EE569 Homework #1

Xiang Gao
9216-3489-10
xianggao@usc.edu

September 22, 2013

1 Abstract and Motivation

This homework contains 3 problems. Problem 1 is a pixel manipulation problem. Part (a) is a simple example of the blue screen technique which is widely used in the film industry. We will remove the blue background in part (a) and doing identification and extraction tasks in part (b) and (c). Part (d) is a simple example of the image inpainting technique. We should remove the undesired "superman" with several shots to get a clean image and later put that "superman" into another image. Problem 2 is about image enhancement. We compared the results of full-range linear scaling method to those of cumulative histogram equalization method. In part (b) we applied two kinds of histogram equalization algorithms separately. We transformed the histogram of an image to a Gaussian distribution function in part (c) and compared the effects of parameters. Problem 3 is about noise removal. In part (a) we use simple median and low-pass filters to denoise images with "pepper and salt" noise as well as gaussian noise. In part (b) I learned bilateral filter in which we use not only spatial but also graylevel information to help denoising. Part (c) I implement "non local mean" method which takes neighborhood similarity into account. I did some experiment with BM3D method in part (d) with matlab code provided by the author. We can see a significant improvement to those previous methods.

2 Approach and Procedures

2.1 Problem 1: Image Manipulation

2.1.1 Part (a): Blue Screen Technique

Since the background color is not a single color, we should find out the range of each RGB channel of it. After exploring the histogram of this image and considering that most pixels of this image is background, I set the background color range to be: R[0,30], G[0,50], B[200,255]. Scan the image pixel by pixel. If all its RGB values fall into that range, I know that this is a background pixel and set it to 'white'. Otherwise copy pixel from input image to output image. After that, we get the clean forground image without blue screen.

2.1.2 Part (b): Identifying Geometrical Objects

To identify objects from this image, I searched from the center of the image to the right. Since I had already removed the blue background, every time I reached a pixel that is not white, and the same time if the previous pixel is white, I knew that I reached an object. Since there are only two types of objects, to identify the shape of objects, I simply have to determine if it's a square which is much easier. To determine whether a pixel I have reached belongs to a square, I traverse from the pixel horizontally as well as vertically. If we reached a pixel that is white, we knew that this is not a square and then must be a circle. I set global variables to store the distance between the object and image center as well as the shape of the object, which will be used in part (c).

2.1.3 Part (c): Extracting Geometrical Objects

After part (b), the shape and the first pixel of each object are stored in an array respectively. This part starts with an input from user that tells the program which object to be extracted. Then begin searching from the first pixel in that object. I implemented an algorithm that searchs 8 pixels of a certain pixel's neighborhood. To begin with, I push the first pixel of the selected object into the stack. During every recursion, I pop a pixel from the stack and begin searching its neighborhood. I push neighbors that are not white and not been marked as used into the stack and mark it as used. Repeat doing this until the stack is empty, then we can extract all pixels from the selected object.

2.1.4 Part (d): Smart Camera - Moveing Objects Removal

In this task we have to remove a pattern from a serious of pictures. I assumed that the two images are different with each other only around the disturbing "superman". So when I take out 2 images, for example #1 and #5, the similar part of them will be in the clearified image. The algorithm I have employed is to take pairs of images iteratively, i.e (#1,#2), (#1,#3),... ,(#1,#8), (#2,#3)..., find the same pixels between them, and fill it into a clearified image. There are $C_8^2 = 28$ pairs of images. After all these iterations, the "Whitehouse" will be restored.

As for the second part of this task, I have to extract the "superman" image first. This will use the clearified image from the above process and a random image that contains the whole "superman". To make sure that the image selected contains the whole "superman", I compared each shots with the restored image and choose the image that has biggest different area with restored image. After extracting the "superman", I put it in the target image with some proper offsets of x, y coordinates.

2.2 Problem 2

2.2.1 Part (a): Histogram Equalization for Grayscale Images

In this task, we used 2 methods to enhance grayscale images.

(1)In full-range scaling method, we did a affine transform of the histogram to extend the original one to the full-range [0-255], according to the transfer function:

$$T(x) = 255 * \frac{x - V_{min}}{V_{max} - V_{min}}$$

where V_{min} is the lower bound of histogram of the original image, V_{max} is the upper bound and x is the gray level.

(2)In cumulative histogram-equalization method, we did a transformation such that the resulting image is with a uniform distribution. The transfer funciton is as follows:

$$T(x) = \frac{255}{N} * \sum_{k=0}^{x} H(k)$$

where x is the gray level, $H(k)$ is the histogram value at k and N is the total number of pixels.

