ECE661: Computer Vision (Fall 2014)

Shaobo Fang: s-fang@purdue

October 27, 2014

Contents

1 Overview

Initially the segmentation algorithm is implemented based on data type 8-bit unsigned integer. Since there might be some difference between the results generated using datatype `uint8` and `double`, the results based on type `double` have also been obtained using the exact same algorithm and attached in section 7.1.

After observation of segmentation result based on different data type, it can be concluded that although the different data type has significant difference regarding segmentation based on RGB values, it does not change that much for the texture based segmentation.

In this assignment image segmentation will be performed on two task images: the `Lake` image and the `Tiger` image. Based on the perception of human observer, we will assume that in the `Lake` image, image is the foreground and the forest is the background. In the `Tiger` image, we will assume that the tiger is the foreground and the trees, etc are the background.

Our task is to extract the foreground out of the whole image from background. After the foreground and the background are segmented, we then will extract the contour using **8 points neighbourhood condition**.

For the image segmentation, two different approaches will be attempted. The first approach of segmentation is based on the RGB values. This approach would be good if we can distinguish the foreground from background solely based on colors. The second approach is based on texture, while texture is defined using the variance of $N \times N$ window at certain pixel location $f(i, j)$. **In both approach, the threshold values used for segmentation is obtained using Otsu's algorithm**.

Note that in order to optimize the result, multiple iterations will be performed while our segmentation is working in an iterative manner.

2 Otsu's Algorithm

Otsu's algorithms is designed so that the optimal threshold will be picked to distinguish foreground from background.

Assume there are total of L different gray levels in a monotone image and the total number of pixel is N, and there are n_i pixels associated with each of the gray level. Then, easily we can calculate the probability p_i that a random sampled pixel is from i^{th} gray level. Denote as:

$$p_i = \frac{n_i}{N}$$

$$where\ p_i \geq 0,\ \sum_{i=1}^{L} p_i = 1\ and\ \sum_{i=1}^{L} n_i = N$$

Assume there are two classes $\mathcal{C}_0$ and $\mathcal{C}_1$ which denote background and foreground. Assume the threshold value in gray level to distinguish foreground and background is k. Thus, we have zero-order moment:

$$w_0 = Pr(\mathcal{C}_0) = \sum_{i=1}^{k} p_i = w(k)$$

$$w_0 = Pr(\mathcal{C}_1) = \sum_{i=k+1}^{L} p_i = 1 - w(k)$$

Accordingly, the conditional mean (first order moment) is:

$$\mu_0 = \sum_{i=1}^{k} i \times Pr(i|\mathcal{C}_0) = \sum_{i=1}^{k} \frac{ip_i}{w_0} = \frac{\mu(k)}{w(k)}$$

$$\mu_0 = \sum_{i=k+1}^{L} i \times Pr(i|\mathcal{C}_0) = \sum_{i=k+1}^{N} \frac{ip_i}{w_1} = \frac{\mu_T - \mu(k)}{1 - w(k)}$$

$$Where:\ w(k) = \sum_{i=1}^{k} p_i,\ \mu(k) = \sum_{i=1}^{k} ip_i,\ \mu_T = \mu(L) = \sum_{i=1}^{L} ip_i$$

Our task is to find the value k that maximize the **within-class variance**:

$$\hat{k} = \arg\max_k \sigma_B^2,\quad where\ \sigma_B^2 = w_0(\mu_0 - \mu_T)^2 + w_1(\mu_1 - \mu_T)^2$$

Thus:

$$\hat{k} = \arg\max_k \{w_0(\mu_0 - \mu_T)^2 + w_1(\mu_1 - \mu_T)^2\}$$

Simplify furthermore:

$$\hat{k} = \arg\max_k \{w_0 w_1(\mu_1 - \mu_0)^2\}$$

Note that in this implementation, σ_B^2 is required for each value k.

3 Image Segmentation Using RGB Values

In this approach, we will perform image segmentation based on R,G,B values **separately** using Otsu's algorithm. This implementation would require logic `AND` operation.

First, set logic indicator at each pixel value to be "**1**" at each pixel location. Now, process each of the R,G,B channel images using Otsu's algorithm, we will obtain three threshold values:

$$T_{kR},\ T_{kG},\ T_{kB} \in [0, 1, 2, ..., 255]$$

Based on specific case, (for example in `Lake` image in the first iteration), set logic operator corresponding to each RGB channel to be **1** (if the RGB channel value is larger than T_{kR}, T_{kG}, T_{kB} accordingly), otherwise set the logic operator to be **0**.

Now, we perform logic **AND** on all pixel locations. **This means the output binary image would have value 1 at a pixel location if and only if all logic operators for R,G,B at this pixel locations are 1!**

Hence, the segmented image is actually the binary image after logic **AND**.

In order to optimize the results, we will carry out the current binarized image for second iteration. Note that all the pixel locations with **0** is **NO LONGER** considered. Thus we set all the pixel values in those locations in R,G,B images to be 0 before second iteration and ignore those pixel locations that has value **0** applying Otsu's algorithm.

4 Texture Based Segmentation

In order to convert the color image into a gray scale image for texture based image segmentation, we follow the equation of CIE709 standard:

$$Y_{709} = 0.2125 \times R + 0.7154 \times G + 0.0721 \times B$$

Texture based image segmentation is very similar to the approach based on R,G,B channels. The only difference is that in this approach the segmentation is mainly based on variance of window $N \times N$.

We pick $N = 3, 5, 7, 9$. Thus, similarly to the previous approach, we obtain the logic operator for different window size and perform logic **AND**.

Data Normalization: It is extremely important to normalize the variance data in this case. As the variance data would vary a lot from case to case. **In my implementation, I substitute the variance with standard deviation to decrease the variations in the data**. This has turned out to be a good way proved by the result of the first one: (`Lake` image). Very likely even if the feature used is standard deviation, the data is still out of bound. One way to solve the this problem is to rescale the data of standard deviation to the range of $[0, 1, ..., 255]$, and substitute those high values with 255.

Similarly, iterative approach will be utilized if segmentation results could be optimized.

5 Contour Extraction

Contour extraction is based on 8 points neighbourhood condition.
In order to qualify for a contour pixel, the following three criteria need to be met:

1. At the pixel location, the logic indicator is **1**
2. The sum of the 8 points neighbourhood('s logic operators) has to be larger than or equal to 3 to eliminate those noises
3. The sum of the 8 points neighbourhood('s logic operators) has to be smaller than or equal to 5 to be qualified as a contour pixel

6 Parameters Set-Up and Results Discussion

6.1 `Lake` Image Result

For the `Lake` image, we perform two iterations of image segmentation on both the RGB-based segmentation and the variance-based segmentation. It has been proven that the iterative implementation would optimize our results. The improvement could be easily observed in the following section.

Method	*Iteration*	*Parameter*	*Otsu Threshold*	*Segmentation Criteria*
RGB Based	$1st$	*Red*	$T_{kR} = 57$	$f_R(i,j) \geq T_{kR}$
RGB Based	$1st$	*Green*	$T_{kG} = 63$	$f_G(i,j) \geq T_{kG}$
RGB Based	$1st$	*Blue*	$T_{kB} = 69$	$f_B(i,j) \geq T_{kB}$
RGB Based	$2nd$	*Red*	$T_{kR} = 159$	$f_R(i,j) \leq T_{kR}$
RGB Based	$2nd$	*Green*	$T_{kG} = 132$	*Did Not Care*
RGB Based	$2nd$	*Blue*	$T_{kB} = 153$	$f_B(i,j) \geq T_{kB}$
Variance Based	$1st$	3×3	$T_{k3\times3} = 67$	$f_{3\times3}(i,j) \leq T_{k3\times3}$
Variance Based	$1st$	5×5	$T_{k5\times5} = 78$	$f_{5\times5}(i,j) \leq T_{k5\times5}$
Variance Based	$1st$	7×7	$T_{k7\times7} = 79$	$f_{7\times7}(i,j) \leq T_{k7\times7}$
Variance Based	$1st$	9×9	$T_{k9\times9} = 79$	$f_{9\times9}(i,j) \leq T_{k9\times9}$
Variance Based	$2nd$	3×3	$T_{k3\times3} = 28$	$f_{3\times3}(i,j) \leq T_{k3\times3}$
Variance Based	$2nd$	5×5	$T_{k5\times5} = 32$	$f_{5\times5}(i,j) \leq T_{k5\times5}$
Variance Based	$2nd$	7×7	$T_{k7\times7} = 17$	$f_{7\times7}(i,j) \leq T_{k7\times7}$
Variance Based	$2nd$	9×9	$T_{k9\times9} = 24$	$f_{9\times9}(i,j) \leq T_{k9\times9}$

Table 1. The Parameters used for `Lake` image. Please note that the segmentation criteria is selected **solely for the purpose of optimizing the result**. That is why the segmentation criteria might change for different iterations.

