EE 569: Homework #1

Issued: 1/7/2019 Due: 11:59PM, 1/22/2019

General Instructions:

1. Read *Homework Guidelines* and *MATLAB Function Guidelines* for the information about homework programming, write-up and submission. If you make any assumptions about a problem, please clearly state them in your report.
2. You need to understand the USC policy on academic integrity and penalties for cheating and plagiarism. These rules will be strictly enforced.

Problem 1: Image Demosaicing and Histogram Manipulation (50%)

(a) Bilinear Demosaicing (10%)

To capture color images, digital camera sensors are usually arranged in form of a color filter array (CFA), called the Bayer array, as shown in Figure 1. Since each sensor at a pixel location only captures one of the three primary colors (R, G, B), the other two colors have to be re-constructed based on their neighbor pixel values to obtain the full color. Demosaicing is the process of translating this Bayer array of primary colors into a color image that contains the R, G, B values at each pixel.

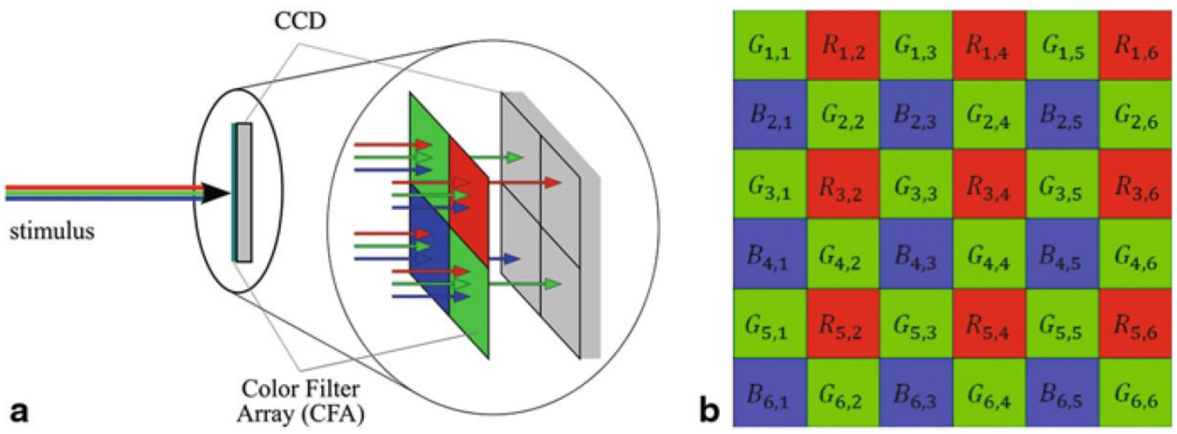

CCD
stimulus
Color Filter Array (CFA)
a
b
$G_{1,1}$ $R_{1,2}$ $G_{1,3}$ $R_{1,4}$ $G_{1,5}$ $R_{1,6}$
$B_{2,1}$ $G_{2,2}$ $B_{2,3}$ $G_{2,4}$ $B_{2,5}$ $G_{2,6}$
$G_{3,1}$ $R_{3,2}$ $G_{3,3}$ $R_{3,4}$ $G_{3,5}$ $R_{3,6}$
$B_{4,1}$ $G_{4,2}$ $B_{4,3}$ $G_{4,4}$ $B_{4,5}$ $G_{4,6}$
$G_{5,1}$ $R_{5,2}$ $G_{5,3}$ $R_{5,4}$ $G_{5,5}$ $R_{5,6}$
$B_{6,1}$ $G_{6,2}$ $B_{6,3}$ $G_{6,4}$ $B_{6,5}$ $G_{6,6}$

Figure 1: (a) Single CCD sensor covered by a CFA and (b) Bayer pattern [1].