2.2.2 Part (b): Histogram Equalization for Color Images

We implemented two histogram equalization techniques in this task:

(1) Cumulative histogram method. This method is used in part (a) and the transfer function is:

$$T(x) = \frac{255}{N} * \sum_{k=0}^{x} H(k)$$

where x is the gray level, $H(k)$ is the histogram value at k and N is the total number of pixels.

(2) Equalized histogram method. That is we divide the pixels uniformly into 256 bins. This makes the resulting histogram nearly strictly uniform. A harder part is how to determine which pixel should transfered to which color because pixels with the same graylevel in the input side may be transfered to different graylevel in the output side. The general idea of my solution is as follows: (a) Get the cumulative number of each graylevel. (b) Construct 256 bins, each with a capacity of N/256, where N is the total number of pixels in the image. (c) For each pixel, calculate the position of this pixel using the cumulative value and the index of position in its graylevel. (d) Calculate the bin it should fall into. If the bin is not full, change this pixel's value to the graylevel of this bin. If that bin is full, check the next bin with greater graylevel. (d) Continue doing this for all pixels in the image.

We apply these two methods respectively to each RGB channel and combine three channels into the output image.

2.2.3 Part (c): Histogram Transform

Histogram transform means we transform a histogram to a new distribution. In histogram equalization, we transform the histogram to a uniform distribution. In this task, we will transform it to a Gaussian distribution. Let's now derive the transfer funtion:

$$\begin{cases} y = T(x) \\ f(x) * dx = g(y) * dy \\ g(y) = \frac{1}{\sqrt{2\pi}\sigma} exp(-\frac{(y-\mu)^2)}{2\sigma^2}) \end{cases}$$

where $f(x)$, $g(y)$ is pdfs of x, y respectively. $T(x)$ is the transfer function.

$$\int_0^x f(t)dt = \int_0^x T'(t)\frac{1}{\sqrt{2\pi}\sigma} exp(-\frac{(T(t)-\mu)^2}{2\sigma^2})d(t)$$

$$\Rightarrow \int_0^x f(t)dt = \int_0^{T(x)} \frac{1}{\sqrt{2\pi}\sigma} exp(-\frac{(y-\mu)^2}{2\sigma^2})dy$$

Since $x = 0, 1, 2...255$, we change the equation into discrete form:

$$\sum_{t=0}^{x} \frac{H(t)}{N} = \sum_{y=0}^{T(x)} g(y)$$

where $H(t)$ is the histogram at t, N is total number of pixels, $g(y)$ is pdf of the target Gaussian distribution at $y = 0, 1, ...255$ So, to find an approximation of $T(x)$, we compare the right side for each $T(x) = 0, 1, 2...255$, which can be calculated in advance, with the left side of the equation and find the most accurate solution.

2.3 Problem 3

2.3.1 Part (a): Mix Noise in Color Image

There are color dots which represent impluse (pepper and salt) noise as well as random (maybe Gaussian) noise in the image.

According to the histograms of 3 channels, I assume there are two kinds of noise in all channels. I will implement "Median Filter" to remove impluse noise and "Low-pass Filter" to remove Gaussian noise in this task and apply filters to each channel respectively.

2.3.2 Part (b): Bilateral Filtering

Bilateral Filtering is a denoising method that uses both domain information and range informantion. This gives the method the advantage that it preserves edges. In this method, we have to use two weight functions (one for domain distance and one for graylevel distance) to determine the weight of each neighbor pixel in averaging within a window.

2.3.3 Part (c): Non-Local Means (NLM) Filtering

From above section, we know that bilateral filter has a good performance in preserving edges. However it just considers spatial and graylevel difference when averaging pixels. In NLM filter, we also take context into account. Pixels with a similar context will have a bigger weight when averaging. So in NLM filtering method, we are looking for those pixels that have similar neighborhood with the pixel we are denoising and the more similar, the greater weight it will be given.

2.3.4 Part (d): Block Matching 3-D (BM3D) Transform Filter

BM3D method use 3D transformation to denoise image. First we should find similar blocks in image, which we may use a similar method with NLM. Then we put all these similar blocks togather to form a "block sequence" which can be viewed as a 3-D array. We do 3-D transform to this 3-D array and use some denoising method in transform domain. Get the denoised iamge through inverse transformation.