6.2 `Tiger` Image Result

For the Tiger image, we perform two iterations of image segmentation based on RGB values to obtain a optimized result. However, for the image segmentation based on variance, one iteration would be enough as the second iteration is not optimizing our result (it actually become worse).

Method	*Iteration*	*Parameter*	*Otsu Threshold*	*Segmentation Criteria*
RGB Based	$1st$	*Red*	$T_{kR} = 28$	$f_R(i,j) \geq T_{kR}$
RGB Based	$1st$	*Green*	$T_{kG} = 67$	$f_G(i,j) \geq T_{kG}$
RGB Based	$1st$	*Blue*	$T_{kB} = 50$	$f_B(i,j) \geq T_{kB}$
RGB Based	$2nd$	*Red*	$T_{kR} = 105$	$f_R(i,j) \geq T_{kR}$
RGB Based	$2nd$	*Green*	$T_{kG} = 247$	$f_G(i,j) \leq T_{kG}$
RGB Based	$2nd$	*Blue*	$T_{kB} = 124$	$f_B(i,j) \geq T_{kB}$
Variance Based	$1st$	3×3	$T_{k3\times3} = 77$	$f_{3\times3}(i,j) \geq T_{k3\times3}$
Variance Based	$1st$	5×5	$T_{k5\times5} = 104$	$f_{5\times5}(i,j) \geq T_{k5\times5}$
Variance Based	$1st$	7×7	$T_{k7\times7} = 98$	$f_{7\times7}(i,j) \geq T_{k7\times7}$
Variance Based	$1st$	9×9	$T_{k9\times9} = 139$	$f_{9\times9}(i,j) \geq T_{k9\times9}$

Table 2. The Parameters used for `Tiger` image. Please note that the segmentation criteria is selected **solely for the purpose of optimizing the result**. That is why the segmentation criteria might change for different iterations.

Note that it has been proven by results that second iteration for `Tiger` image will actually worsen the result using texture based segmentation. Thus we only perform one iteration.

7 Results: `8-bit Unsigned Integer` and `Double`

Very Important Notice: Throughout the program the data type used is **8-bit unsigned integer**. Thus the result would look different from others' if they were saving using **double** or **float**. However, the final result looks correct. So the different data type would only affect the appearance of intermediate output. The difference in intermediate output is mainly due to different threshold returned by Otsu's algorithm. First of all I will attach the result of segmented image using **double** data type. The advantage of using **8-bit unsigned integer** is, it is a lot easier to check for different gray-level within the image.

7.1 Appendix: Extra Results Obtained Using `Double`

Fig I. The Segmented Image Based on RGB Values Using `Double` data type. From left to right: Red, Green, Blue channels.

Fig II. The segmentation based on **RGB values**. `One iteration. 'Double' data type`

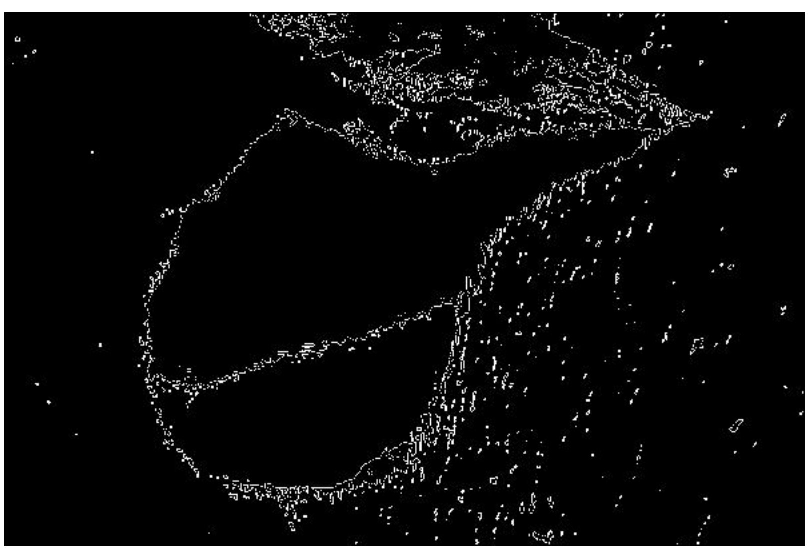

Fig III. The contour extracted from segmentation based on **RGB values**. `One iteration. 'Double' datatype`

Fig IV. The segmentation based on **texture(variance)**. `First iteration. 'Double' datatype`

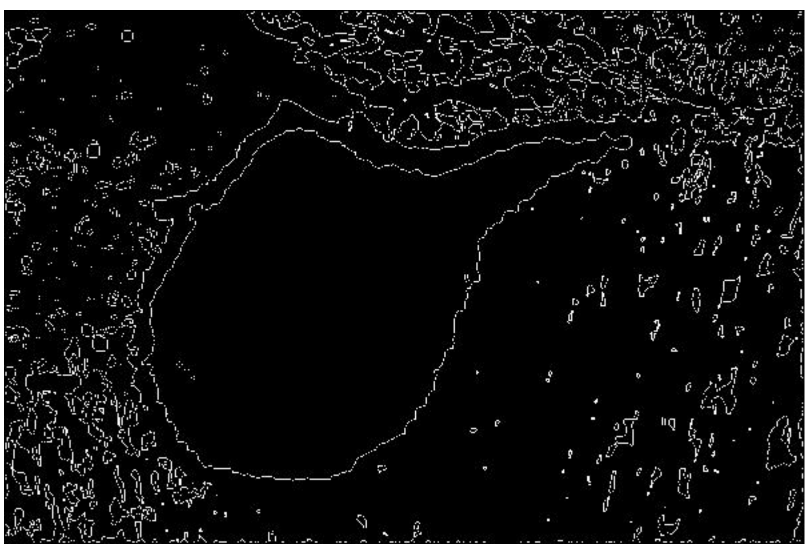

Fig V. The contour extracted from the segmentation based on **texture(variance)**. `First iteration. 'Double' datatype`

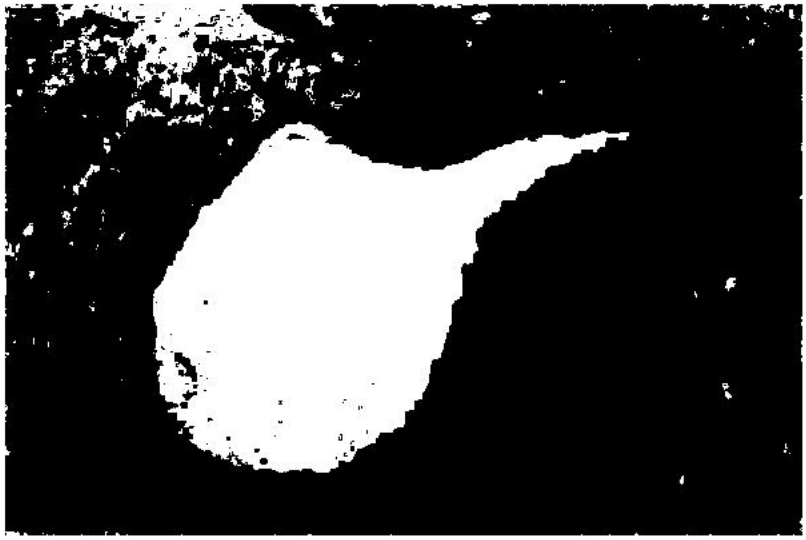

Fig VI. The segmentation based on **texture(variance)**. `Second iteration. 'Double' datatype`

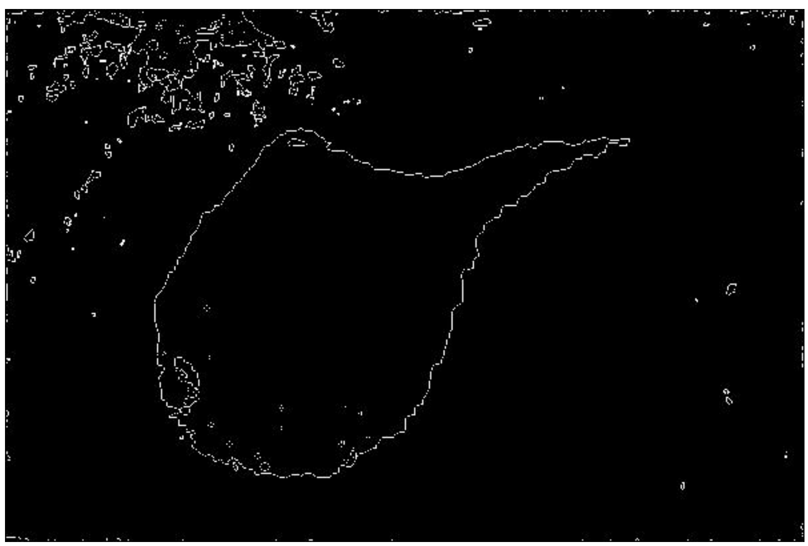

Fig VII. The contour extracted from the segmentation based on **texture(variance)**. `Second iteration. 'Double' datatype`

Comparing the segmented image based on texture to those segmented based on RGB values, the lake would not be 'crossed' in the middle. Although the different data type has significant difference regarding segmentation based on RGB values, it does not change that much for the texture based segmentation.