Implement the simplest demosaicing method based on bilinear interpolation. Exemplary demosaicing results are given in Figure 2. With this method, the missing color value at each pixel is approximated by bilinear interpolation using the average of its two or four adjacent pixels of the same color. To give an example, the missing blue and green values at pixel $R_{3,4}$ are estimated as:

$$\hat{B}_{3,4} = \frac{1}{4}(B_{2,3} + B_{2,5} + B_{4,3} + B_{4,5})$$

$$\hat{G}_{3,4} = \frac{1}{4}(G_{3,3} + G_{2,4} + G_{3,5} + G_{4,4})$$

As for pixel $G_{3,3}$, the blue and red values are calculated as:

$$\hat{R}_{3,3} = \frac{1}{2}(R_{3,2} + R_{3,4})$$

$$\hat{B}_{3,3} = \frac{1}{2}(B_{2,3} + B_{4,3})$$

(1) Apply the bilinear demosaicing to the cat image in Figure 3 and show your results.

(2) Do you observe any artifacts? If yes, explain the cause of the artifacts and provide your ideas to improve the demosaicing performance.

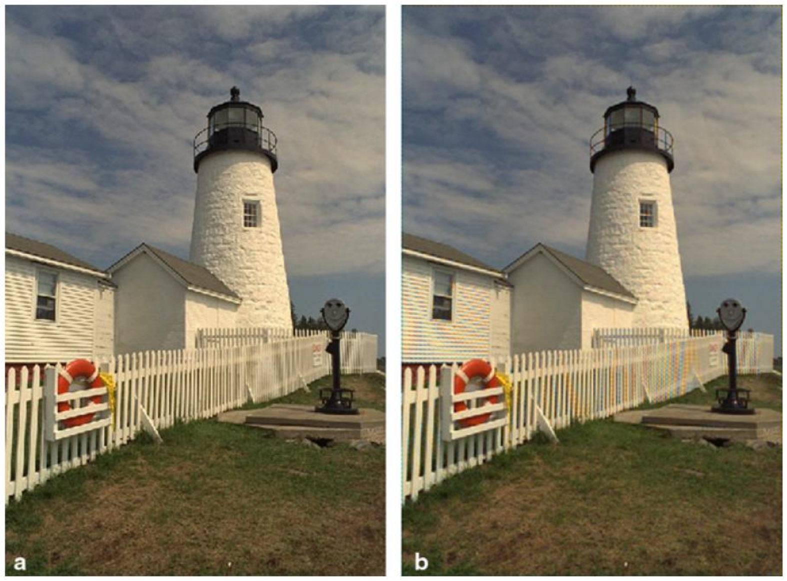


Figure 2: (a) The original lighthouse image and (b) the demosaiced lighthouse image by bilinear interpolation.

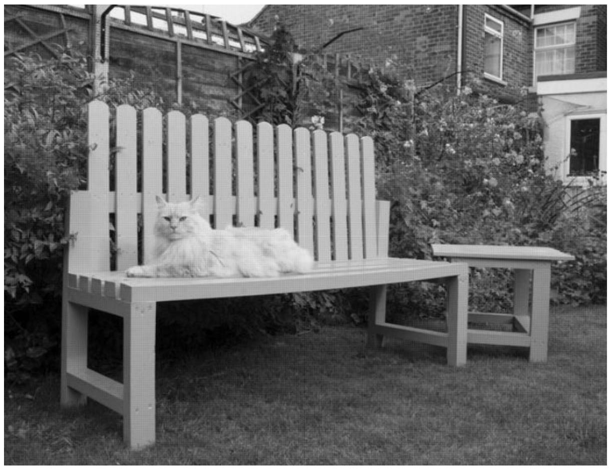


Figure 3: The cat image with the CFA sensor input.

(b) Malvar-He-Cutler (MHC) Demosaicing (20%)

Malvar et al. [1] proposed an improved linear interpolation demosaicing algorithm. It yields a higher quality demosaicing result by adding a 2nd-order cross-channel correction term to the basic bilinear demosaicing result. Both the bilinear and the MHC demosaicing results of the *Fruit_Shop* image are shown in Figure 4.