3 Experimental Results and Discuss

3.1 Problem 1: Image Manipulation

3.1.1 Part (a): Blue Screen Technique

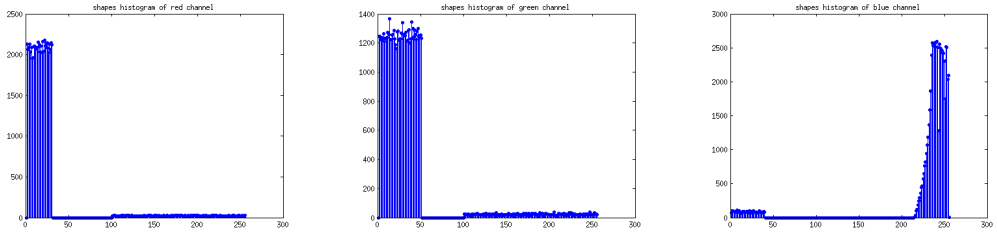

Figure 1: Histograms of RGB channels

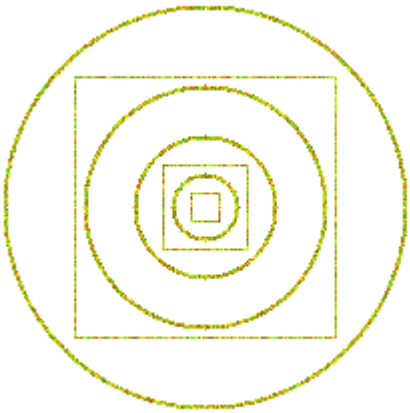

Figure 2: Part A result

We can easily tell the range of RGB from the above histograms in this particular case because there are obvious gaps between foreground and background in these three histograms. The range of bluescreen in this case is selected as R[0,30], G[0,50], B[200,255].

3.1.2 Part (b): Identifying Geometrical Objects

```
Part B begins:
The total number of circles is: 4 , and their respective radius is: 15 34 59 100

The total number of squares is: 3 , and their respective side length is: 14 42 1
30
The total number of geometrical objects is: 7
Part B ends.
```

Figure 3: Part B results

Since objects have 'thickness', the radius or side length I have used is the inner edge of the object.

3.1.3 Part (c): Extracting Geometrical Objects

```
Part C begins:
Please selete an object to print[1-7:S C S C C S C ]: 7
Part C ends.
Part C begins:
Please selete an object to print[1-7:S C S C C S C ]: 3
Part C ends.
```

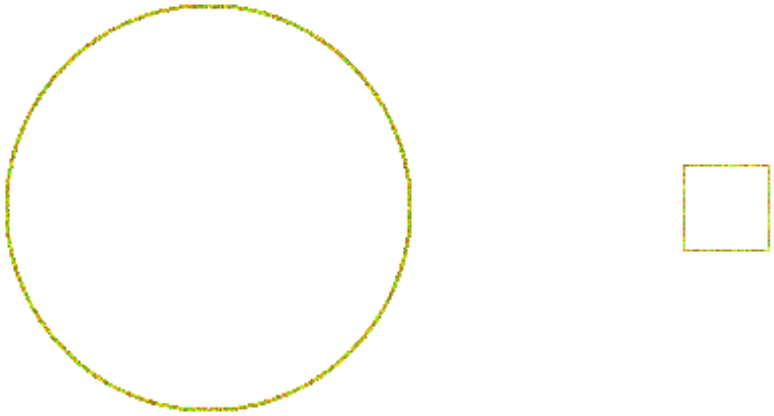

Figure 4: Part C results

We can see from the vector of shapes "([1-7:S C S C C S C])" which is generated in part (b), the largest circle is the 7th object and the second largest square is the 3rd object.

In this part, object can be extracted saperately only if they are not connected in the sense of 8 neighbors. The object printed using this algorithm is exactly the same as the original one.

3.1.4 Part (d): Smart Camera - Moveing Objects Removal

Figure 5: Result of Object Removal

We use 28 pairs of images to reconstruct the background. We assume that if a pixel is the same in both images, then it's a pixel of the background.