Fig VIII. The Segmented Image Based on RGB Values Using `Double` data type. From left to right: Red, Green, Blue channels.

Fig IX. The segmentation based on **RGB values**. `One iteration. 'Double' data type`

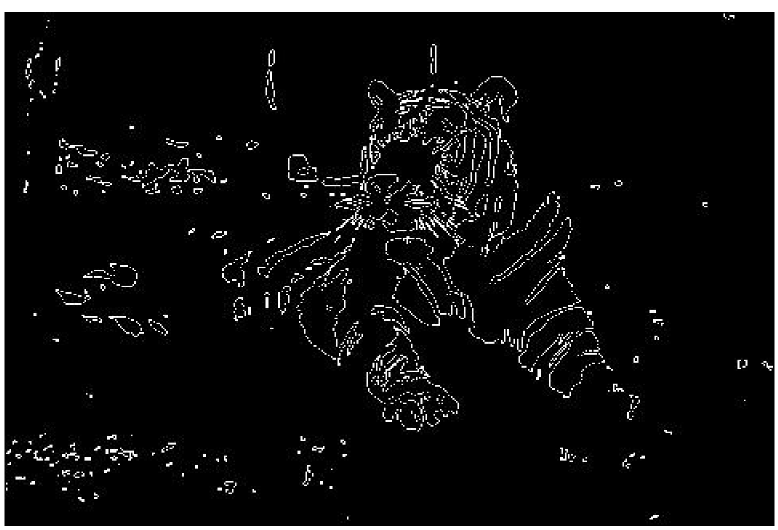

Fig X. The contour extracted from segmentation based on **RGB values**. `One iteration. 'Double' datatype`

Fig XI. The segmentation based on **texture(variance)**. `First iteration. 'Double' datatype`

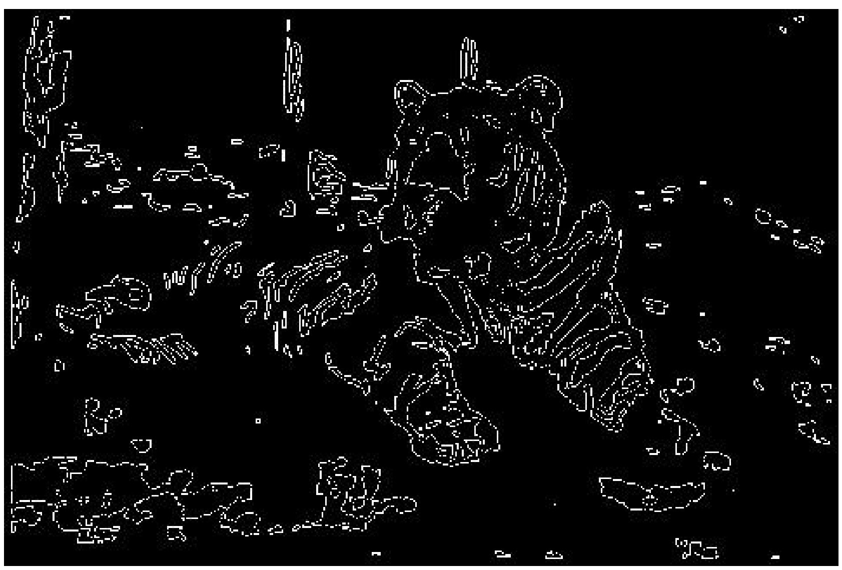

Fig XII. The contour extracted from the segmentation based on **texture(variance)**. `First iteration. 'Double' datatype`

Fig XIII. The segmentation based on **texture(variance)**. `Second iteration. 'Double' datatype`

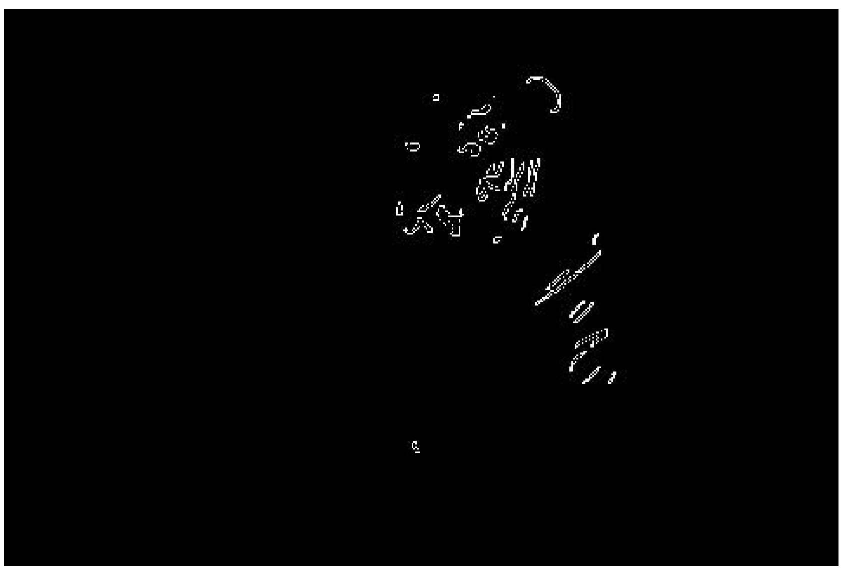

Fig XIV. The contour extracted from the segmentation based on **texture(variance)**. `Second iteration. 'Double' datatype`

Although the different data type has significant difference regarding segmentation based on RGB values, it does not change that much for the texture based segmentation.

7.2 The `Lake` Image: `8-bit Unsigned Integer`

Fig 1. The original lake image

7.2.1 RGB Based Image Segmentation

Fig 2. The segmented images based on **Red** channel. `First Iteration`

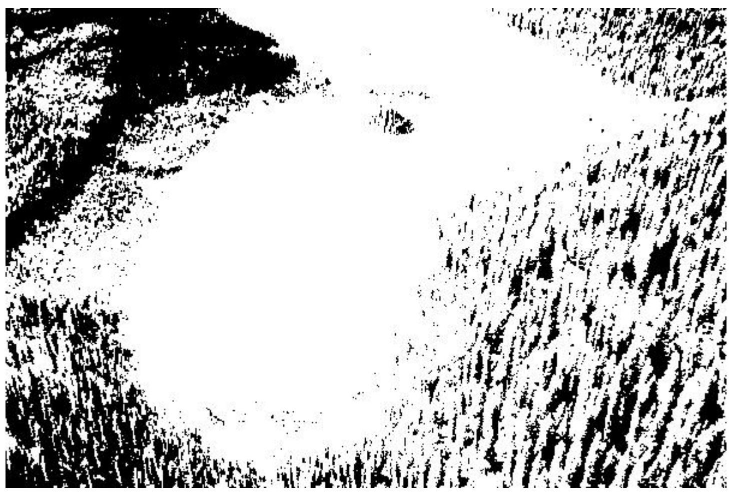

Fig 3. The segmented images based on **Green** channel. `First Iteration`

Fig 4. The segmented images based on **Blue** channel. `First Iteration`

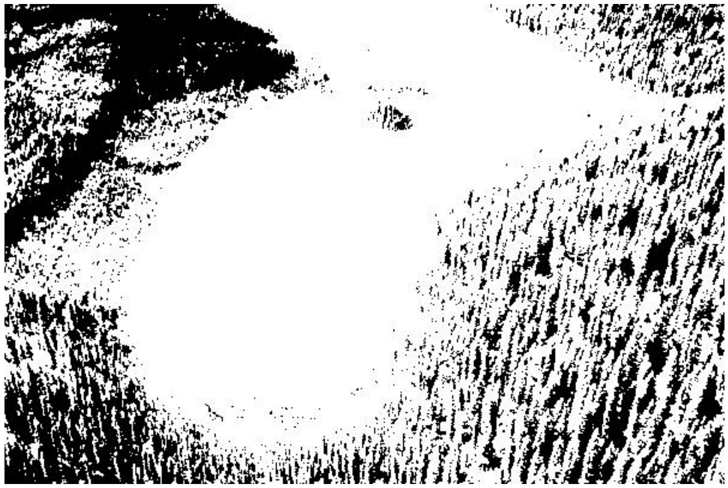

Fig 5. The segmented images based on **Red, Green, Blue** channels. `First Iteration`