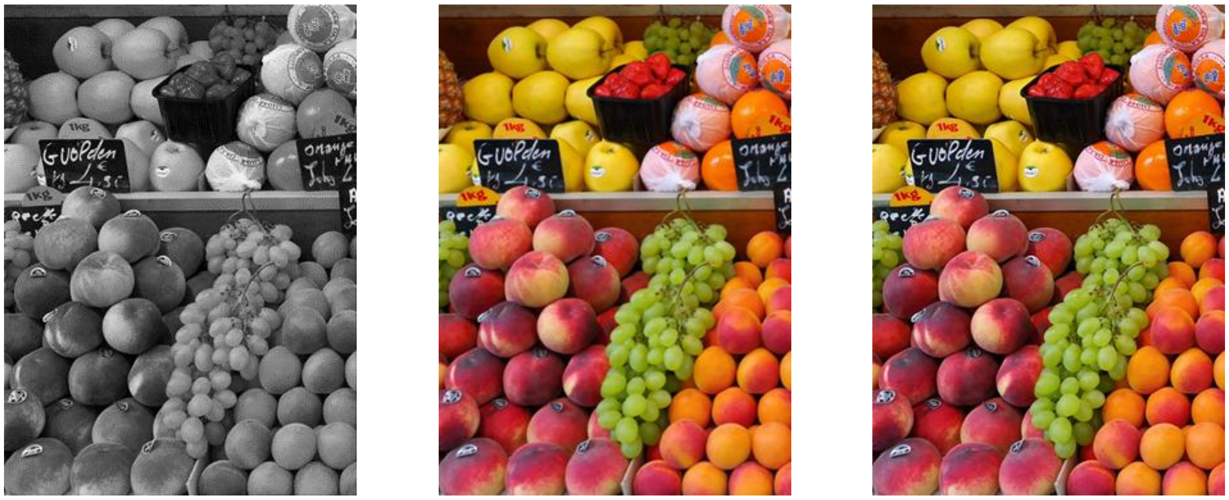


Figure 4: **Demosacing results of Fruit_Shop image: the CFA input (left), the bilinear demosaicing result (middle) and the MHC demosaicing result (right)**

The MHC algorithm is stated below.

To estimate a green component at a red pixel location, we have

$$\hat{G}(i,j) = \hat{G}^{bl}(i,j) + a\mathrm{D}_R(i,j) \tag{1}$$

where the 1st term at the right-hand-side (RHS) is the bilinear interpolation result given in (1) and the 2nd term is a correction term. For the 2nd term, alpha is a weight factor, and Δ_R is the discrete 5-point Laplacian of the red channel:

$$\mathrm{D}_R(i,j) = R(i,j) - \frac{1}{4}\big(R(i-2,j) + R(i+2,j) + R(i,j-2) + R(i,j+2)\big) \tag{2}$$

To estimate a red component at a green pixel location, we have

$$\hat{R}(i,j) = \hat{R}^{bl}(i,j) + b\mathrm{D}_G(i,j) \tag{3}$$

where Δ_G is a discrete 9-point Laplacian of the green channel.

To estimate a red component at a blue pixel location,

$$\hat{R}(i,j) = \hat{R}^{bl}(i,j) + g\mathrm{D}_B(i,j) \tag{4}$$

where Δ_B is a discrete 9-point Laplacian of the blue channel. The weights a, b, g control how much correction is applied, and their default values are:

$$a = \frac{1}{2}, b = \frac{5}{8}, g = \frac{3}{4} \tag{5}$$

The above formulas can be generalized to missing color components at each sensor location. Consequently, the MHC demosaicing can be implemented by convolution with a set of linear filters. There are eight different filters for interpolating the different color components at different locations as illustrated in Figure 5 [1].

