Assignment 4: Advanced Image Manipulations with OpenCV

Param Chokshi

Department of Computer Science and Engineering

University of South Florida, Tampa, Florida, USA

1. Introduction

The assignment focuses on performing basic image manipulation of gray scale and color images. The scope of this assignment is to perform manipulations like thresholding, binarization, color modification, smoothing, histogram stretching, RGB to HSI conversion, and edge detection on user defined region of interests (ROIs) within the given image. Here thresholding helps in enhancing or highlighting certain pixels falling in a given range. Binarization helps with segmentation of the image creating a foreground and background based on the threshold value. Color modification allows to adjust the RGB values for the image which helps in creating an image catered to the viewer's color perception. Uniform smoothing helps in reducing noise in the image. Histogram stretching helps to brighten an image. Edge detection is another technique for achieving segmentation and can be used to differentiate objects in the image. Histogram Equalizations is also an useful method for contrast enhancement which can also be used to improve edge detection results. The report will contain a detailed description of the algorithms and their implementation used to perform transformations to images, followed by an analysis of the results and conclusion.

2. Description of algorithms

This section introduces the algorithms used in the program. The program consists of sixteen major filters which are used to transform inputted images. They are (i) add intensity filter, (ii) binarize filter, (iii) scaling filter, (iv) thresholded add intensity filter, (v) double thresholding filter, (vi) color modification filter, (vii) 2D smoothing, (viii) incremental 1D smoothing, (ix) histogram stretching, (x) thresholded histogram stretching, (xi) histogram stretching on color images (RGB), (xii) histogram stretching on color images (HSI), (xiii) Edge detection using Sobel's operator, (xiv) Using OpenCV for edge detection with Canny and Sobel operators. (xv) Histogram Modification with OpenCV and (xvi) Histogram Equalization followed by edge detection using Sobel and Canny operators.

2.1 Add Intensity Filter

The add intensity filter increases the intensity of the image by a user entered value. This can also be used to decrease intensity of the image (if the user-defined value is negative). The algorithms achieve this by visiting each pixel in the ROI (within the image array) of a grayscale image and changing its value by adding the user defined value. Thus, if the size of ROI is NxN pixels, it performs the N^2 operations. Thus, it its average running time is $O(N^2)$.

2.2 Binarize Filter

The binarize filter binarizes the values of the pixels in the image to either the lowest or the highest value possible. The algorithm accepts a user defined threshold along with up to 3 ROIs, and all the pixels within the ROI (within the image array) with intensity over that value are set to the max-value (e.g. 255), while all the pixels with intensity less than the threshold are set to the minimum value (e.g. 0). Here also if the size of ROI is NxN pixels, it performs the N^2 operations. Thus, it its average running time is $O(N^2)$.

2.3 Scaling Filter

The scaling filter is used to change image size to either twice of its original size or to half of its original size. For making the size of the image twice the algorithm uses pixel replication, i.e. it takes a pixel from the original image and replicate its value to four pixels in the new image.

For decreasing the size to half, pixel averaging is used, i.e. averaging the value of four surrounding pixels from the original image and mapping it to a single pixel in the new image.

Here if the size of image is NxN the algorithm performs either $4N^2$ operations or $N^2/4$ operations. Thus, it its average running time is $O(N^2)$.

2.4 Thresholded Add Filter

This algorithm takes three user defined values, the threshold, two values (V1 and V2), up to 3 ROIs. If the intensity of the pixel is greater than the threshold value, the filter adds V1 to the intensity of that pixel. While if the intensity of the pixel is less that the threshold value, it decreases the value of intensity by subtracting V2. If the intensity of the pixel is equal to the threshold, the function leaves it as it is. Here also if the size of ROI is NxN pixels, it performs the N^2 operations. Thus, it its average running time is $O(N^2)$.

2.5 Double Thresholding Filter

This algorithm takes two user defined values as thresholds along with up to 3 ROIs. If the intensity of the pixel falls between two threshold values, pixel is made red in color. The algorithm visits each pixel in the ROI to achieve this manipulation. Here also if the size of ROI is NxN pixels, it performs the N^2 operations. Thus, it its average running time is $O(N^2)$.

2.6 Color Modification Filter

This algorithm takes three user defined values along with up to 3 ROIs. Each of the user defined values are used to change the intensities of the RGB channels of the image. The algorithm visits each pixel in the ROI to achieve this manipulation. Here also if the size of ROI is NxN pixels, it performs the N^2 operations. Thus, it its average running time is $O(N^2)$.

2.7 2D Smoothing Filter (Uniform)

This algorithm takes user defined window size along with up to 3 ROIs. The user defined window size (ws) is used to generate a mask of ws*ws (2D) pixels. This mask is used to perform uniform smoothing (averaging) over the given ROIs. The algorithm achieves smoothing by applying the mask of each of the pixels in the given ROI. Thus, if the size of ROI is NxN pixels, and the size of the mask is MxM pixels, it will perform N^2M^2 operations. Thus, its average running time is $O(N^2M^2)$.

2.8 Incremental 1D Smoothing Filter (Uniform)

This algorithm takes user defined window size (ws) along with up to 3 ROIs. Here an incremental approach is used to speed up smoothing algorithm. This algorithm uses a ws*1 mask and a 1*ws mask on each pixel to achieve uniform smoothing. Here the previous (previously processed pixel) mean is used to calculate the value of the next pixel. Thus, if the size of ROI is NxN pixels, it will perform n^2 operations. Thus, its average running time is $O(N^2)$.

2.9 Histogram Stretching

This algorithm takes two user defined values a and b along with up to 3 ROIs. Within the ROIs the pixel falling in the range [a,b] are stretched to [0-255]. Here contrast clipping is also implemented, i.e. pixel from [0,a] are mapped to 0, and pixels from [b,255] are mapped to 255. Since the minimum and the maximum value in the range is already provided by the user, the algorithm takes $O(N^2)$ time if the given ROI is NxN pixels as each pixel within the ROI is visited once iteratively and changed.

2.10 Thresholded Histogram Stretching

This algorithm takes a user defined threshold (t) along with up to 3 ROIs. This algorithm combines two filter, which are image thresholding and histogram stretching. Using the threshold, the image is divided into foreground and the background. Histogram stretching with contrast clipping is applied to each image separately. Here the range used in stretching is [minimum pixel intensity within a region, maximum pixel intensity within a region]. The process of thresholding, finding the max and the min pixel intensities, and stretching all take $O(N^2)$ runtime as they all visit each pixel iteratively within a NxN ROI. Thus, the runtime of the algorithm is also $O(N^2)$.