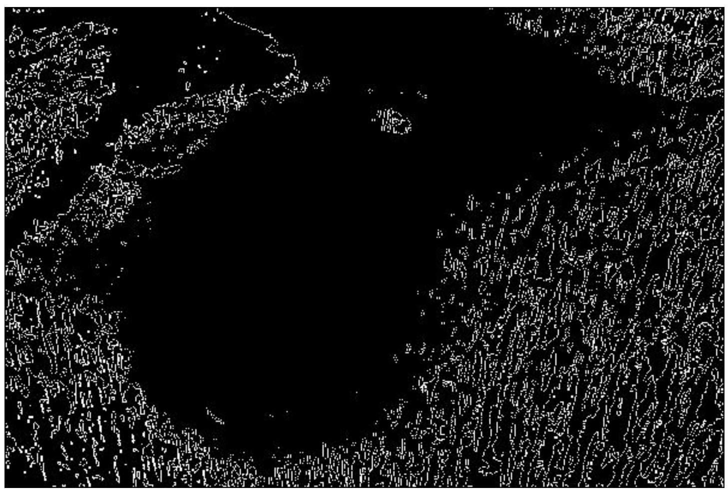

Fig 6. The contour extraction of segmented images based on **Red, Green, Blue** channels. `First Iteration`

Based on the results above, we can concluded that the results could be optimized, thus we will perform a second iteration based on RGB values.

First iteration end, start second iteration.

Fig 7. The segmented images based on **Red** channel. `Second Iteration`: Note that we are picking $f_R(i,j) < T_{kR}$ as segmentation criteria.

Fig 8. The segmented images based on **Green** channel. `Second Iteration`: We did not care about green channel in our second iteration.

Fig 9. The segmented images based on **Blue** channel. `Second Iteration`

Fig 10. The segmented images based on **Red, Green, Blue** channels. `Second Iteration`

Fig 11. The contour extraction of segmented images based on **Red, Green, Blue** channels. `Second Iteration`

Note that except for that the lower part of the lake was cut-off inaccurately, other part works perfectly fine. The lower corner of the lake was cut-off mainly because in those area the water appear darker compared to the other locations.

7.2.2 Texture Based Image Segmentation

Fig 12. The converted gray scale image used for texture based image segmentation.

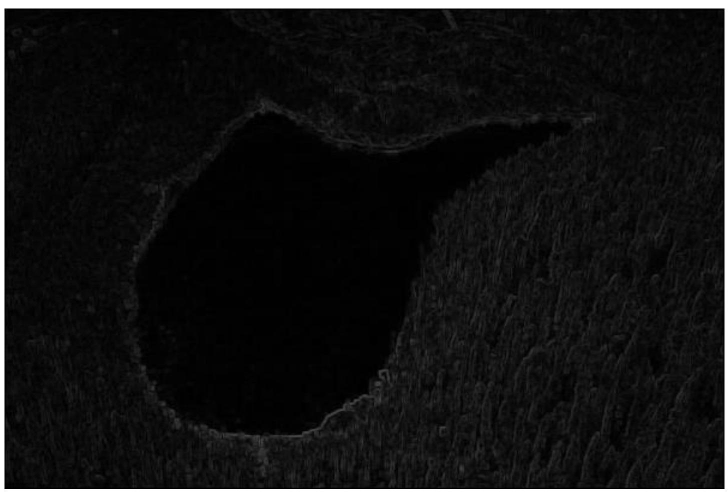

Fig 13. The texture of 3×3 window

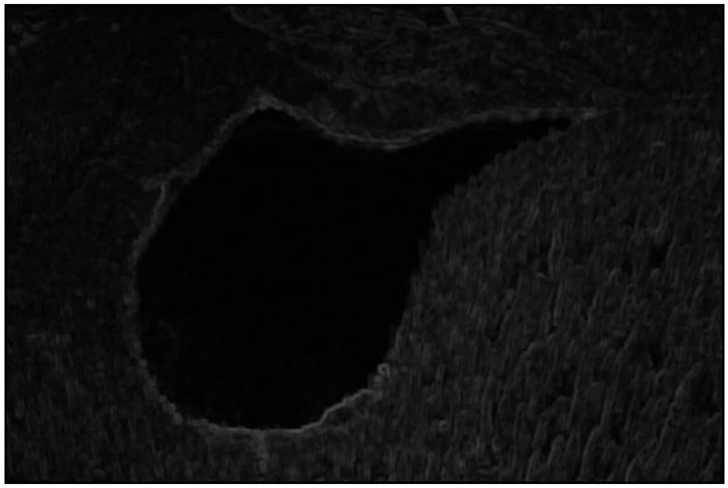

Fig 14. The texture of 5×5 window

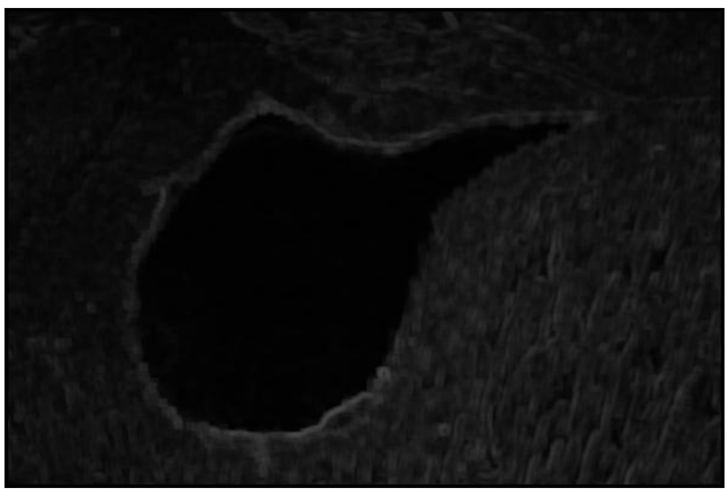

Fig 15. The texture of 7×7 window

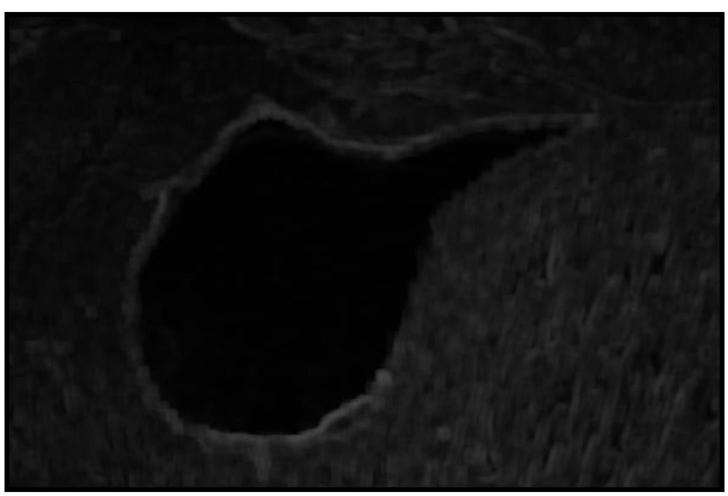

Fig 16. The texture of 9×9 window

First iteration of texture based image segmentation result.