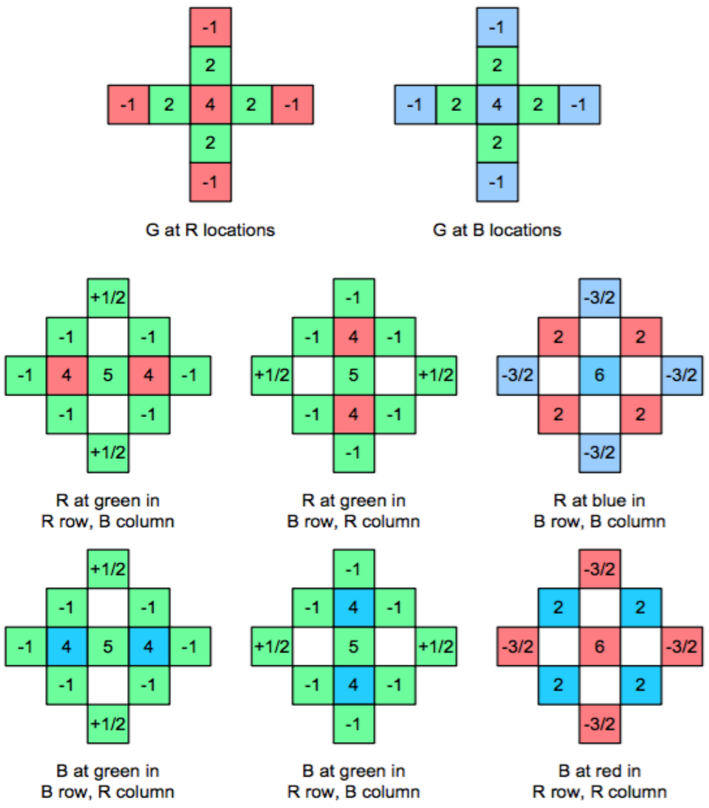

G at R locations
G at B locations
R at green in R row, B column
R at green in B row, R column
R at blue in B row, B column
B at green in B row, R column
B at green in R row, B column
B at red in R row, R column

Figure 5: Filter coefficients

(1) Implement the MHC linear demosaicing algorithm and apply it to the *Cat* image in Figure 3. Show your results.

(2) Compare the MHC and the bilinear demosaicing results and explain the performance differences between these two algorithms in your own words.

(c) Histogram Manipulation (20%)

Implement two histogram equalization techniques:

- Method A: the transfer-function-based histogram equalization method
- Method B: the cumulative-probability-based histogram equalization method

Enhance the contrast of images *rose_dark* and *rose_bright* in Figure 6 using both methods. Describe the procedure and show final results with the following steps.

(1) Plot the histograms of all images (2 input and 4 output images). The figure should have the intensity value as the x-axis and the number of pixels as the y-axis.

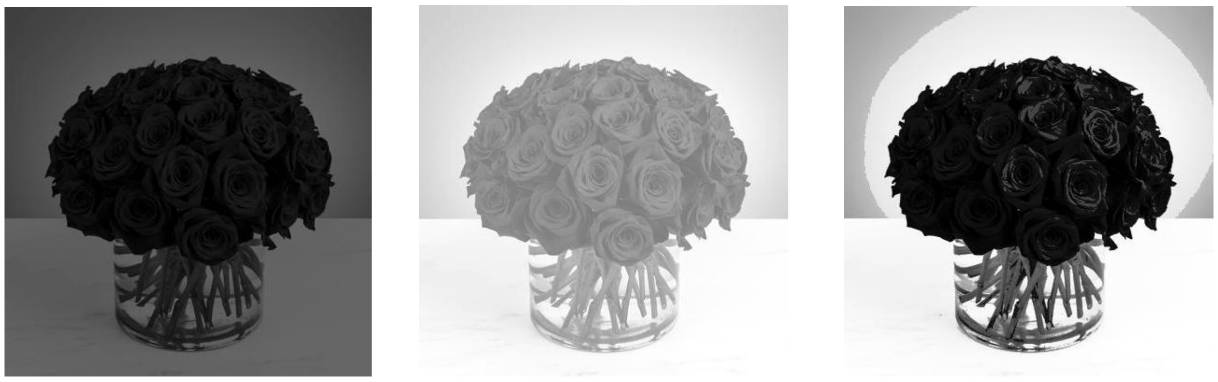
(2) Apply Method A to *rose_dark* and *rose_bright* and show the enhanced image. Plot the transfer function for each testing image.

(3) Apply Method B to *rose_dark* and *rose_bright* and show the enhanced image. Plot the cumulative histogram for each testing image.