2.11 Histogram Stretching on Color Images (RGB)

This algorithm takes three user defined values a, b, and RGB code along with up to 3 ROIs. This algorithm performs exactly as the histogram stretching algorithm. Here based on the user inputted RGB code, stretching is applied to either the red channel, the green channel, the blue channel, or all RGB channels. To apply stretching to all the RGB channels, each channel undergoes individual histogram stretching and then are combined to form the resulting image. Here based on the input, stretching is applied up to 3 times. Thus, the runtime is $O(N^2)$ for a NxN ROI.

2.12 Histogram Stretching on Color Images (HSI)

This algorithm takes two user defined values a, and b, along with up to 3 ROIs. The algorithm implements RGB to HSI conversion. Histogram stretching is performed on the I component of the image and the image is then converted back to RGB from HSI. Performing stretching on the I component yields a better result and the HSI color space is more accurate to the human perception of color. Here histogram stretching is also implemented on the SI components as well as all HSI components. Here is the ROI is NxN pixels big, the conversion from RGB to HSI takes $O(N^2)$ time, and the stretching also takes $O(N^2)$ time as both of these operations visits each pixel within the ROI iteratively. Thus, the runtime is $O(N^2)$ for a NxN ROI.

2.13 Edge Detection using Sobel's Operator

This algorithm takes three user defined values gradient threshold, direction threshold, and the window size for the Sobel Operator, along with up to 3 ROIs. The algorithm uses Sobel's mask to calculate gradient in both x and y direction. The magnitude of x and y components are displayed as an grayscale image with edges. Thresholding is used to make the edges stronger. For color images the edge detection is applied to the I channel after converting the image form RGB to HSI. Here if the ROI is NxN, all thresholding, and color scale conversions each take $O(N^2)$ while the convolution with the masks take $O(N^2M^2)$ time where M is the window size. Thus, the runtime is $O(N^2M^2)$ for a NxN ROI and MxM mask. Edge detection on color images takes more time due to the need for color scale conversion.

2.14 Using OpenCV for Edge Detection with Canny and Sobel's operator

OpenCV (library of computer vision functions) provides some real time optimized Computer Vision tools. Utilizing its functions to calculate edge detection increases the performance and accuracy of the written program. Here Canny and Sobel functions are used from OpenCV to output required edge images.

2.15 Histogram Modification with OpenCV

Histogram stretching and Histogram Equalization is applied on grayscale and color images using OpenCV. Applying Histogram Equalization on multiple channels of color images yield different results. The algorithm performs the modifications in user entered ROIs.

2.16 Histogram Equalization followed by edge detection using Sobel and Canny operators.

The algorithm applies Histogram Equalization prior to applying edge detection. It compares the result of edge detection with and without histogram equalization by producing a difference image of the both images.

3. Description of implementation

The whole program is developed in C++ language. The program includes an Image class which is used for reading and writing image files (PGM and PPM). It also provides basic functionalities like accessing pixel values, size, number of columns, and number of rows etc. Using the Image class a Utility class is made which consist of all the aforementioned filters/algorithms. Additionally, OpenCV library is used to utilize some high level optimized functions directly. Finally, there is file which consists of the main class which reads a parameter file from the user and creates image objects and perform image transformations based on the parameter file's content.

4. Description and Analysis of Results

This section is used to go over the image transformation achieved through the use of all the implemented algorithms.

4.1 Add Intensity Filter

Figure 1 shows the result of the add intensity filter. The algorithm iteratively visits all the pixels in the given ROIs and increases the intensity by the value given. Increasing the intensity of the image makes the image whiter (or bright) as the intensity value approaches 255.

ROI:

Start: (0,100), Size: 100*500, value: 150

Start: (100,0), Size: 100*100, value: 150

Start: (200,200), Size: 100*100, value: 150

Figure 1: (left) Original grayscale image, (right) transformed image after applying add intensity filter.

4.2 Binarize Filter

Figure 2 shows the results of the binarize filter. The algorithm iteratively visits all the pixels in the given ROIs and binarizes the values based on the given threshold. The binarization filter makes the image black and white leaning towards black or white depending on the threshold. It is used as a segmentation technique to divide then image into foreground and background.

ROI:

Start: (0,100), Size: 100*500, threshold: 150

Start: (100,0), Size: 100*100, threshold: 150

Start: (200,200), Size: 100*100, threshold: 150

Figure 2: (left) Original grayscale image, (right) transformed image after applying binarize filter.

4.3 Scale Filter

Figure 3 shows the results of the scale filter used to increase the size of the image by the ratio 2

Figure 3: (left) Original grayscale image, (right) transformed image after applying scale filter.

4.4 Thresholded Add Filter

Figure 4 shows the transformation achieved by using the thresholded add filter. Based on the threshold value, the filter enhances the image by making darker pixels darker and brighter pixels brighter. The algorithm iteratively visits all the pixels in the given ROIs and changes the value of a pixel if it falls over or under the threshold.

Start: (0,100), Size: 100*500, threshold: 100, v1: 70, v2: 30

Start: (100,0), Size: 100*100, threshold: 100, v1: 70, v2: 30

Start: (200,200), Size: 100*100, threshold: 100, v1: 70, v2: 30

Figure 4: (left) Original grayscale image, (right) transformed image after applying thresholded add filter.

4.5 Double Thresholding Filter

The double thresholding filter visits the pixels in the ROI interactively and if they fall between the given two threshold value, the pixel is made red. Here the Red channel is increased to its max value while blue and green channels are set to 0 or minimum. This filter helps in highlighting pixels falling in a certain range of intensities.