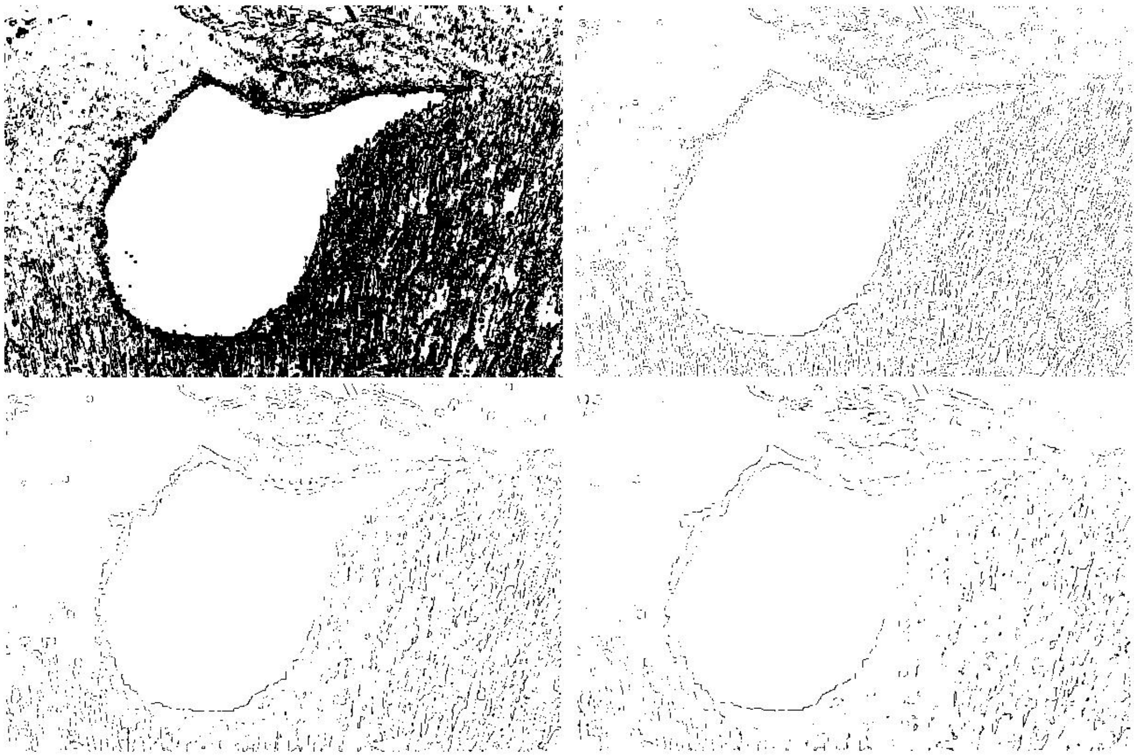

Fig 17. Logic operators at each pixel location. Window Size: Upper Left 3×3, Upper Right 5×5, Lower Left 7×7, Lower Right 9×9.

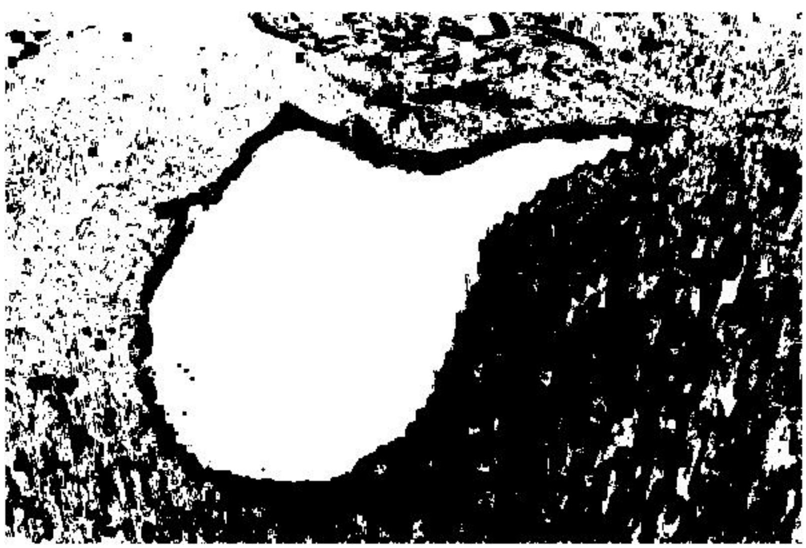

Fig 18. The segmented image using texture based image segmentation.

Fig 19. The extracted contour of the segmented image using texture based image segmentation.

Second iteration of texture based image segmentation result.

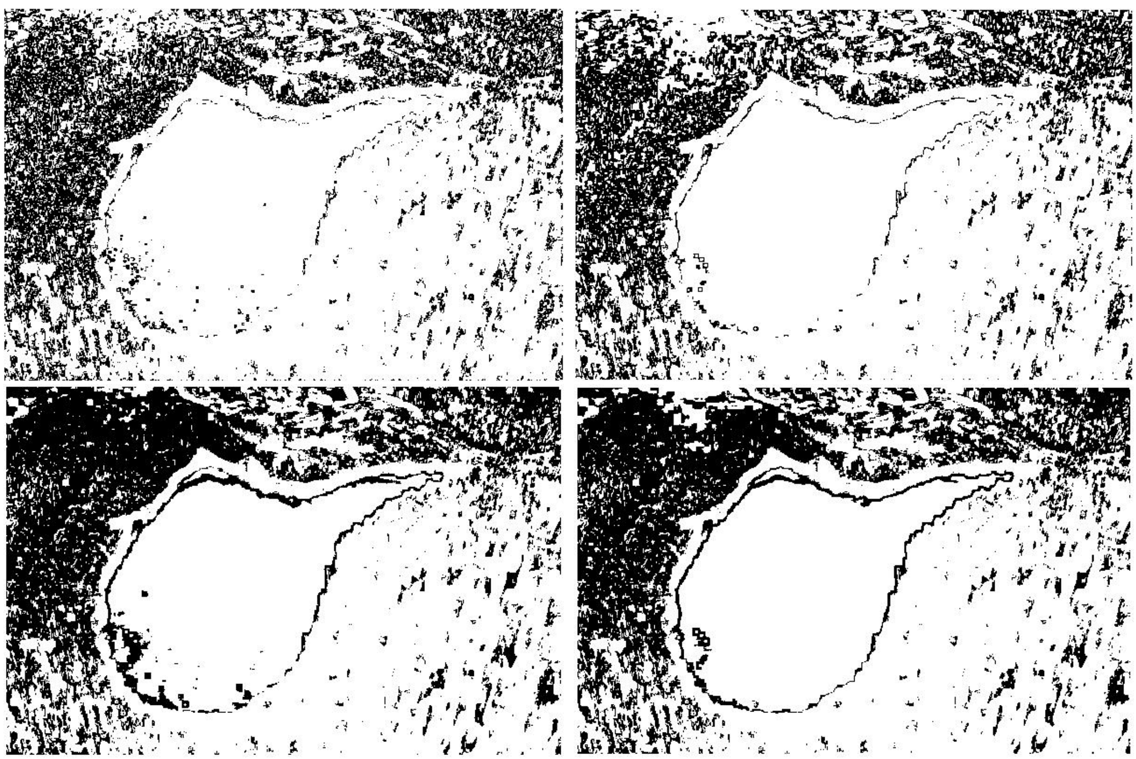

Fig 20. Logic operators at each pixel location. Window Size: Upper Left 3×3, Upper Right 5×5, Lower Left 7×7, Lower Right 9×9.

Fig 21. The segmented image using texture based image segmentation.

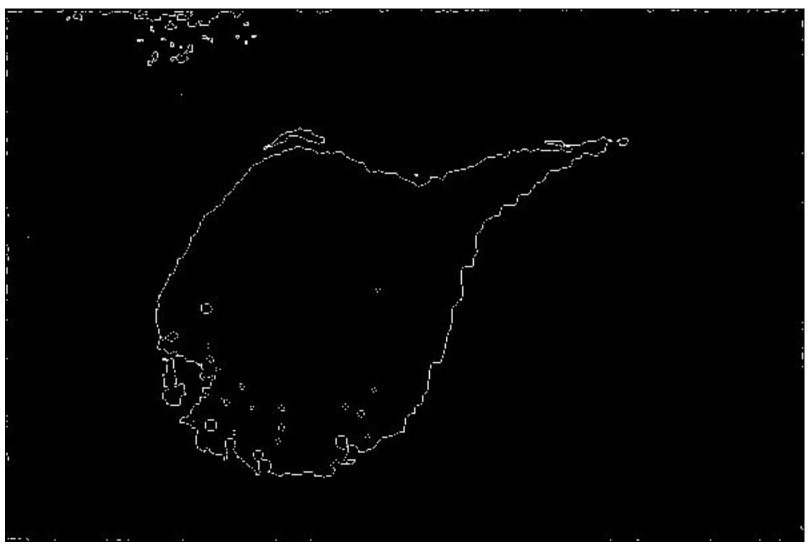

Fig 22. The extracted contour of the segmented image using texture based image segmentation.

Discussion: It could be easily seen that after second iteration the results improved. The advantage of texture based segmentation is that the lower part of the lake was preserved (in color based image segmentation the dark part of the lake was cropped off).