(4) Discuss your observations on these two enhancement results. Do you have any idea to improve the current result?

(5) Apply your implemented Method A and Method B to *rose_mix* in Figure 4 and show the result. Can you get similar result as in previous part? If not, how to modify your implementation? Please justify your answer with discussion.

Note that MATLAB users CANNOT use functions from the Image Processing Toolbox except displaying function like imshow().



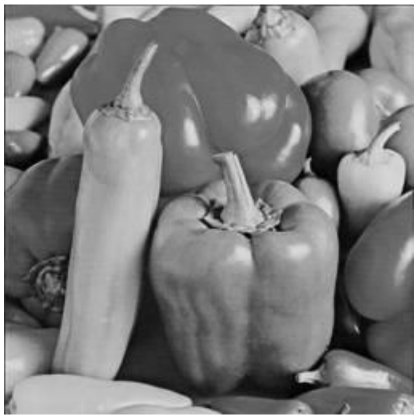
(a) rose_dark **(b) rose_bright** **(c) rose_mix**

Figure 6: Three rose images with different gray-scale histograms.

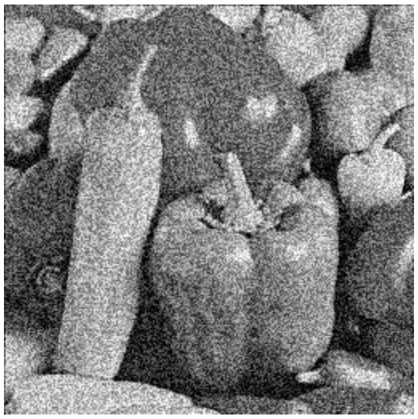
Problem 2: Image Denoising (50%)

(a) Gray-level image (20%)

Remove noise in the image in Figure 7(b), compare it with the original image in Figure 7(a), and answer the following questions:



(a)



(b)

Figure 7: (a) the original pepper image and (b) the pepper image with added noise.

(1) What is the type of embedded noise in Figure 7(b)?

(2) Apply a linear filter of size *N* by *N* to the noisy image. Compare the performance of two choices of the filter parameters – the uniform weight function and the Gaussian weight function and study the effect of the window size (i.e. parameter *N*). Plot the peak-to-signal-noise (PSNR) value as a function of *N* for both weighting functions.
Note: The PSNR value between images I and Y can be calculated as:

$$\text{PSNR(dB)} = 10\log_{10}\left(\frac{\text{Max}^2}{\text{MSE}}\right) \tag{1}$$

where

$$\text{MSE} = \frac{1}{NM}\sum_{i=1}^{N}\sum_{j=1}^{M}\left(Y(i,j) - I(i,j)\right)^2 \tag{2}$$

and where I is the noise-free image of size N × M, Y is the filtered image of size N × M, Max for 8-bit image is 255.

(3) In most low-pass linear filters, we often see degradation of edges. However, using some non-linear filters, we can preserve the edges. Bilateral filters are one such kind of filters. A discrete bilateral filter is given by:

$$Y(i,j) = \frac{\sum_{k,l} I(k,l)w(i,j,k,l)}{\sum_{k,l} w(i,j,k,l)} \tag{3}$$

$$w(i,j,k,l) = exp\left(-\frac{(i-k)^2 + (j-l)^2}{2\sigma_c^2} - \frac{\|I(i,j) - I(k,l)\|^2}{2\sigma_s^2}\right) \tag{4}$$

where (k,l) is the neighboring pixel location within the window centered around (i,j), I is the image with noise, Y is the filtered image. σ_c and σ_s are the spread parameters. Implement the bilateral denoising filter and apply it to the noisy image.