Figure 6: Result of 'Dog and Superman'

For the second part, I used a threshold to denoise in extracting the superman. Without using that threshold, there will be a lot of noise around the superman. As is shown below:

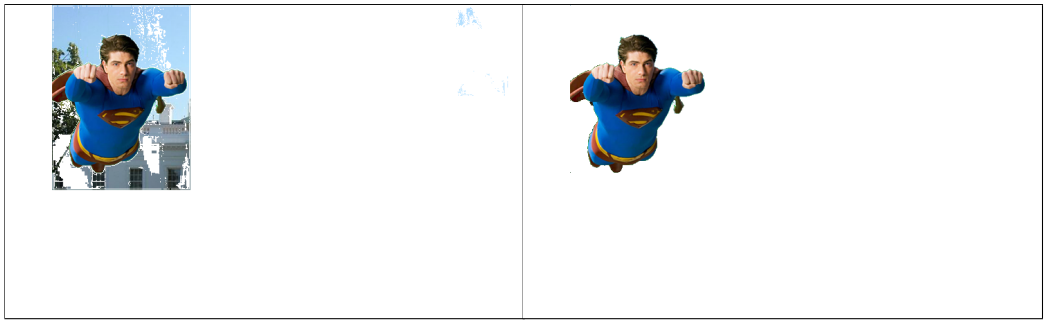

Figure 7: Superman without threshold and with threshold=20

3.2 Problem 2

3.2.1 Part (a): Histogram Equalization for Grayscale Images

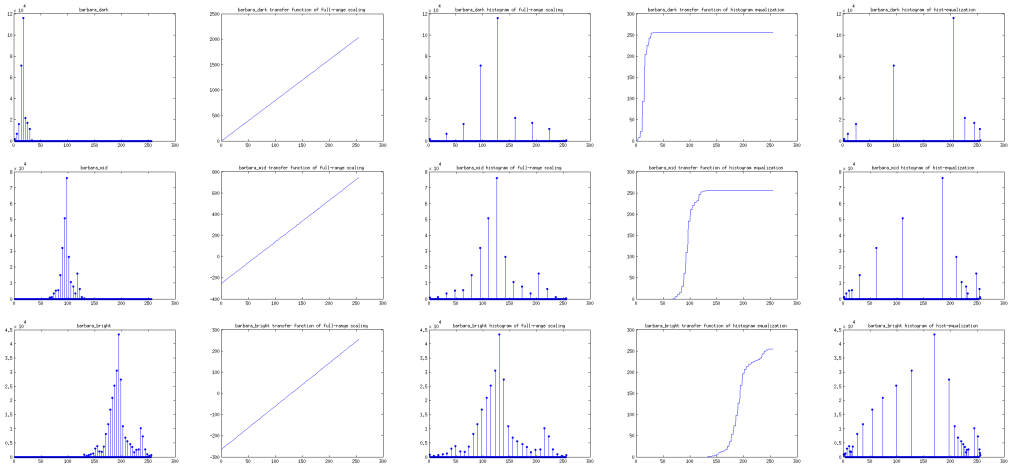

Figure 8: From left to right: Histogram of input image, Transfer function of full-range, Histogram of output image of full-range, Transfer function of histogram equalization, Histogram of output image of histogram equalization

Figure 9: From left to right: Input Image, Full-range Scaling, Histogram Equalization

From the results we can see the full-range scaling method stretched the histogram to the full range. However, gray levels that are close to each other in the original image stays close in the transformed image. As for the histogram equalization method, the distance between two nearby gray level is changed non-linearly.