7.3 The `Tiger` Image: `8-bit Unsigned Integer`

Fig 23. The original `Tiger` image.

7.3.1 RGB Based Image Segmentation

Fig 24. The segmented images based on **Red** channel. `First Iteration`

Fig 25. The segmented images based on **Green** channel. `First Iteration`

Fig 26. The segmented images based on **Blue** channel. `First Iteration`

Fig 27. The segmented images based on **Red, Green, Blue** channels. `First Iteration`

Fig 28. The contour extraction of segmented images based on **Red, Green, Blue** channels. `First Iteration`

The segmentation result based on RGB color is not that good. However this is expected because color is not the feature in the original image that separate the tiger from the background. We will apply a second iteration to see if result might improve

First iteration end, start second iteration.

Fig 29. The segmented images based on **Red** channel. `Second Iteration`

Fig 30. The segmented images based on **Green** channel. `Second Iteration`: Note that we are picking $f_G(i,j) \leq T_{kG}$ as segmentation criteria.

Fig 31. The segmented images based on **Blue** channel. `Second Iteration`

Fig 32. The segmented images based on **Red, Green, Blue** channels. `Second Iteration`

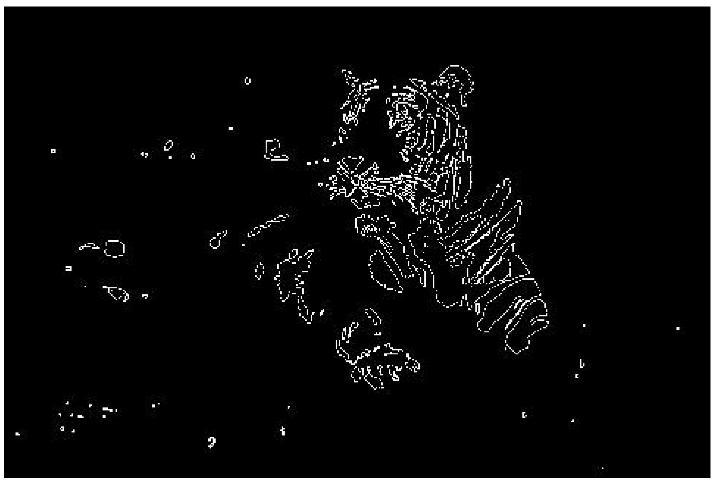

Fig 33. The contour extraction of segmented images based on **Red, Green, Blue** channels. `Second Iteration`

After second iteration a lot of background noise was suppressed. However only the face of the tiger was preserved well. As it is not the color features in original image that human 'segment' the tiger from the background, it is expected that the Otsu's algorithm would not work well.

Now let us check if segmentation based on feature would work well on second image.

7.3.2 Texture Based Image Segmentation

Fig 34. The converted gray scale image used for texture based image segmentation.

Fig 35. The texture of 3×3 window

Fig 36. The texture of 5×5 window

Fig 37. The texture of 7×7 window

Fig 38. The texture of 9×9 window

First iteration of texture based image segmentation result.

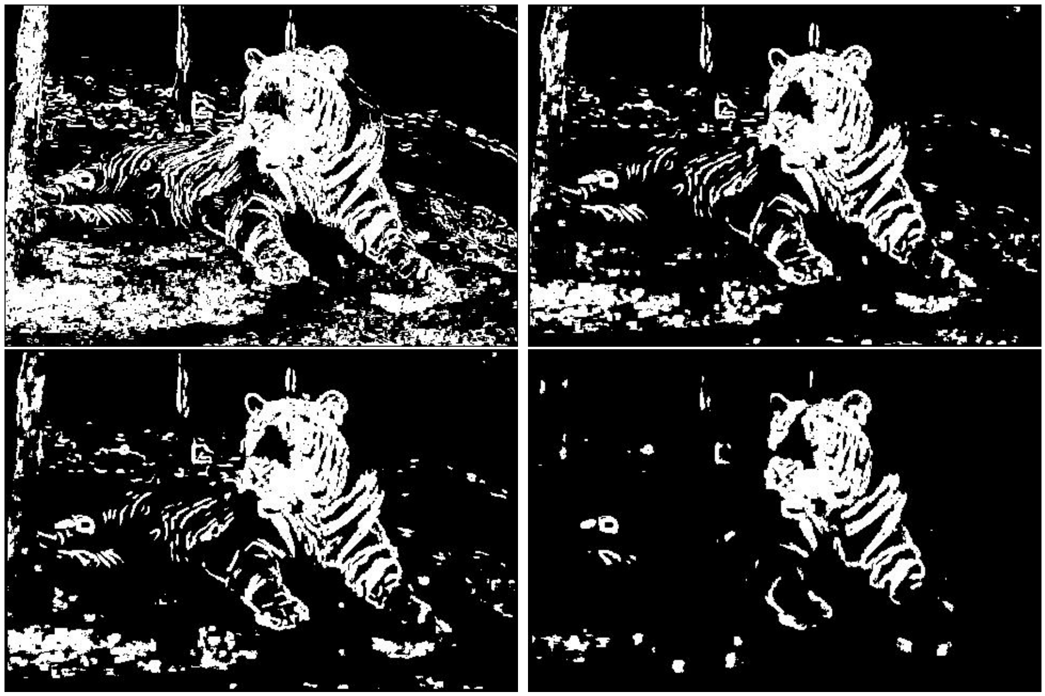

Fig 39. Logic operators at each pixel location. Window Size: Upper Left 3×3, Upper Right 5×5, Lower Left 7×7, Lower Right 9×9.

Fig 40. The segmented image using texture based image segmentation.

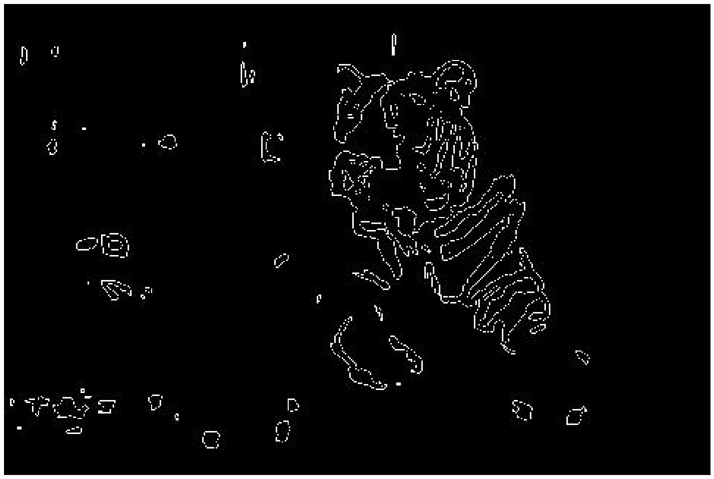

Fig 41. The extracted contour of the segmented image using texture based image segmentation.

Stop Here. If we apply the second iteration based on texture the result would be a lot worse

Fig 42. The extracted contour of the segmented image using texture based image segmentation. **Second Iteration, where the result become worse**.