(4) The non-local mean filter utilizes the pixel value from a larger region rather the mean of a local window centered around the target pixel. A discrete non-local mean filter with Gaussian weighting function is as follows:

$$Y(i,j) = \frac{\sum_{k=1}^{N'} \sum_{l=1}^{M'} I(k,l) f(i,j,k,l)}{\sum_{k=1}^{N'} \sum_{l=1}^{M'} f(i,j,k,l)} \tag{5}$$

$$f(i,j,k,l) = \exp\left(- \frac{\left\| I(N_{i,j}) - I(N_{k,l}) \right\|_{2,a}^2}{h^2} \right) \tag{6}$$

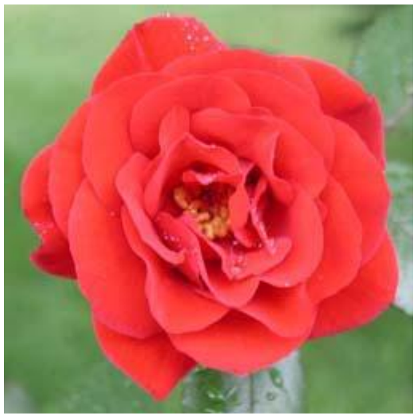
where I, Y are the noisy and filtered images respectively, $N_{x,y}$ is the window centered around location (x, y), and h is the filtering parameter, $N' \leq N$ and $M' \leq M$ denote the window size of your choice.
The Gaussian weighted Euclidian distance between window $I(N_{i,j})$ and $I(N_{k,l})$ is defined as:

$$\left\| I(N_{i,j}) - I(N_{k,l}) \right\|_{2,a}^2 = \sum_{n_1, n_2 \in N} G_a(n_1, n_2)(I(i-n_1, j-n_2) - I(k-n_1, l-n_2))^2 \tag{7}$$

and

$$G_a(n_1, n_2) = \frac{1}{\sqrt{2\pi} a} \exp\left(-\frac{n_1^2 + n_2^2}{2a^2}\right) \tag{8}$$

where N denotes the local neighborhood centered at the origin, $n_1, n_2 \in N$ denotes the relative position in the neighborhood window. $a > 0$ is the standard deviation of the Gaussian kernel.
Implement the non-local mean filter and apply to the noisy image.


(a)

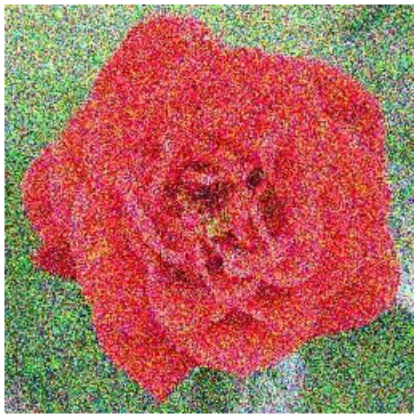

(b)

Figure 8: (a) the original rose image (b) the rose image with mixed noise.

(b) Color image (20%)

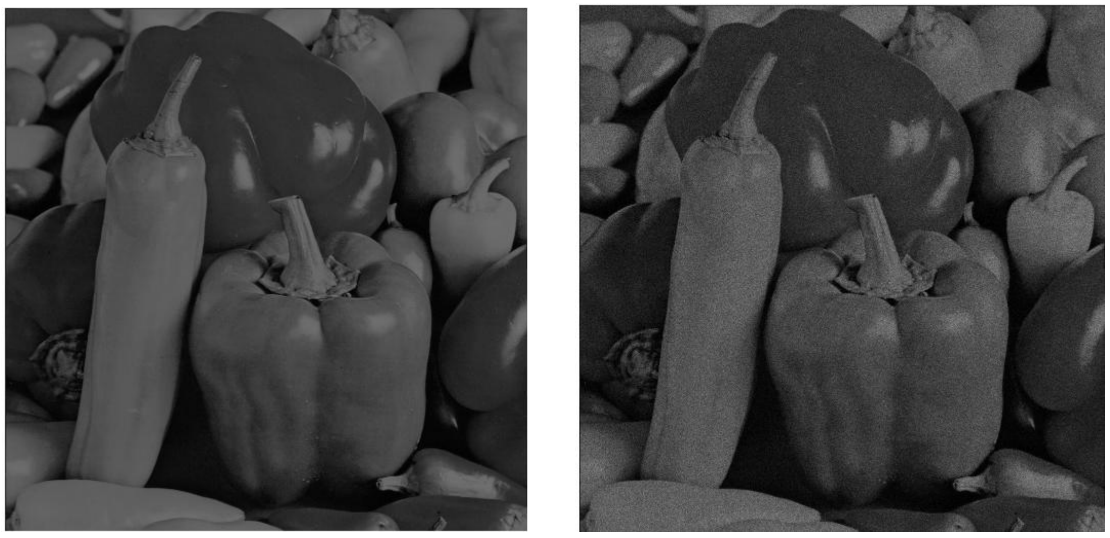
Figure 8 (b) is a noisy color image, where each channel is corrupted by both impulse and uniform noises. Try your best to remove noise and compare the result with the original image as shown in Figure