ROI:

Start: (0,100), Size: 100*500, threshold1: 100, threshold2: 200

Start: (100,0), Size: 100*100, threshold1: 50, threshold2: 100

Start: (200,200), Size: 100*100, threshold1: 100, threshold2: 200

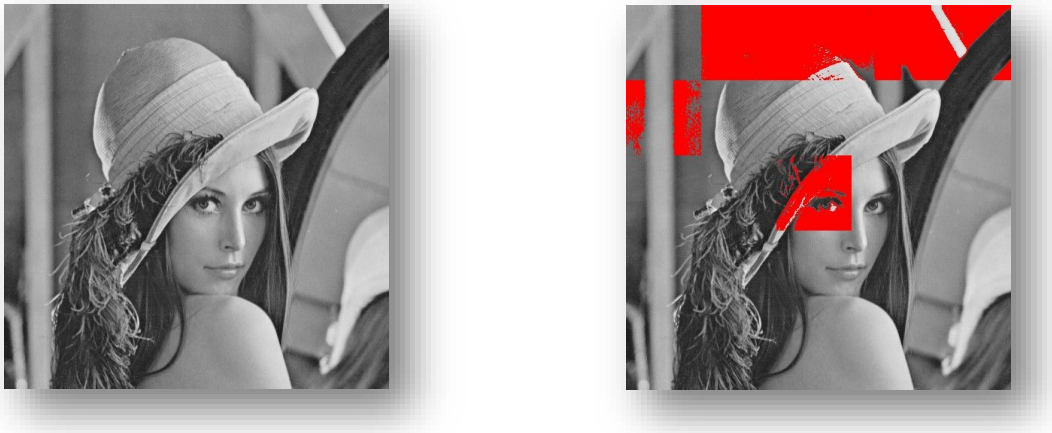


Figure 5: (left) Original grayscale image, (right) transformed image after applying double thresholding filter.

4.6 Color Modification Filter

This filter visits each pixel in the ROIs iteratively and change the value of RGB channels of the pixels by a user given amount.

ROI:

Start: (0,100), Size: 100*500, dR: 50, dG: -50, dB: 50

Start: (100,0), Size: 100*100, dR: 50, dG: 50, dB: 50

Start: (200,200), Size: 100*100, dR: 75, dG: 80, dB: -60

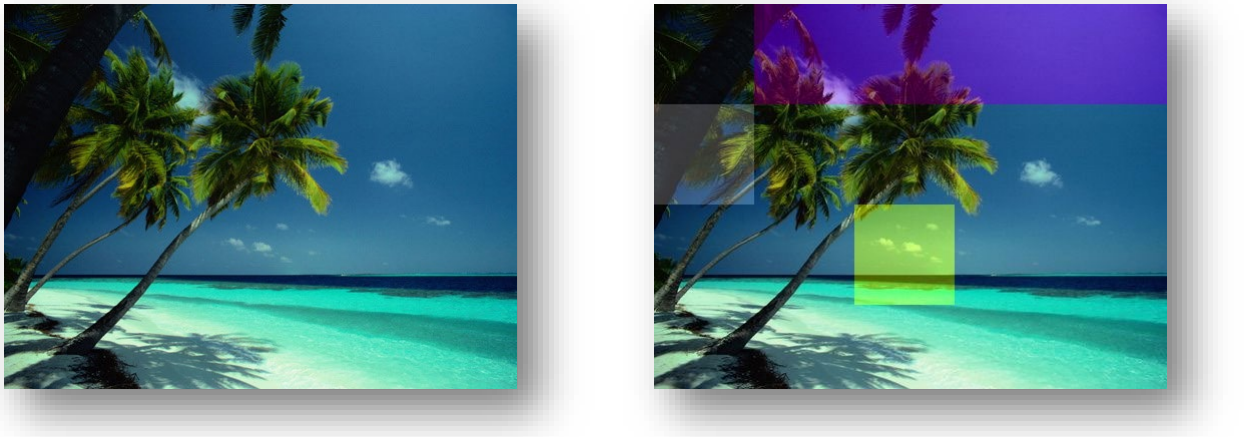


Figure 6: (left) Original color image, (right) transformed image after applying color modification filter.

4.7 2D Smoothing Filter (Uniform)

This filter is used to reduce noise from the image. Here uniform smoothing (averaging) is used to implement smoothing. In this algorithm a 2D mask is used to process each pixel within the ROI. The algorithm visits each pixel in the ROI iteratively and for each pixel, it averages the values present in the 2D mask with the pixel at its center. This averaging is also done by visiting each pixel in the mask iteratively. The size of this square 2D mask is given by the user. For edges and corner pixel where the whole mask is not in bounds of the image, average is calculated using the pixels which are in bounds of the image. This process is computationally expensive as it runs in $O(N^2M^2)$ time where the ROIs are NxN and the window size(ws) is MxM. Increasing the window size increases the smoothing (blurring) effect on the image.

ROI:

Start: (0,100), Size: 100*500, ws: 5/7/9

Start: (100,0), Size: 100*100, ws: 5/7/9

Start: (200,200), Size: 100*100, ws: 5/7/9

Figure 7: (top left) Original grayscale image, (top right) transformed image after applying 2D smoothing filter with ws = 5, (bottom left) transformed image after applying 2D smoothing filter with ws = 7, (bottom right) transformed image after applying 2D smoothing filter with ws = 9.

4.8 Incremental 1D Smoothing Filter (Uniform)

This algorithm is used to speed up the uniform smoothing process. Here 1D windows are used to calculate the new value for the pixel (in the ROI) once for every row. This previously calculated mean is stored and used to calculate the mean vale for next pixel in the row. Then the same process is repeated for columns. This can be represented by the following set of formulas:

For ws = 5

$I'(i,j) = (I(i-2.j) + I(i-1.j) + I(i.j) + I(i+1.j) + I(i+2.j))/5$

$I'(i+1,j) = I'(i,j) + I(i+3,j) - I(i-2,j)$

Since this calculation takes only O(1) time, the total complexity of the algorithm comes down to $O(N^2)$ where the size of ROIs is NxN. Here, in this implementation pixels near the edges are ignored as they don't have pixels in bounds of the ROI to use the first formula.

Increasing the window size increases the smoothing (blurring) effect on the image. The incremental implementation gives the same result as 2D smoothing theoretically. Here a very minute difference is seen between the two only when we increase the window size. This difference arises due to the fact that the algorithm ignores a few border rows and border columns of the ROI as there is no sufficient data available to calculate the new value using the formula.

ROI:

Start: (0,100), Size: 100*500, ws: 5/7/9

Start: (100,0), Size: 100*100, ws: 5/7/9

Start: (200,200), Size: 100*100, ws: 5/7/9

Figure 8: (top left) Original grayscale image, (top right) transformed image after applying 1D smoothing filter with ws = 5, (bottom left) transformed image after applying 1D smoothing filter with ws = 7, (bottom right) transformed image after applying 1D smoothing filter with ws = 9.