8 Appendix: Matlab Code Used

8.1 Matlab Code: `main.m`

```
function [] = main( image_name )
close all; clc;
%% Main Function
% Load Image
% Otsu's Algorithms
% First Approach: R,G,B values based Segmentation
% Second Approach: Texture based Segmentation
% Contour Extraction

%% First Approach: R,G,B values based Segmentation   %%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Load Image For Processing
img = imread(image_name);
img_size = size(img);
imshow(img);

% Seperate Different Channels
imgr = (img(:,:,1)); % Red Channel
imgg = (img(:,:,2)); % Green Channel
imgb = (img(:,:,3)); % Blue Channel

% Find the size of the image
I_img  = ones(img_size(1),img_size(2));


% Find the threshold values using Otsu's algorithms
Tkr = Otsu(I_img,imgr)
Tkg = Otsu(I_img,imgg)
Tkb = Otsu(I_img,imgb)

% Initializing the Logic Operator, preparing for the image segmentation
I_imgr = zeros(img_size(1),img_size(2));
I_imgg = zeros(img_size(1),img_size(2));
I_imgb = zeros(img_size(1),img_size(2));
I_img  = ones(img_size(1),img_size(2));

% Performing Logic 'AND'. Update the pixel values in R,G,B channels for
% second iteration
for i = 1:1:img_size(1)
    for j = 1:1:img_size(2)

        if (imgr(i,j) >= Tkr)
            I_imgr(i,j) = 1;
        else
            imgr(i,j) = 0;
            imgg(i,j) = 0;
            imgb(i,j) = 0;
```

```
        end

        if (imgg(i,j) >= Tkg)
            I_imgg(i,j) = 1;
        else
            imgr(i,j) = 0;
            imgg(i,j) = 0;
            imgb(i,j) = 0;
        end

        if (imgb(i,j) >= Tkb)
            I_imgb(i,j) = 1;
        else
            imgr(i,j) = 0;
            imgg(i,j) = 0;
            imgb(i,j) = 0;
        end

    end
end

% Show the result after first iteration
figure
subplot(3,1,1)
imshow(I_imgr.*255);
subplot(3,1,2)
imshow(I_imgg.*255);
subplot(3,1,3)
imshow(I_imgb.*255);

I_img = I_img.*I_imgr.*I_imgg.*I_imgb;
figure
imshow(I_img);

% Save the images after 1st iteration
save_name = ['step1_result_Otsu_' image_name];
imwrite(I_img,save_name,'jpeg')
save_name = ['step1_result_Otsu_r_channel_' image_name];
imwrite(I_imgr,save_name,'jpeg')
save_name = ['step1_result_Otsu_g_channel_' image_name];
imwrite(I_imgg,save_name,'jpeg')
save_name = ['step1_result_Otsu_b_channel_' image_name];
imwrite(I_imgb,save_name,'jpeg')

% Extract the contour
contour_img = contour_extract(I_img);

% Save the contour after first iteration
save_name = ['step1_result_Otsu_contour_' image_name];
imwrite(contour_img,save_name,'jpeg')

% Perform second iteration using Otsu's algorithm
Tkr = Otsu(I_img,imgr)
Tkg = Otsu(I_img,imgg)
```

```
Tkb = Otsu(I_img,imgb)

% Re-initiate the logic operators
I_imgr = zeros(img_size(1),img_size(2));
I_imgg = zeros(img_size(1),img_size(2));
I_imgb = zeros(img_size(1),img_size(2));

% Performing Logic 'AND' to segment the images
for i = 1:1:img_size(1)
    for j = 1:1:img_size(2)

        if (imgr(i,j) >= Tkr)
            I_imgr(i,j) = 1;
        else
        end


        if (imgg(i,j) <= Tkg)
            I_imgg(i,j) = 1;
        else
        end

        if (imgb(i,j) >= Tkb)
            I_imgb(i,j) = 1;
        else
        end

    end
end

% Show the result after second iteration
figure
subplot(3,1,1)
imshow(I_imgr.*255);
subplot(3,1,2)
imshow(I_imgg.*255);
subplot(3,1,3)
imshow(I_imgb.*255);

I_img = I_img.*I_imgr.*I_imgb.*I_imgg;
save_name = ['step2_result_Otsu_' image_name];
figure
imshow(I_img);

% Save the images after 2nd iteration
imwrite(I_img,save_name,'jpeg')
save_name = ['step2_result_Otsu_r_channel_' image_name];
imwrite(I_imgr,save_name,'jpeg')
save_name = ['step2_result_Otsu_g_channel_' image_name];
imwrite(I_imgg,save_name,'jpeg')
save_name = ['step2_result_Otsu_b_channel_' image_name];
imwrite(I_imgb,save_name,'jpeg')

% Save the contour after second iteration
```

```
contour_img = contour_extract(I_img);
save_name = ['step2_result_Otsu_contour_' image_name];
imwrite(contour_img,save_name,'jpeg')

%% Second Approach: Texture based Segmentation   %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Y709 = 0.2125R + 0.7154G + 0.0721B
% Load Image For Processing
img = imread(image_name);
I_img = ones(img_size(1),img_size(2));
I_img_var_3 = zeros(img_size(1),img_size(2));
I_img_var_5 = zeros(img_size(1),img_size(2));
I_img_var_7 = zeros(img_size(1),img_size(2));
I_img_var_9 = zeros(img_size(1),img_size(2));
img_var_3 = zeros(img_size(1),img_size(2));
img_var_5 = zeros(img_size(1),img_size(2));
img_var_7 = zeros(img_size(1),img_size(2));
img_var_9 = zeros(img_size(1),img_size(2));

% Generate the gray-scale image for finding variances of local window
gray_img = zeros(img_size(1),img_size(2));
for i = 1:1:img_size(1)
    for j = 1:1:img_size(2)
        gray_img(i,j) = 0.2125*img(i,j,1) + 0.7154*img(i,j,2) + 0.0721*img(i,j,3);
        gray_img(i,j) = round(gray_img(i,j));
    end
end

% Show the gray-scale images generate
figure
imshow(uint8(gray_img));

% Save the gray-scale image
save_name = ['gray_scale_image_' image_name];
imwrite(uint8(gray_img), save_name, 'jpeg')

% Find the Standard Deviation for different N*N window, N = 3,5,7,9
for i = 2:1:img_size(1)-1
    for j = 2:1:img_size(2)-1
        img_var_3(i,j) = round(std(reshape(gray_img(i-1:1:i+1,j-1:1:j+1),1,9)));
    end
end

for i = 3:1:img_size(1)-2
    for j = 3:1:img_size(2)-2
        img_var_5(i,j) = round(std(reshape(gray_img(i-2:1:i+2,j-2:1:j+2),1,25)));
    end
end

for i = 4:1:img_size(1)-3
    for j = 4:1:img_size(2)-3
```

```
        img_var_7(i,j) = round(std(reshape(gray_img(i-3:1:i+3,j-3:1:j+3),1,49)
            ));
    end
end

for i = 5:1:img_size(1)-4
    for j = 5:1:img_size(2)-4
        img_var_9(i,j) = round(std(reshape(gray_img(i-4:1:i+4,j-4:1:j+4),1,81)
            ));
    end
end

% Show the Standard Deviation images
figure
imshow(uint8(img_var_3))
figure
imshow(uint8(img_var_5))
figure
imshow(uint8(img_var_7))
figure
imshow(uint8(img_var_9))

% Save the Standard Deviation images
save_name = ['var_3_image_' image_name];
imwrite(uint8(img_var_3), save_name, 'jpeg')
save_name = ['var_5_image_' image_name];
imwrite(uint8(img_var_5), save_name, 'jpeg')
save_name = ['var_7_image_' image_name];
imwrite(uint8(img_var_7), save_name, 'jpeg')
save_name = ['var_9_image_' image_name];
imwrite(uint8(img_var_9), save_name, 'jpeg')

% Find the mean and the median of var_N images in order to rescale the
% pixel value

mean_var_3 = mean(reshape(img_var_3,1,img_size(1)*img_size(2)))
mean_var_5 = mean(reshape(img_var_5,1,img_size(1)*img_size(2)))
mean_var_7 = mean(reshape(img_var_7,1,img_size(1)*img_size(2)))
mean_var_9 = mean(reshape(img_var_9,1,img_size(1)*img_size(2)))

median_var_3 = median(reshape(img_var_3,1,img_size(1)*img_size(2)))
median_var_5 = median(reshape(img_var_5,1,img_size(1)*img_size(2)))
median_var_7 = median(reshape(img_var_7,1,img_size(1)*img_size(2)))
median_var_9 = median(reshape(img_var_9,1,img_size(1)*img_size(2)))

% Rescaling the pixel value to optimize output
img_var_3 = img_var_3.*(64/mean_var_3);
img_var_5 = img_var_5.*(64/mean_var_5);
img_var_7 = img_var_7.*(64/mean_var_7);
img_var_9 = img_var_9.*(64/mean_var_9);

% Crop off the pixel values of var_N images so the result will be optimized
for i = 1:1:img_size(1)
    for j = 1:1:img_size(2)
```

```

        if (img_var_3(i,j) >= 255)
            img_var_3(i,j) = 255;
        else
            img_var_3(i,j) = round(img_var_3(i,j));
        end

        if (img_var_5(i,j) >= 255)
            img_var_5(i,j) = 255;
        else
            img_var_5(i,j) = round(img_var_5(i,j));
        end

        if (img_var_7(i,j) >= 255)
            img_var_7(i,j) = 255;
        else
            img_var_7(i,j) = round(img_var_7(i,j));
        end

        if (img_var_9(i,j) >= 255)
            img_var_9(i,j) = 255;
        else
            img_var_9(i,j) = round(img_var_9(i,j));
        end

    end
end

% Prepare var_N images for Otsu's Algorithms
img_var_3 = uint8(img_var_3);
img_var_5 = uint8(img_var_5);
img_var_7 = uint8(img_var_7);
img_var_9 = uint8(img_var_9);

figure
imshow(img_var_3)
figure
imshow(img_var_5)
figure
imshow(img_var_7)
figure
imshow(img_var_9)

% Performing Otsu's Algorithms on processed var_N images for threshold
% value
Tk_var_3 = Otsu(I_img,img_var_3)
Tk_var_5 = Otsu(I_img,img_var_5)
Tk_var_7 = Otsu(I_img,img_var_7)
Tk_var_9 = Otsu(I_img,img_var_9)

% Performing Logic 'AND'. Update the pixel values in R,G,B channels for
% second iteration
for i = 1:1:img_size(1)
    for j = 1:1:img_size(2)
```

```
        if (img_var_3(i,j) >= Tk_var_3)
            I_img_var_3(i,j) = 1;
        else
            img_var_3(i,j) = 0;
            img_var_5(i,j) = 0;
            img_var_7(i,j) = 0;
            img_var_9(i,j) = 0;
        end

        if (img_var_5(i,j) >= Tk_var_5)
            I_img_var_5(i,j) = 1;
        else
            img_var_3(i,j) = 0;
            img_var_5(i,j) = 0;
            img_var_7(i,j) = 0;
            img_var_9(i,j) = 0;
        end

        if (img_var_7(i,j) >= Tk_var_7)
            I_img_var_7(i,j) = 1;
        else
            img_var_3(i,j) = 0;
            img_var_5(i,j) = 0;
            img_var_7(i,j) = 0;
            img_var_9(i,j) = 0;
        end

        if (img_var_9(i,j) >= Tk_var_9)
            I_img_var_9(i,j) = 1;
        else
            img_var_3(i,j) = 0;
            img_var_5(i,j) = 0;
            img_var_7(i,j) = 0;
            img_var_9(i,j) = 0;
        end
    end
end

% Show the segmented images based on different var_N

figure
imshow(I_img_var_3*255);
figure
imshow(I_img_var_5*255);
figure
imshow(I_img_var_7*255);
figure
imshow(I_img_var_9*255);

I_img = I_img.*I_img_var_3.*I_img_var_5.*I_img_var_7.*I_img_var_9;
figure
imshow(I_img*255);

% Save the segmented images based on different var_N
```

```
save_name = ['I_img_var_3_1st_iteration_image_' image_name];
imwrite(I_img_var_3, save_name, 'jpeg')
save_name = ['I_img_var_5_1st_iteration_image_' image_name];
imwrite(I_img_var_5, save_name, 'jpeg')
save_name = ['I_img_var_7_1st_iteration_image_' image_name];
imwrite(I_img_var_7, save_name, 'jpeg')
save_name = ['I_img_var_9_1st_iteration_image_' image_name];
imwrite(I_img_var_9, save_name, 'jpeg')
save_name = ['I_img_1st_iteration_image_' image_name];
imwrite(I_img, save_name, 'jpeg')

% Extract and save the contour of 1st iteration
contour_img = contour_extract(I_img);
save_name = ['contour_img_1st_iteration_image_' image_name];
imwrite(contour_img, save_name, 'jpeg')

% Re-initilize the logic operator for second iteration
I_img_var_3 = zeros(img_size(1),img_size(2));
I_img_var_5 = zeros(img_size(1),img_size(2));
I_img_var_7 = zeros(img_size(1),img_size(2));
I_img_var_9 = zeros(img_size(1),img_size(2));

% Find the threshold using Otsu's algorithms of second iteration
Tk_var_3 = Otsu(I_img,img_var_3)
Tk_var_5 = Otsu(I_img,img_var_5)
Tk_var_7 = Otsu(I_img,img_var_7)
Tk_var_9 = Otsu(I_img,img_var_9)

% Performing Logic 'AND' of second iteration
for i = 1:1:img_size(1)
    for j = 1:1:img_size(2)
        if (img_var_3(i,j) >= Tk_var_3)
            I_img_var_3(i,j) = 1;
        else
        end

        if (img_var_5(i,j) >= Tk_var_5)
            I_img_var_5(i,j) = 1;
        else
        end

        if (img_var_7(i,j) >= Tk_var_7)
            I_img_var_7(i,j) = 1;
        else
        end

        if (img_var_9(i,j) >= Tk_var_9)
            I_img_var_9(i,j) = 1;
        else
        end
    end
end

figure
```

```
imshow(I_img_var_3*255);
figure
imshow(I_img_var_5*255);
figure
imshow(I_img_var_7*255);
figure
imshow(I_img_var_9*255);

% Show and save the segementaion result of second iteration
I_img = I_img.*I_img_var_3.*I_img_var_5.*I_img_var_7.*I_img_var_9;
figure
imshow(I_img*255);
save_name = ['I_img_var_3_2nd_iteration_image_' image_name];
imwrite(I_img_var_3, save_name, 'jpeg')
save_name = ['I_img_var_5_2nd_iteration_image_' image_name];
imwrite(I_img_var_5, save_name, 'jpeg')
save_name = ['I_img_var_7_2nd_iteration_image_' image_name];
imwrite(I_img_var_7, save_name, 'jpeg')
save_name = ['I_img_var_9_2nd_iteration_image_' image_name];
imwrite(I_img_var_9, save_name, 'jpeg')
save_name = ['I_img_2nd_iteration_image_' image_name];
imwrite(I_img, save_name, 'jpeg')

% Extract and save the contour of 2nd iteration
contour_img = contour_extract(I_img);
save_name = ['contour_img_2nd_iteration_image_' image_name];
imwrite(contour_img, save_name, 'jpeg')
end
```