3.2.2 Part (b): Histogram Equalization for Color Images

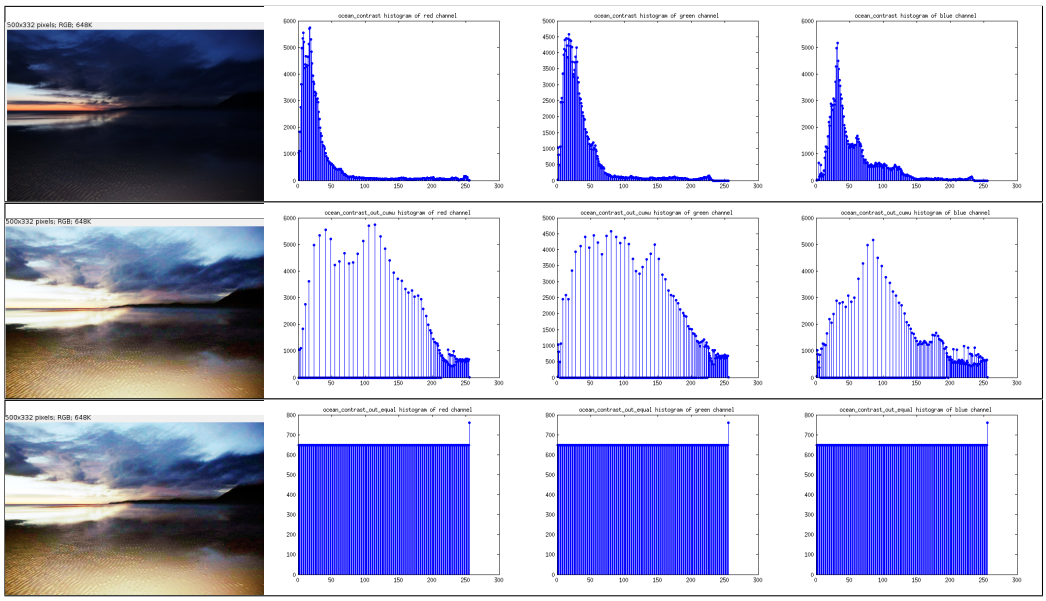

Figure 10: From left to right: Image, Histogram of Red Channel, Green Channel, Blue Channel; From top to bottom: Input, Output of Cumulative Histogram Method, Output of Equalized Histogram Method

We can see that the histograms of results of equalized histogram method is uniformed. The only difference occurs in 255, that's because the total number of pixels is not divisible by 256. My algorithm just sets those remaining pixels to 255. The major difference between these two histogram equalization method is that the cumulative method is a "one-to-one" mapping and the second method is a "one-to-many" mapping, which can be shown with the experiment below.

An example showing the drawbacks of equalized histogram method:

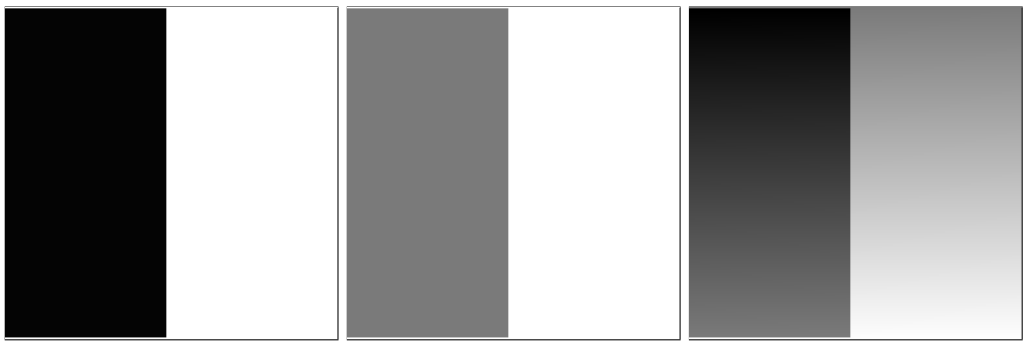

Figure 11: Left: Original; Middle: Cumulative method; Right: Equalized histogram method

We can see from above, a two-color image becomes a multi-color image after using the 2nd histogram equalization method.

3.2.3 Part (c): Histogram Transform

(1) $\mu = 120, \sigma = 50$:

Figure 12: Left:Input Image, Right:Output Image

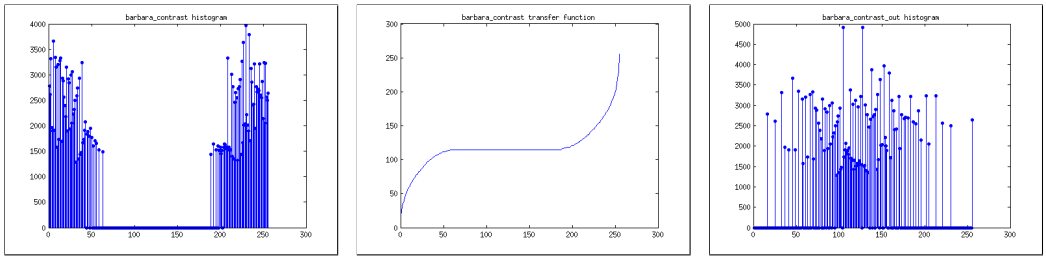

Figure 13: Left:Histogram of Input Image, Middle:Transfer Function, Right:Histogram of Output Image

(2) $\mu = 70, \sigma = 20$:

Figure 14: Left:Input Image, Right:Output Image

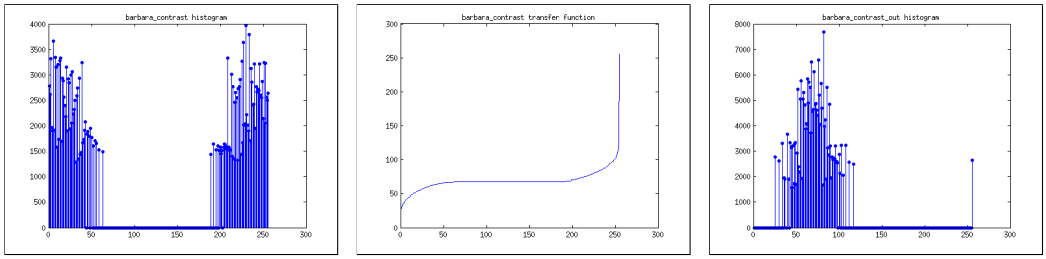

Figure 15: Left:Histogram of Input Image, Middle:Transfer Function, Right:Histogram of Output Image

We can see the mean gray level of output histogram always falls at about

μ. Frequency also follows the "68-95-99.7" rule. So the output is approximately the desired form.