4.9 Histogram Stretching

Histogram stretching with contrast clipping is applied to improve the brightness of the image. Here each pixel within the ROI undergoes transformation through this formula:

$I'(i,j) = 0$ if $I(i,j) < a$

$I'(i,j) = (I(i,j) - a)(255/(b-a))$ if $a < I(i,j) < b$

$I'(i,j) = 255$ if $I(i,j) > b$

Here all grayscale values between a and b are mapped to [0-255]. This increases the contrast of the image, i.e. spreads out the gray scale values between a through b to (0,255). Figure 9 shows the pre and the post histogram stretching histograms. The post-histogram shows that the peak observed in pre-histogram was flattened.

ROI:

Start: (50,50), Size: 300*300, a: 100, b: 200

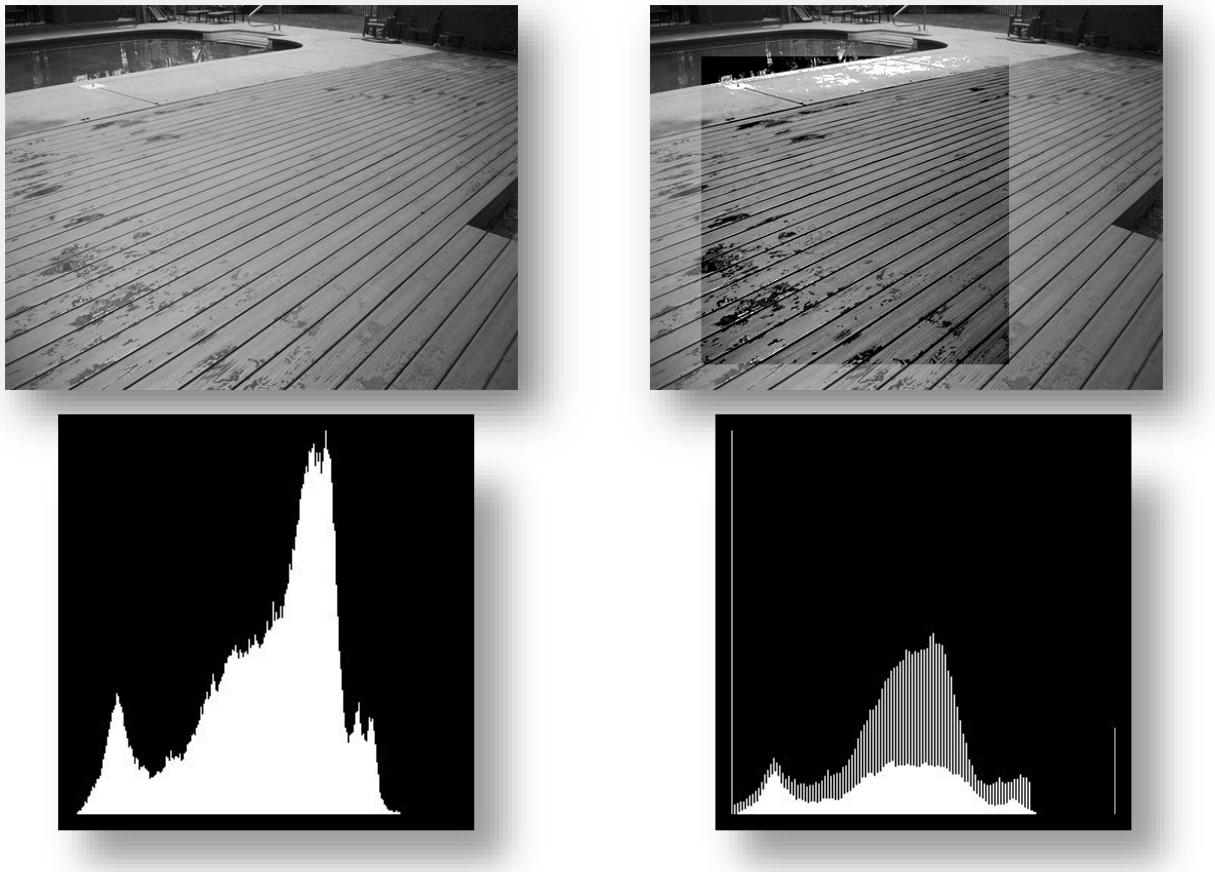


Figure 9: (top left) Original grayscale image, (top right) transformed image after applying histogram stretching, (bottom left) histogram before stretching, (bottom right) histogram after stretching.

4.10Thresholded Histogram Stretching

This algorithm combines two filters, histogram stretching and thresholding. Thresholding divides the image into two parts. Each of them undergoes histogram stretching and the resulting image is the combination of both of those images. For both the foreground and the background, the grayscale values are stretched from regional minimum value to maximum value to (0, 255). This process helps in separating grayscale values in to two groups over which histogram stretching with better values can be applied.

ROI:

Start: (0,0), Size: 387*250, t: 100

Figure 10: (left) Original image, (right) transformed image after applying thresholded histogram stretching.

4.11 Histogram Stretching on Color Images (RGB)

This algorithm applies histogram stretching with contrast clipping on each of the RGB channels and combines them as a final bright image. This method of brightness transformation in color images is not very accurate as it doesn't fit human perception very well. Also, with different values of the channels RGB, different colors are obtained by additive mixing of colors. Applying histogram stretching on individual channels changes the proportion of the individual colors changing the color inaccurately. One such instance can be observed in figure 11 (top-right).

ROI:

Start: (0,0), Size: 200*200, a: 0, b: 127

Figure 11: (top left) Original grayscale image, (top right) transformed image after applying RGB histogram stretching, (bottom left) transformed image after applying histogram stretching on R channel, (bottom middle) transformed image after applying histogram stretching on G channel, (bottom right) transformed image after applying histogram stretching on B channel.

4.12 Histogram Stretching on Color Images (HSI)

To achieve better results from histogram stretching on color images, HSI color coordinates can be used. In HSI, the I component represents the intensity value of the pixel, while the H and S contributes to the color of the image. Thus, this algorithm converts pixels from RGB to HSI, and applies histogram stretching to the I component. The final image is produced by converting pixels from HSI to RGB. This produces a more accurate brightness transformation. This method also works as it is closer to how humans perceive color. Figure 12 also shows the result of histogram stretching on SI components, and the HSI components. The resulting images form both of those transformations have different colors as the channels H & S are chromatic channels which contributes to color of the image.