8.2 Matlab Code: `Otsu.m`

```
function [ Tk ] = Otsu(I_img, img )
%% Otsu's algorithm to detect threshold value for image segmentation

% Disregard those points at which the indicator value is '0'
% (As our approach is iterative based thus this is important)
img_size = size(img);
imgmin = img;
for i = 1:1:img_size(1)
    for j = 1:1:img_size(2)
        if (img(i,j) == 0)
            imgmin(i,j) = 255;
            I_img(i,j) = 0;
        else
        end
    end
end

% Find the range of pixel values
img_min = min(min(imgmin));
img_max = max(max(img));

% Find the mu_T value
mut = sum(sum(img))/sum(sum(I_img));

% Find the threshold value
cnt = 1;
for k = img_min:1:img_max
    p(cnt) = sum(sum(img == k))/sum(sum(I_img));
    cnt = cnt + 1;
end

cnt = 1;
for k = img_min:1:img_max
    p_sum(cnt) = sum(p(1:cnt));

    m0(cnt) = 0;
    m1(cnt) = 0;

    cnt_i = 1;
    for i = img_min:1:k
        m0(cnt) = m0(cnt) + i*p(cnt_i);
        cnt_i = cnt_i + 1;
    end

    for i = k+1:1:img_max-1
        m1(cnt) = m1(cnt) + i*p(cnt_i);
        cnt_i = cnt_i + 1;
    end

    mu0(cnt) = m0(cnt)/p_sum(cnt);
```

```
    mu1(cnt) = (m1(cnt))/(1-p_sum(cnt));
    sigma_b(cnt) = (p_sum(cnt)*(1-p_sum(cnt)))*((mu0(cnt) - mu1(cnt))^2);

    cnt = cnt + 1;
end

size_sigma = size(sigma_b);

% Return the threshold value
for k = 1:1:size_sigma(2)
    if (sigma_b(k) == max(sigma_b))
        Tk = img_min + k;
    else
    end
end


end
```

8.3 Matlab Code: contour_extract.m

```
function [contour_img ] = contour_extract( I_img )
%% Contour extraction based on 8 points neighbourhood
img_size = size(I_img);

% Initialize the contour images
contour_img = zeros(img_size(1),img_size(2));

% Find the countour based on 8-points neighbourhood
% Contour points must satisfy:
% 1. Indicator value at pixel location is non-zero
% 2. neighbourhood sum is larger than 3 to supress noise
% 3. Neighbourhood sum is smaller than 5 to detect edges
for i = 2:1:img_size(1)-1
    for j = 2:1:img_size(2)-1
        if (I_img(i,j) ~= 0)
        neighbour_sum = 0;
        neighbour_sum = I_img(i-1,j-1) + I_img(i-1,j) + I_img(i-1,j+1) + ...
            I_img(i,j-1) + I_img(i,j+1) + I_img(i+1,j-1) + I_img(i+1,j) + ...
            I_img(i+1,j+1);
        if (neighbour_sum >=3) && (neighbour_sum <=5)
            contour_img(i,j) = 1;
        else
        end
        else
        end
    end
end

% Show the contour image
figure
imshow(contour_img*255)


end
```