According to the algorithm, 255 always transforms to 255. This seems to be a problem (like white spot shown in the picture above) when the desired distribution is with a small mean as well as a small variance (just like 2nd experiment above). I changed the algorithm a little bit by changing the persentage of used cumulation (99.99%) to solve this problem to make output picture smooth. Results are shown below.

Figure 16: Left:Input Image, Right:Output Image

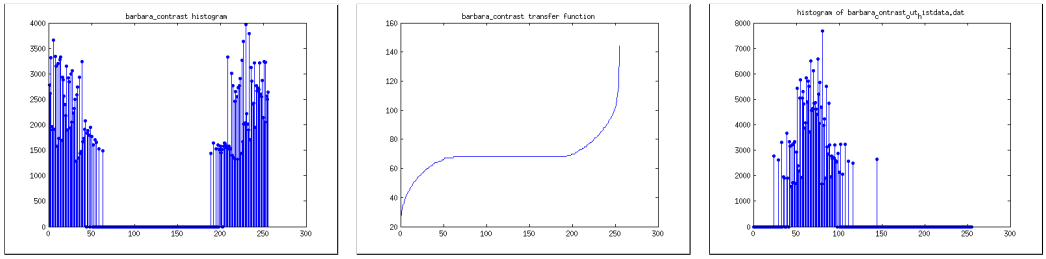

Figure 17: Left:Histogram of Input Image, Middle:Transfer Function, Right:Histogram of Output Image

We can see that 255 transfered to a value not that far as in the original algorithm and the transfer function is not that sharp near 255 after the modification.

3.3 Problem 3

3.3.1 Part (a): Mix Noise in Color Image

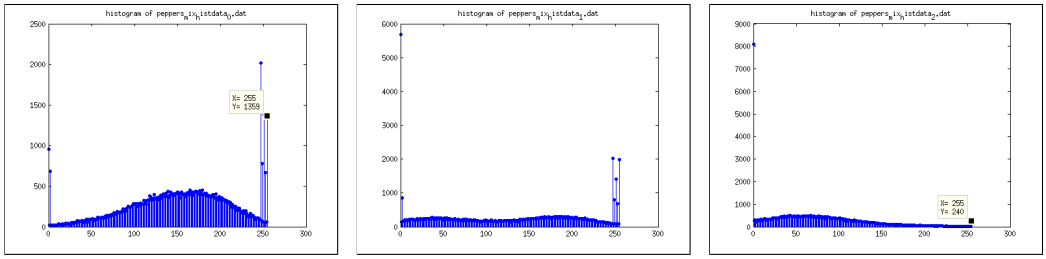

Figure 18: Histogram of 3 channels

We can see from the histograms above that there are "pepper and salt" noise in three channels. There's also some random noise in these channels. To apply filters to image, first we should expand the original image. In this task, I use the "repeating" method that expands image by repeating boundary line.

219,44,14	219,44,14	219,44,14	191,116,79	141,101,
219,44,14	219,44,14	219,44,14	191,116,79	141,101,
219,44,14	219,44,14	219,44,14	191,116,79	141,101,
156,195,153	156,195,153	156,195,153	141,85,96	200,42,
195,100,12	195,100,12	195,100,12	188,23,160	221,124,
180,64,10	180,64,10	180,64,10	224,101,9	2,2,
0,0,108	0,0,108	0,0,108	132,25,22	187,4
185,125,63	185,125,63	185,125,63	180,144,68	166,82,
243,99,47	243,99,47	243,99,47	174,97,30	165,62,
227,24,37	227,24,37	227,24,37	201,98,90	162,89,
194,105,73	194,105,73	194,105,73	165,104,75	215,143,
191,98,104	191,98,104	191,98,104	194,143,71	176,197,

Figure 19: Image Expanding: Expanded by 2 lines

Since image filtering is not linear process, cascading filters in different

orders may have different results.(Here I use 3 by 3 low-pass and median filters)

Figure 20: Left: First Low-pass then Median; Right: First Median then Low-pass

It seems that the result is better if we first use median filter then low-pass filter. That makes sense because if we use low-pass filter first, those "pepper and salt" noises will be averaged around.