ROI:

Start: (0,0), Size: 200*200, a: 0, b: 127

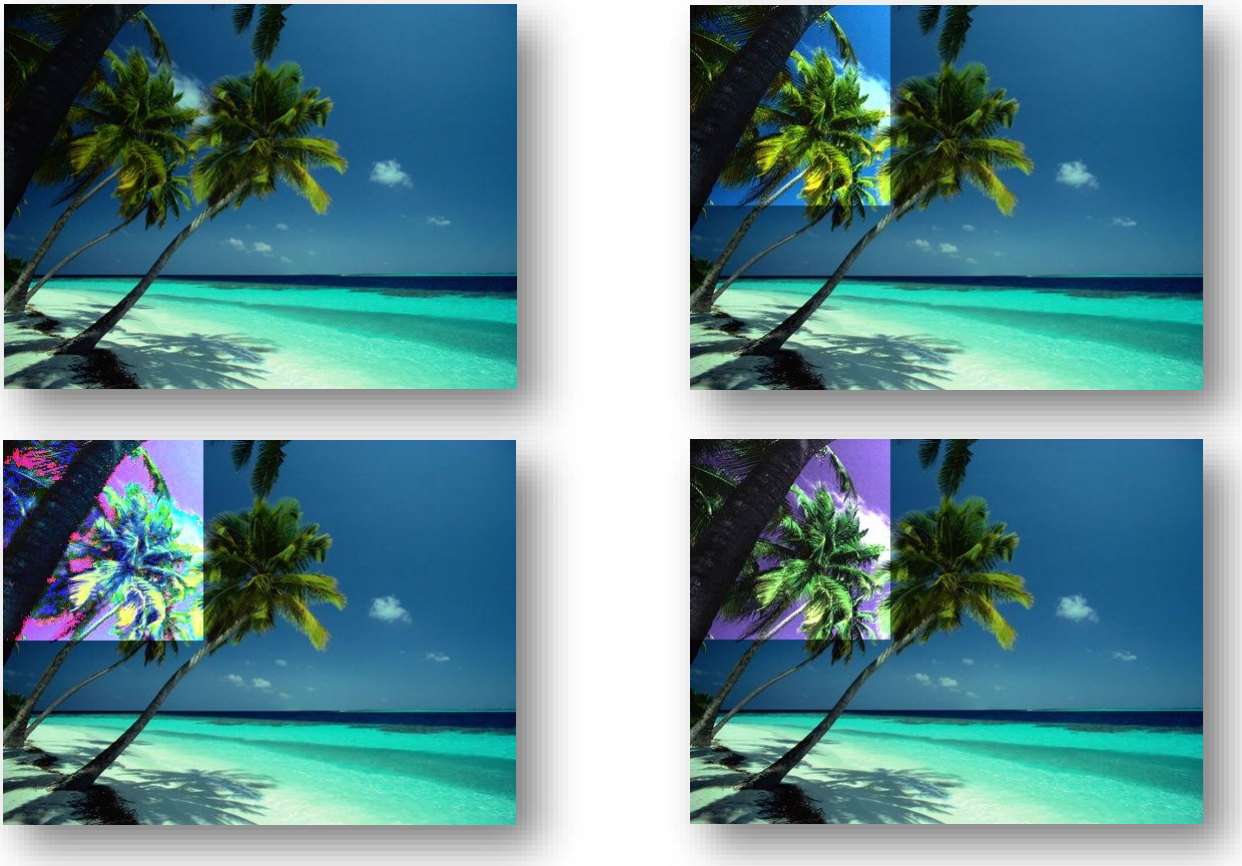


Figure 12: (top left) Original grayscale image, (top right) transformed image after applying histogram stretching on I , (bottom left) transformed image after applying histogram stretching on I & S, (bottom right) transformed image after applying histogram stretching on HSI.

4.13 Edge Detection using Sobel's Operator

The algorithm finds the gradient of the image, which is used to find pixels over which a rapid change in values is observed. Gradient with high enough values are thresholded to get better and brighter edges. The algorithm also saves edge direction, which can be used to segregate edges with the same direction. For color images the edge detection is performed on the I channel to achieve better and more accurate results.

ROI:

Start: (0,100), Size: 100*500, ws: 3, threshold: 30, angle: 45

Start: (100,0), Size: 100*100, ws: 3, threshold: 30, angle: 45

Start: (200,200), Size: 100*100, ws: 3, threshold: 30, angle: 45

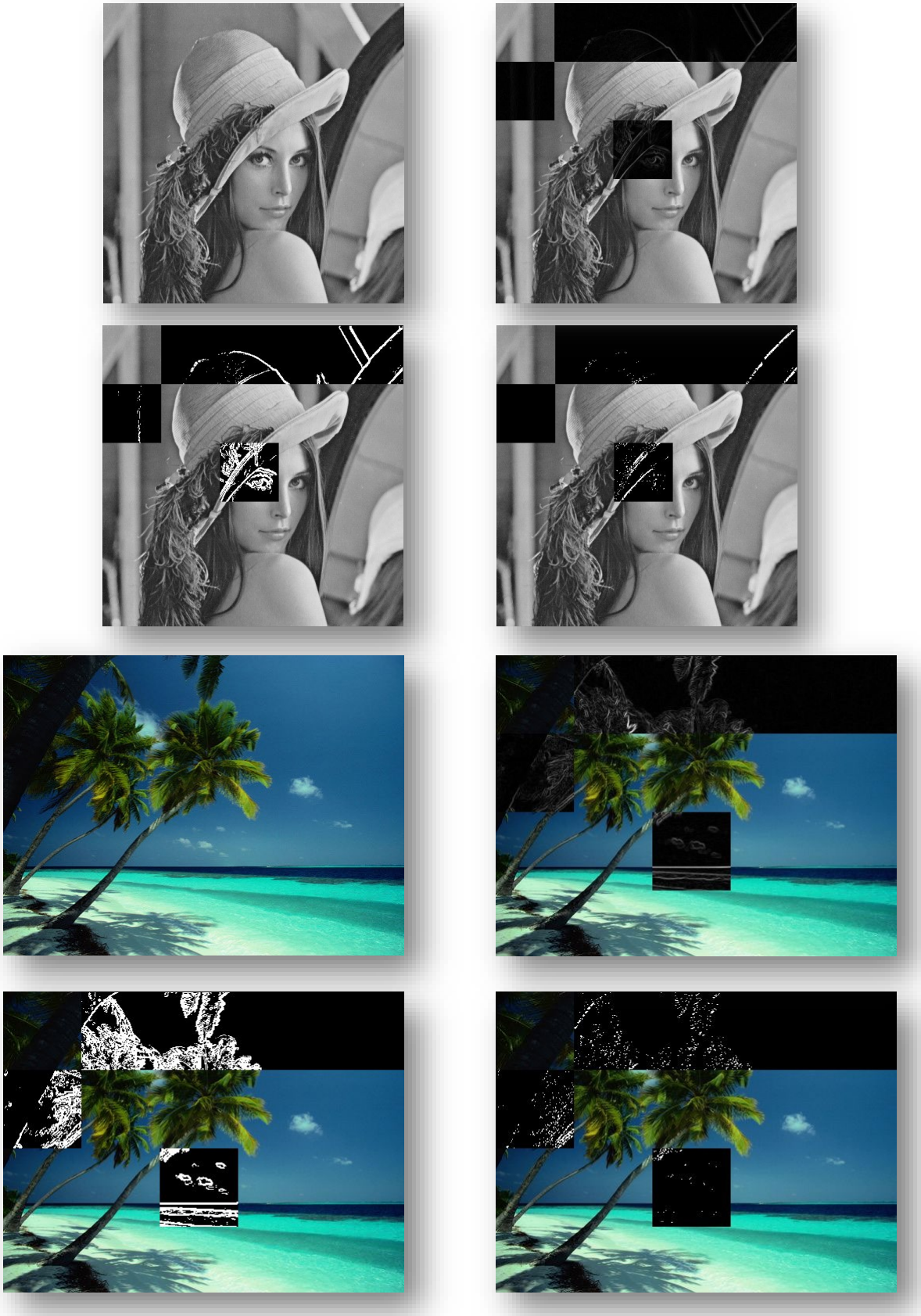


Figure 13: (first row left) Original grayscale image, (first row right) gradient amplitude image, (second row left) gradient image after binarization, (second row right) edges directed at 45 degrees, (third row left) Original image, (third row right) gradient amplitude image, (fourth row left) gradient image after binarization, (fourth row right) edges directed at 45 degrees.

4.14 Using OpenCV for Edge Detection with Canny and Sobel's operator

OpenCV functions are faster and more optimized to the functions implemented in the program. Color scale conversions are also easier. For color images the edge detection is applied on the V channel of the HSV color space. Here the key difference between edges calculated by Sobel and Canny functions arises due to non-maximal suppression. Non-maximal suppression produced thin defined edges in Canny unlike Sobel's operator. Canny function also does smoothing to reduce white noise before processing.

ROI:

Start (0, 300), Size: 480*420, ws: 3, t1: 30, t2: 45

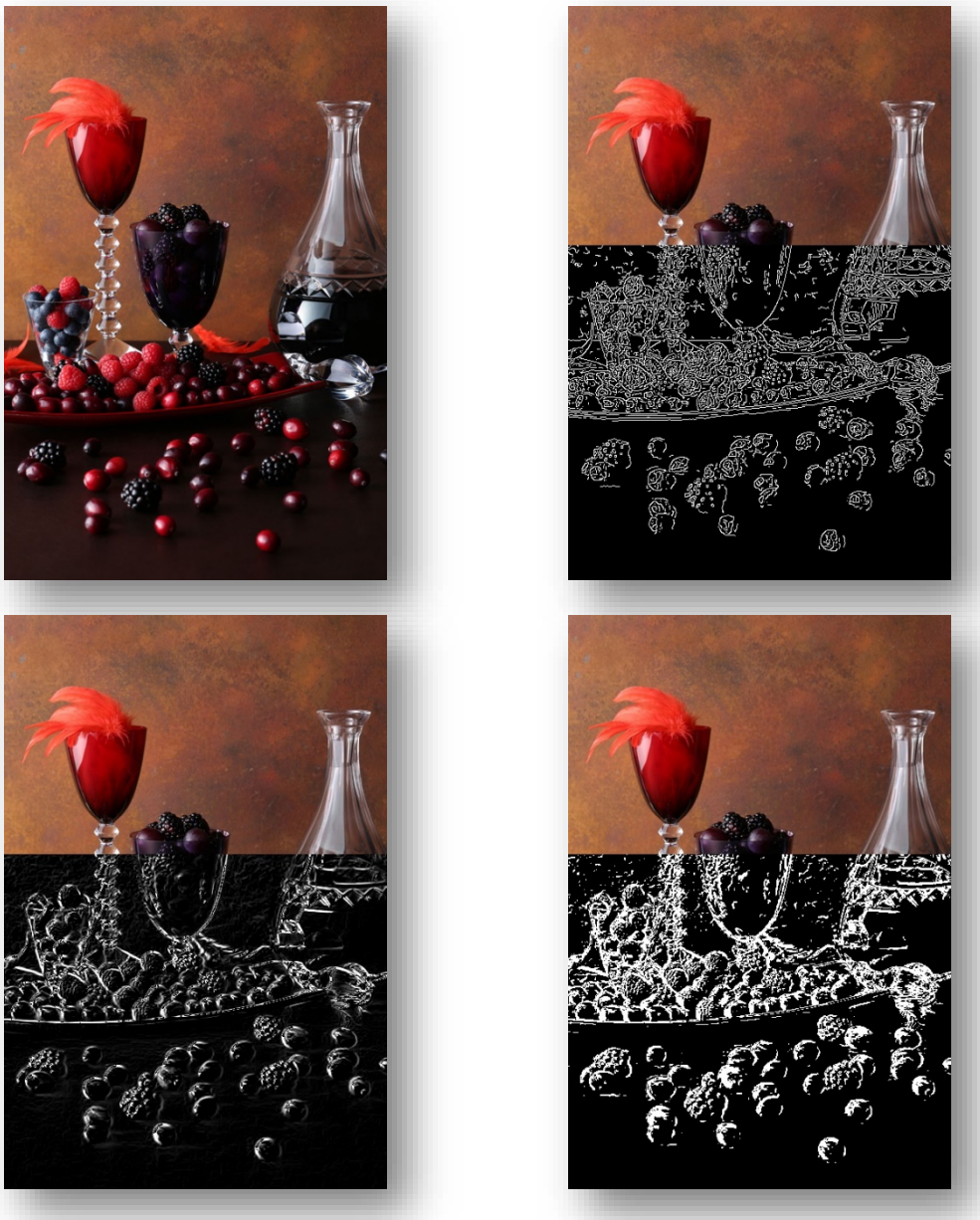


Figure 14: (first row left) original image, (first row right) edges produced by Canny operator, (second row left) gradient amplitude image produced by Sobel's operator, (second row right) gradient image after binarization.