8(a). Discuss your denoising strategy and answer the following questions:

(1) Should you perform filtering on individual channels separately for both noise types?
(2) What filters would you like use to remove mixed noise?
(3) Can you cascade these filters in any order? Justify your answer.
(4) It could be difficult to remove uniform noise satisfactorily. Filters may blur the object's edge when smoothing noise. A successful design of a uniform noise filter depends on its ability to distinguish the pattern between the noise and the edges. Please suggest an alternative to low pass filter with Gaussian weight function and discuss why such solution can potentially work better. (You are welcome to implement the filter, but it is not required)

(c) Shot noise (10%)

Electronic camera image sensor, especially the CMOS sensor typically has noise in the dark parts of the captured image. Such noise is called shot noise, which is caused by statistical quantum (the photon) fluctuations.



(a) (b)

Figure 9: (a) the original pepper image and (b) the pepper image with shot noise.

Figure 9(b) shows a typical image with shot noise. Unlike uniform noise, shot noise is Poisson distributed. A common solution is firstly applying the Anscombe root transformation on each pixel z:

$$f(z) = 2\sqrt{z + \frac{3}{8}}$$

to the input image, resulting an image with additive Gaussian noise of unit variance. Then, a conventional denoising filter for Gaussian noise is used. The denoised image is finally inverse transformed. The detailed denoising steps can be found in [3].

(1) Implement the denoising method in [3] and apply it to Figure 9(b), where you can remove the additive Gaussian noise using two methods.
- Use a Gaussian low pass filter
- Use the MATLAB code for block-matching and 3-D (BM3D) transform from [4].

(2) Show the final noise-removed pepper images and compare the PSNR values and visual quality of the resulting images.

Appendix:

Problem 1: Image Demosaicing and Histogram Manipulation

cat_ori.raw	390x300x3	24-bit	color(RGB)
cat.raw	390x300	8-bit	gray
rose_ori.raw	400x400	8-bit	gray
rose_dark.raw	400x400	8-bit	gray
rose_bright.raw	400x400	8-bit	gray
rose_mix.raw	400x400	8-bit	gray

Problem 2: Noise Removal

pepper.raw	256x256	8-bit	gray
pepper_noise.raw	256x256	8-bit	gray
rose_color.raw	256x256x3	8-bit	color(RGB)
rose_color_noise.raw	256x256x3	8-bit	color(RGB)
pepper_dark.raw	256x256	8-bit	gray
pepper_dark_noise.raw	256x256	8-bit	gray

Reference Images

All images in this homework are from the USC-SIPI image database [5] or Google images [6].

References

[1] M. E. Celebi et al. (eds.), Color Image and Video Enhancement
[2] Malvar, Henrique S., Li-wei He, and Ross Cutler. "High-quality linear interpolation for demosaicing of Bayer-patterned color images." Acoustics, Speech, and Signal Processing, 2004. Proceedings. (ICASSP'04). IEEE International Conference on. Vol. 3. IEEE, 2004.
[3] Makitalo, Markku, and Alessandro Foi. "Optimal inversion of the Anscombe transformation in low-count Poisson image denoising." IEEE transactions on Image Processing 20.1 (2011): 99-109.
[4] [Online]. Available: http://www.cs.tut.fi/~foi/GCF-BM3D/
[5] [Online] http://sipi.usc.edu/database/
[6] [Online] http://images.google.com/