Now let's change the size of low pass filter:

Figure 21: Filters with different sizes: 3, 7, 11, 15

We can see that as the size grows up, the image is getting blurrier and blurrier. This is the averaging effect of low-pass filters.

By changing filters' size, we can get a relatively better denoised image. But the obvious shortcoming of this simply method is that when we use low-pass filter, we average pixels in the window hoping that noise is averaged out. However, original image color is also averaged with this process. That

mean we will lose useful information. A simple improvement will be that we use Gussian filter instead of uniform low-pass filter. We know that nearby pixels usually share similar colors. Using Gaussian will introduce spatial information when we doing the averaging. However, this still will blurs edges. A more advanced method is to introduce not only spatial but also graylevel range information to the denoising process. This is the so-called "Bilateral Filtering Method".

3.3.2 Part (b): Bilateral Filtering [1]

Figure 22: A Result of Bilaterial Filtering with $\sigma_c = 5, \sigma_s = 50$ and windowsize = 7

As for the performances with respect to different σ_c and σ_s, windowsize=7:

Figure 23: From top to bottom: $\sigma_c = 1, 5, 9$; from left to right: $\sigma_s = 10, 50, 90$

σ_c is the variance of domain function which corresponds to the spatial weight. σ_s is variance of range function which corresponds to the range weight. Smaller σ_c basically means more "picky" in space and farther pixels make less contribution to the center point. Smaller σ_s means if 2 pixels are not close in graylevel, they contribute less to each other. We can see from the comparison above, with the same σ_s, larger σ_c causes the result to be blury which is also an averaging effect like low-pass filter.

Let's play with the window size. Window of Bilaterial Filter determines the searching area. I use $\sigma_c = 5$ *and* $\sigma_s = 50$ here.

Figure 24: From left to right: Windowsize = 3,11,19,27

We can see for chosen σ_s and σ_c, increasing window size will give little improvement when window is large enough. That's because when searching window is large enough, the "domain function" takes power and controls the weight.

Now compare the performance between bilateral filter and low-pass filter of the same size (7x7):

Figure 25: Left: Bilateral ; Right: Low-pass

Clearly, output of bilateral filter looks much sharper than that of low-pass filter. Let's experiment with another black and white image("blackwhite.raw",400x400)

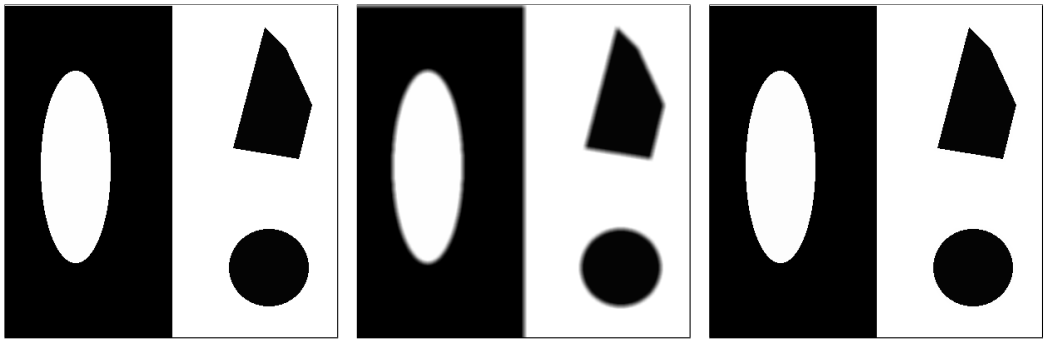

Figure 26: Left: Original ; Middle: after Low-pass ; Right: after Bilateral

We can conclude that bilateral filter preserves edges better than low-pass filter.

3.3.3 Part (c): Non-Local Means (NLM) Filtering

Generally, in NLM we have to set 3 parameters: NeighborSize (defines context), SearchWindowSize and weight function parameter (h).[2]

Let's compare the performance of different neighbor size: (SearchWindowSize=21, h=180)

Figure 27: NeighborSize = 5,7,9,11

We can see when neighbor size is small, the impact of context is little so that the image is blurry. However, when the neighbor size is too large, we can hardly find a pixel with similar neighborhood and get a worse denoising result.

