**Assignment - 1 ID:104768826**

**1.) 1.1 Histogram Equalization:**

Histogram equalization is used to adjust contrast of the image. When we plot the intensity values of an image on to a histogram, they might be in a uneven order. When histogram equalization is applied these intensity values plotted on a histogram are adjusted to a given bins number or 64 bins by default.

**1.2 Results of binary format of the lena image:**

Threshold value 0.75

Threshold value 0.50

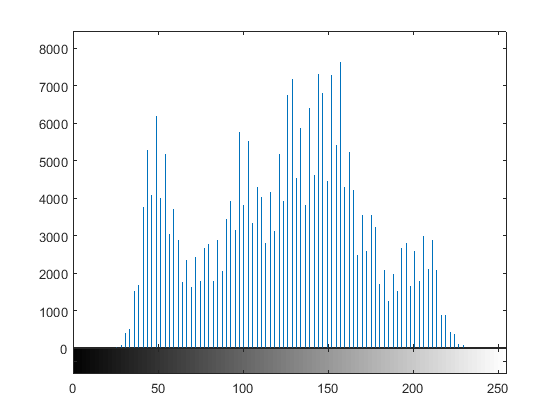


Threshold value 0.25



Threshold value 0.05(white image)



**1.3. Histogram of the lena image and the result of histogram equalization**



**After Histogram Equalization:**



**1.4 Codes:**

image=imread('C:\Users\Aditya\Desktop\Android\lena.tiff','tiff');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% thresholding

bin\_img1=im2bw(image,0.75);

bin\_img2=im2bw(image,0.50);

bin\_img3=im2bw(image,0.25);

bin\_img4=im2bw(image,0.05);

%%%%%%%%%%%% Histogram Equalization

grey\_image=rgb2gray(image);

imhist(grey\_image,100)

histeq(grey\_image,100)

**2.)**

**2.1. Background of noise filtering**

Noise is an unwanted intensity values in an image, noise in an outlier which degrades an image quality. Noise filtering is necessary to remove those unwanted pixels without effecting the original image quality. Filtering can be classified into two types:

1.Linear (ex: Mean and Gaussian, simple to design, works best for salt and pepper noise)

2.Non-linear(ex :Median filter, more difficult to design, can preserve edges)

**2.2. Results of the filtered image obtained with median and mean filters**



**Analysis:**

Median filter can only be applied on a 2d gray image, while mean filter can be applied on 3d RGB image.We can see that median filtered images are sharper with fine lines, images filtered by mean filter appear smoother than original image. Median filter is passive for any outliers(noise).Also by increasing window size the smoothness increases ,which is not desirable. Also it involves more complexity.

**2.3 Codes:**

%%%%%%%% median filter with 3 and 5 windows sizes

medfilt3=medfilt2(grey\_image,[3,3]);

medfilt5=medfilt2(grey\_image,[5,5]);

%%%%%%%% mean filter with 3 and 5 windows sizes

filt3=fspecial('average',[3 3]);

meanfilt3=imfilter(image,filt3);

filt5=fspecial('average',[5 5]);

meanfilt5=imfilter(image,filt5);

%%%%%% plotting images with titles

subplot(2,2,1),imshow(medfilt3),title('Median filter, Window 3')

subplot(2,2,2),imshow(medfilt5),title('Median filter, Window 5')

subplot(2,2,3),imshow(meanfilt3),title('Mean filter, Window 3')

subplot(2,2,4),imshow(meanfilt5),title('Mean filter, Window 5')

**3.)**

**3.1. Background of edge detectors: Sobel, Prewitt, Laplacian of Gaussian and Canny edge detector:**

Sobel:

* It is an Gradient based edge detection algorithm, which detects the maximum and minimum in the First derivative of the Image.
* It uses a 3\*3 convolution mask, by which we calculate the gradient of each pixel position in the image. Sobel masks for x-direction : {[-1 -2 -1],[0 0 0],[1 2 1]} , y-direction :{[1 2 2],[1 1 0],[2 4 1]}

Prewitt:

* Gradient based.
* Prewitt masks for x-direction: {[-1 0 1],[ -1 0 1],[ -1 0 1]} , y-direction: {[1 1 1],[0 0 0],[-1 -1 -1]}
* Edges are calculated as a difference between corresponding pixel values.

Laplacian of Gaussian:

* Laplace operator can be used to detect edges in an image, which comes along with noise. To avoid the image noise, we first implement Gaussian filter with some kernel width and Standard deviation.

Canny:

* First the we apply Gaussian filter to reduce noise from the image.
* Then take a 2D first derivative of the image, which generates an image highlighting the edges.
* Now the algorithm traces along the edges and sets the pixels to zero which are not on the top of the ridge.