4.15 Histogram Modification with OpenCV

Histogram stretching and histogram equalization are contrast enhancement methods used to make images brighter for viewing and analyzing purposes.

ROI:

Start (0, 0), Size: 300*150

Start (0, 200), Size: 300*150

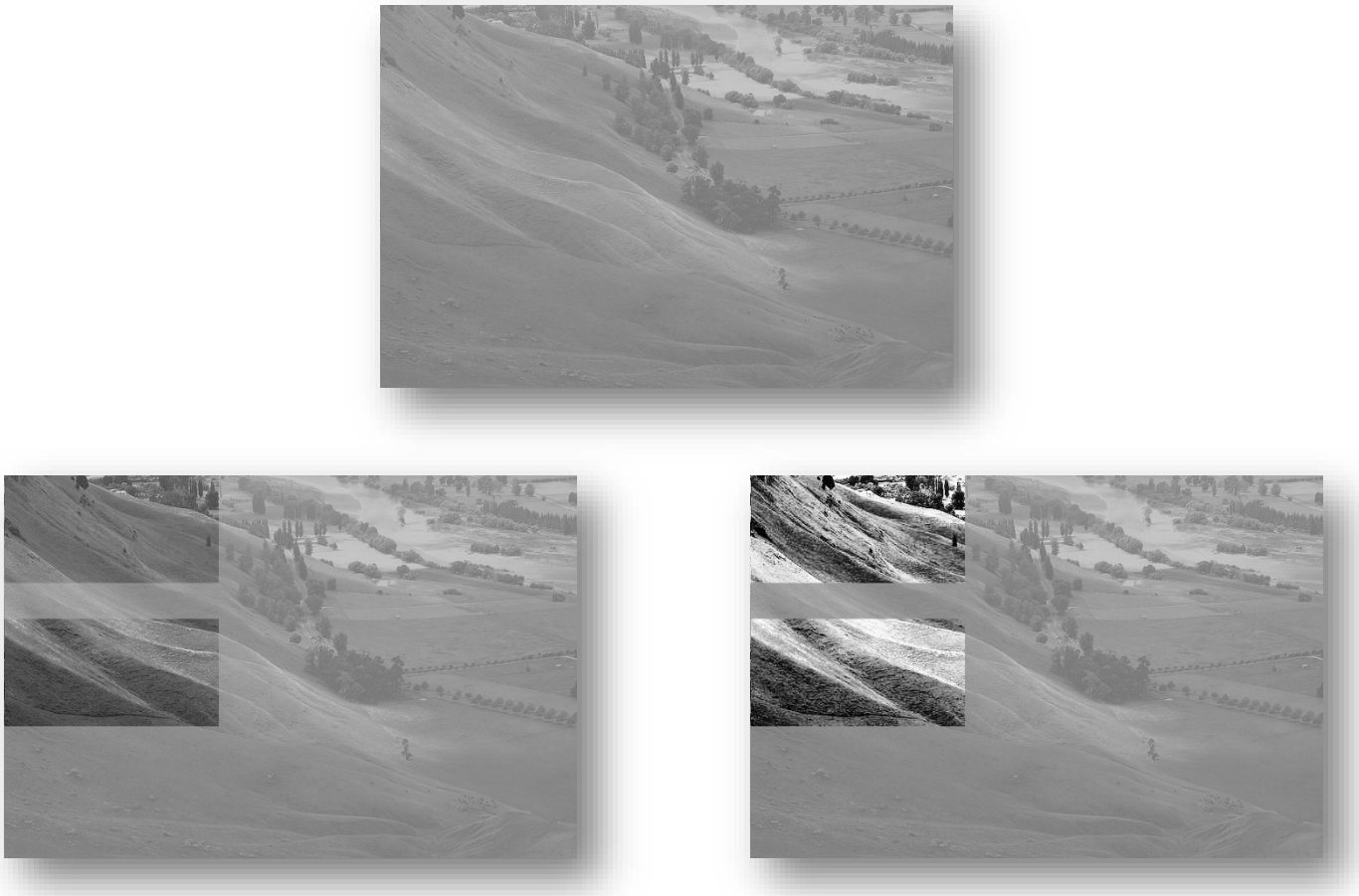
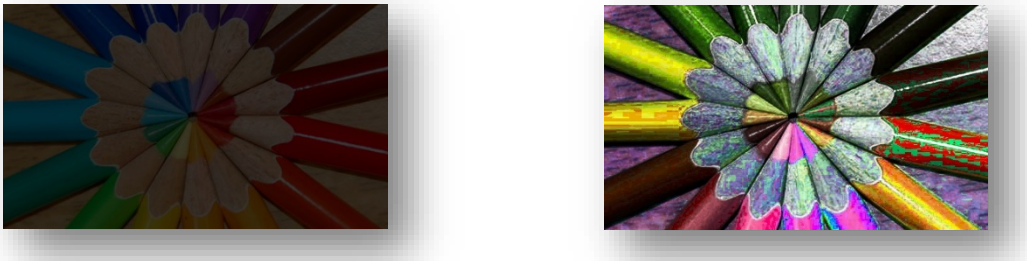


Figure 15: (top) Original Image, (bottom left) Image after applying Histogram Stretching, (bottom right) Image after applying Histogram Equalization.

For color images Histogram Equalization is applied after converting RGB images to HSV images. Equalization can be applied to any of the channels and the results are shown below. Generally applying on the V channel results into better contrast enhancements.