Let's compare the performance of different searching window size: (NeighborSize = 7, h = 180):

Figure 28: Searching Window Size = 10,20,30

Larger searching window size may allow us to find more similar pixels, however, at an expense of heavier computation.

Let's compare the performance of different h: (SearchWindowSize=21, NeighborSize = 7):

Figure 29: h = 120,150,180,210

Smaller "h" means less tolerance to block difference, which has a similar effect with larger neighbor size. Larger "h", the result becomes more smooth and less sharp.

A comparison with "Bilateral filter" and "Low-pass filter":(WindowSize=7)

Figure 30: Low-pass, Bilateral, NLM

We can see the great improvement with NLM.
An experiment with "blackwhite.raw":

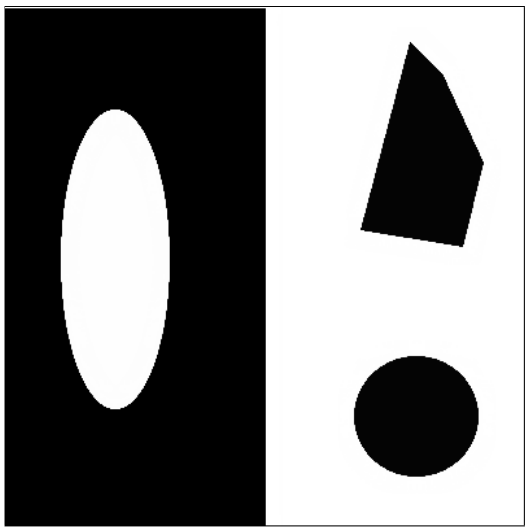

Figure 31: Output of NLM filter

The NLM filter also preserves edges well.

We can also conclude there's a connection between NLM and bilateral filtering method. If we let the neighbor size of NLM method to be one and the searching window to be the same size as the window size in bilateral filter, then it's just a special case of bilateral filter with normal domain weight function (i.e "c" function is constant 1 and "s"function is exponential).

3.3.4 Part (d): Block Matching 3-D (BM3D) Transform Filter

In BM3D, we can change the expected noise standard variance. Some results are as follows using software provided by author [4].

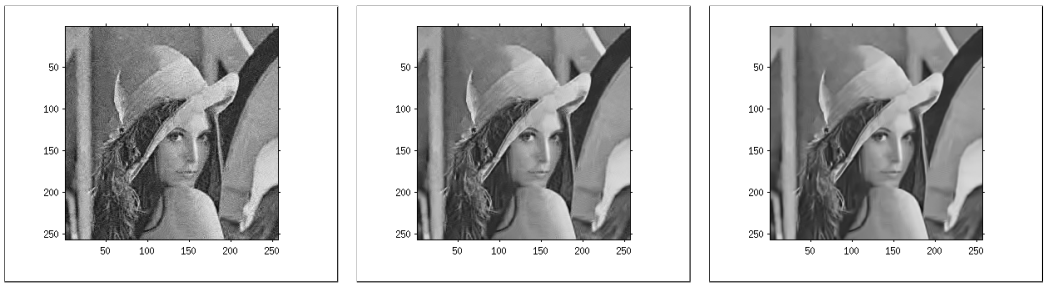

Figure 32: BM3D: $\sigma = 20, 25, 30$

As the expected noise standard variance grows, the output image becomes smoother and smoother.

BM3D is majorly a frequency domain method because it uses 3-D transformation and denoise in transformation domain. It still uses block matching method to find similars blocks and form a block stack. I think this part belongs to spatial domain.[3]

A comparison with "Bilateral filter" and NLM method:

Figure 33: Left:Bilateral filter, Middle:NLM, Right:BM3D($\sigma = 30$)

We can see the performance is improved significantly with BM3D method especially BM3D preserves more details than NLM.

References

[1] C. Tomasi and R. Manduchi, "Bilateral filtering for gray and color images" in *Computer Vision, 1998. Sixth International Conference* on IEEE, 1998, pp. 839846.

[2] A. Buades, B. Coll, and J.-M. Morel, "A non-local algorithm for image denoising" in *Computer Vision and Pattern Recognition, 2005.* CVPR 2005. IEEE Computer Society Conference on, vol. 2. IEEE, 2005, pp. 6065.

[3] K. Dabov, A. Foi, V. Katkovnik, and K. Egiazarian, "Image denoising by sparse 3-d transform-domain collaborative filtering" *mage Processing* EEE Transactions on, vol. 16, no. 8, pp. 20802095, 2007.

[4] [Online]: `http://www.cs.tut.fi/foi/GCF-BM3D/`