**3.2. Quantitative and qualitative analysis of Canny detector with varying parameters:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Threshold** | **Kernel Size** | **RMSE** | **Figure Number** |
| 0.9 | 1 | 41.611 | 1 |
| 0.5 | 1 | 35.835 | 2 |
| 0.01 | 1 | 109.677 | 3 |
| 0.09 | 1 | 29.574 | 4 |
| 0.09 | 2 | 24.968 | 5 |
| 0.07 | 3 | 49.085 | 6 |

Images:





2.)



3.)



4.)



5.)



6.)



Analysis:

* More Threshold values corresponds to less number of edges.
* More kernel size results in over smoothing of the image.
* At a constant kernel size, lowest threshold values give highest RMSE values.
* Varying kernel size results in no particular trend in RMSE values. As the patch size doubles with increase in a single value.
* Lowest RMSE is obtained at threshold at 0.09 and kernel 2.

**3.3. Results of applying Sobel, Prewitt, Laplacian of Gaussian detector:**

* Threshold and kernel size are selected by comparing corresponding RMSE values.
* For Laplace of Gaussian filter kernel(hsize) and threshold(sigma) are selected such that the sigma value is maximum (1) and max mask size [7\*7]. RMSE values doesn’t change below 3\*3 mask size and 0.63 sigma.
* For Sobel and Prewitt, optimal RMSE is obtained at thresholds 0.08 and 0.09 respectively.

Optimal Images:

Sobel[threshold=0.09]

Prewitt[threshold=0.08]

Laplace of Gaussian[mask=[7 7],sigma=1]



Advantages of Canny detector:

* It has Gaussian filter to remove any noise from the image.
* Enhances a signal based on noise ratio, also known as Non-maximal suppression.
* Algorithm has thresholding method which improves detection of edges.
* Images generated by Canny detector gives smooth images, perfect for circles and homogenous objects.

|  |  |
| --- | --- |
| **Detector** | RMSE |
| Canny | 24.968 |
| Sobel | 37.892 |
| Prewitt | 37.531 |
| Laplace of Gaussian | 93.224 |

Canny detector has the lowest RMSE value.

**3.4 Codes:**

%%%%%%%%%%% Canny Edge Detector %%%%%%%%%%%

image=imread('C:\Users\Aditya\Desktop\Android\lena.tiff','tiff');

ref\_image=imread('C:\Users\Aditya\Desktop\Android\lena\_edge.tif','tiff');

grey\_image=rgb2gray(image);

edge\_image=edge(grey\_image,'Canny',0.09,2);

mse=0;

for i=1:512

for j=1:512

mse=mse+sqrt(1/262144\*(abs(edge\_image(i,j)-ref\_image(i,j))));

end

end

%%%%%%%%%%%%% Sobel

image=imread('C:\Users\Aditya\Desktop\Android\lena.tiff','tiff');

ref\_image=imread('C:\Users\Aditya\Desktop\Android\lena\_edge.tif','tiff');

grey\_image=rgb2gray(image);

edge\_image\_sobel=edge(grey\_image,'Sobel',0.08);

mse\_sobel=0;

for i=1:512

for j=1:512

mse\_sobel=mse\_sobel+sqrt(1/262144\*abs((edge\_image\_sobel(i,j)-ref\_image(i,j))));

end

end

%%%%%%%%%%%%%% Prewitt

image=imread('C:\Users\Aditya\Desktop\Android\lena.tiff','tiff');

ref\_image=imread('C:\Users\Aditya\Desktop\Android\lena\_edge.tif','tiff');

grey\_image=rgb2gray(image);

edge\_image\_prewitt=edge(grey\_image,'Prewitt',0.08);

mse\_prewitt=0;

for i=1:512

for j=1:512

mse\_prewitt=mse\_prewitt+sqrt(1/262144\*abs((edge\_image\_prewitt(i,j)-ref\_image(i,j))));

end

end

%%%%%%%%%%%%%% Laplacian of Gaussian

image=imread('C:\Users\Aditya\Desktop\Android\lena.tiff','tiff');

ref\_image=imread('C:\Users\Aditya\Desktop\Android\lena\_edge.tif','tiff');

grey\_image=rgb2gray(image);

h=fspecial('log',[7 7],0.63);

edge\_log=imfilter(ref\_image,h);

mse\_log=0;

for i=1:512

for j=1:512

mse\_log=mse\_log+sqrt(1/262144\*abs((edge\_log(i,j)-ref\_image(i,j))));

end

end

subplot(2,2,1),imshow(edge\_image\_sobel),title('Sobel[0.09]');

subplot(2,2,2),imshow(edge\_image\_prewitt),title('Prewitt[0.09]');

subplot(2,2,3),imshow(edge\_log),title('Laplace of Gaussian[sigma=0.63,mask=[7 7]]');