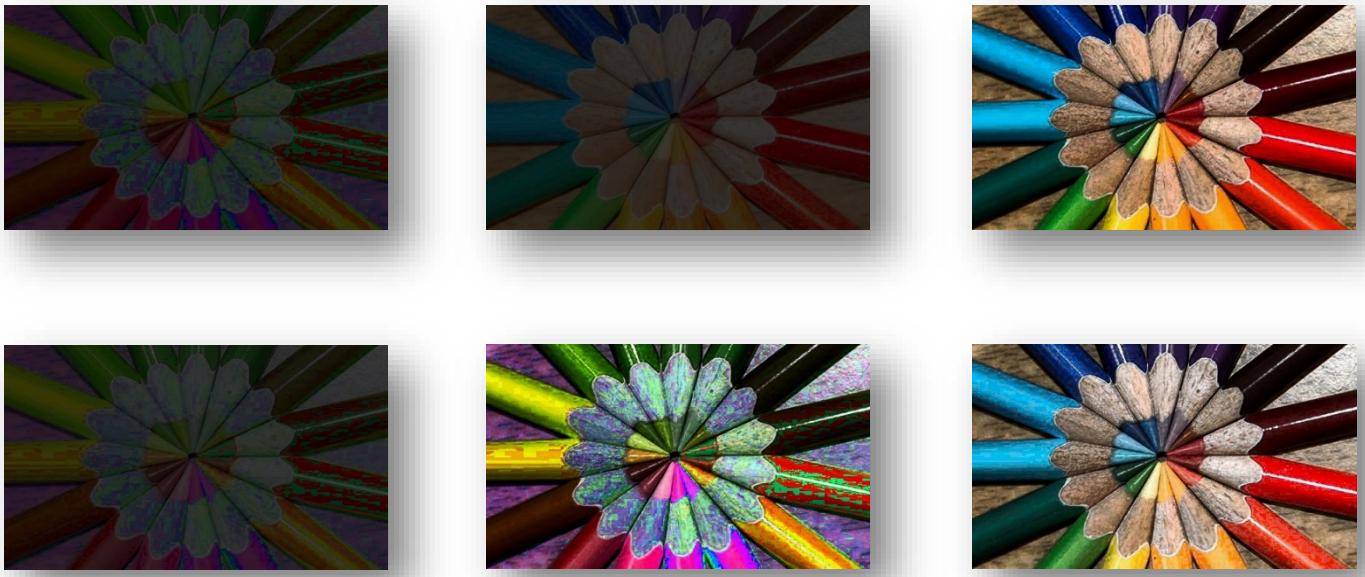


Figure 16: (first row left) Original Image, (first row right) Image after applying Histogram Equalization on HSV channels, (second row left) Image after applying Histogram Equalization on H channel, (second row middle) Image after applying Histogram Equalization on S channel, (second row right) Image after applying Histogram Equalization on V channel, (third row left) Image after applying Histogram Equalization on HS channels, (third row middle) Image after applying Histogram Equalization on HV channels, (third row right) Image after applying Histogram Equalization on SV channels

4.16 Histogram Equalization followed by edge detection using Sobel and Canny operators.

Combining Histogram Equalization with edge detection yields better edge detection. This is observed due to the fact that Equalization enhances the contrast thereby enhancing the gradient calculation of edges and as a result producing better results. The difference image shows the extra edges produced by using Equalization prior to edge detection.

(Sobel) ROI: Full image , ws: 3

Figure 17: (top left) Original Image, (top right) Image after applying Sobel Edge detection , (bottom left) Image after applying Histogram Equalization followed by Sobel Edge detection, (bottom right) Difference image .

(Canny**)**ROI: Full image, ws: 3, t1: 200, t2: 150

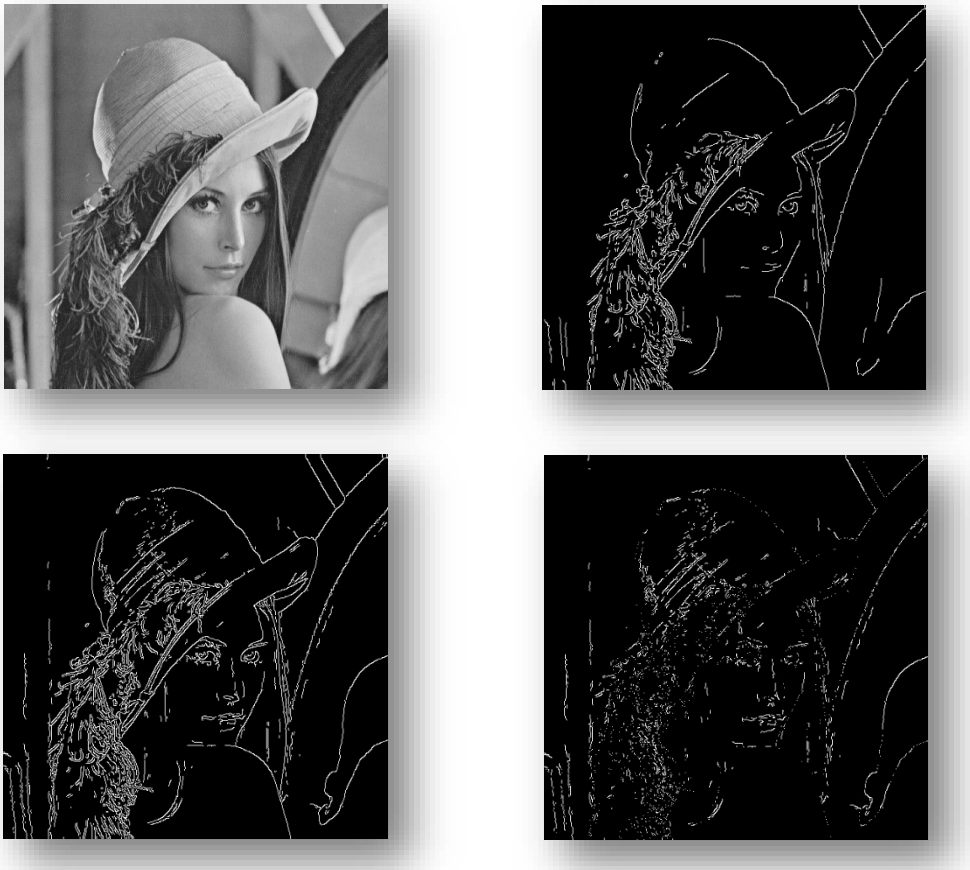


Figure 18: (top left) Original Image, (top right) Image after applying Canny Edge detection , (bottom left) Image after applying Histogram Equalization followed by Canny Edge detection, (bottom right) Difference image .

5. Conclusions

This assignment was aimed towards some basic and advanced image transformations for image processing. Region of interests (ROIs) were used within the image to achieve better processing by using specific sub-images. The assignment also focused on manipulating color channels of the pixels. Other integral implementation was the smoothing filters. Two implementations to achieve uniform smoothing were implemented. An important outcome was the speed up achieved by using 1D mask in an incremental way. This method is useful when the masks (smoothing filters) are separable. Histogram Stretching was implemented in different ways on gray scale and color images. Each implementation was compared to find better stretching method for brightness transformation. Two implementation of edge detection were also compared. Histogram Equalization was implemented to achieve image contrast enhancement. Histogram Equalization was also used prior to Edge Detection to improve the detection process. OpenCV functions were also used to achieve better performing and more accurate results.