,w,c] = size(Img);
Fimg = zeros(h,w,c);
Bimg = zeros(h,w,c);
Cimg = Img;
for i = 1:h
    for j = 1:w
        if Mask(i,j) == 0
%             for C = 1:c
%                 Fimg(i,j,C) = Img(i,j,C);
%             end
            Fimg(i,j,:) = Img(i,j,:);
        else
%             for C = 1:c
%                 Bimg(i,j,C) = Img(i,j,C);
%             end

              Bimg(i,j,:) = Img(i,j,:);
        end
        if contour(i,j,:) == 1
            Cimg(i,j,2) = 0;
            Cimg(i,j,1) = 255;
            Cimg(i,j,3) = 0;
        end
    end
end
imwrite(uint8(Fimg),[filename,'_foreground.png']);
% imwrite(uint8(Bimg),[filename,'_background.png']);
% imwrite(uint8(Cimg),[filename,'_contour.png']);
```

4.2.4 Contour Extraction

```
%% ECE 661 2018 Fall Homework 6
% Ye Shi
% shi349@purdue.edu

function contour = Contour( Mask )
% Contour Extraction
[m,n] = size(Mask);
contour = zeros(m,n);
N = 3; % 3x3 window size
halfWin = floor((N-1)/2);
Mask = padarray(Mask,[(N-1)/2 (N-1)/2]);
for i = 1:m
    for j = 1:n
        if(Mask(halfWin+i,halfWin+j) == 0)
            contour(i,j) = 0;
        else
            window = Mask(halfWin+(i-halfWin):halfWin+(i+halfWin),halfWin+(j-
                halfWin):halfWin+(j+halfWin));
            if(sum(window(:)) ~= N*N) % take out the inner part
                contour(i,j) = 1;
            end
        end
    end
end
end
```