Korean Advanced Institute of Science and Technology

School of Electrical Engineering

EE535 Digital Image Processing Spring 2018

Student Name: Dinh Vu

Student ID: 20184187

Homework 2

1. Math preliminary

1.1. Theoretical background and programming strategy

The convolution algorithm is based on the formula below:

$$Y = H * X \Rightarrow y(m,n) = \sum_{m=1}^{yRows} \sum_{n=1}^{yCols} h(m-m', n-n') \times x(m,n)$$

Where:

- X and Y is the input and output image, respectively
- H is kernel
- yRows and yCols are the number of rows and columns of the output image Y, respectively.

However, unlike normal convolution between 2 matrices, the size of Y equals to the size of X. So, it only requires zero padding enough to cover all image, such as in Problem 1, a zero boundary is added outside the input image.

The programming strategy and the result of Problem 1 is shown in Figure 1.1.

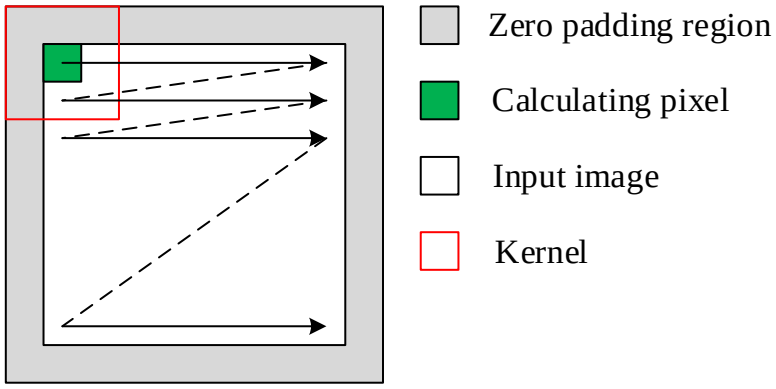


Figure 1.1. Programming strategy in Problem 1

1.2. Result and analysis result

The result of Problem 1 is presented in Figure 1.2. It is noticeable that the convolution with kernel H_1 making image smoother, less noise than with H_2. For kernel H_2, it is very hard to see the boundary of object.

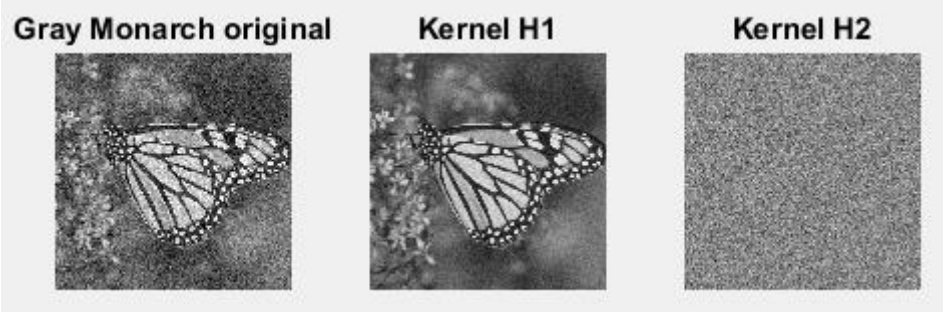


Figure 1.2. The result of Problem 1

2. Color Coordinate Transform

2.1. Theoretical background and programming strategy

Because of data format in RAW image, displayed in Figure 2.1, it is possible to extract each component Red, Green, Blue to three separable visible images.

R (1,1)	G (1,1)	B (1,1)	R (1,2)	G (1,2)	B (1,2)	……..	R (1,W)	G (1,W)	B (1,W)
……..									
R (H,1)	G (H,1)	B (H,1)	R (H,2)	G (H,2)	B (H,2)	……..	R (H,W)	G (H,W)	B (H,W)

Figure 2.1. Data format of RAW image (H: image height, W: image width)

Two other color systems HSI and YC_bC_r can be transformed from three images RGB. First, for HSI system, H presents Hue, the pure color itself, S is Saturation, colorfulness of a stimulus relative to its own brightness and I is intensity.

$$\begin{cases} I = \dfrac{R_N + G_N + B_N}{3} \\ S = 1 - \dfrac{\min(R_N, G_N, B_N)}{I} \\ H = \dfrac{1}{2\pi} arccos\left[\dfrac{2R_N - G_N - B_N}{2\sqrt{(R_N - G_N)^2 + (R_N - B_N)(G_N - B_N)}}\right] \end{cases}$$

Second, for YC_bC_r system, Y is the luminance component, similar to brightness and C_b and C_r are the blue-difference and red-difference chroma components.

$$\begin{bmatrix} Y \\ C_b \\ C_r \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.169 & -0.331 & 0.500 \\ 0.500 & -0.419 & -0.081 \end{bmatrix} \begin{bmatrix} R_N \\ G_N \\ B_N \end{bmatrix}$$

R_N, G_N, B_N are the value of R, G, B normalized, that mean:

$$R_N = \frac{R}{255}, \qquad G_N = \frac{G}{255}, \qquad B_N = \frac{B}{255}$$

2.2. Result and analysis result

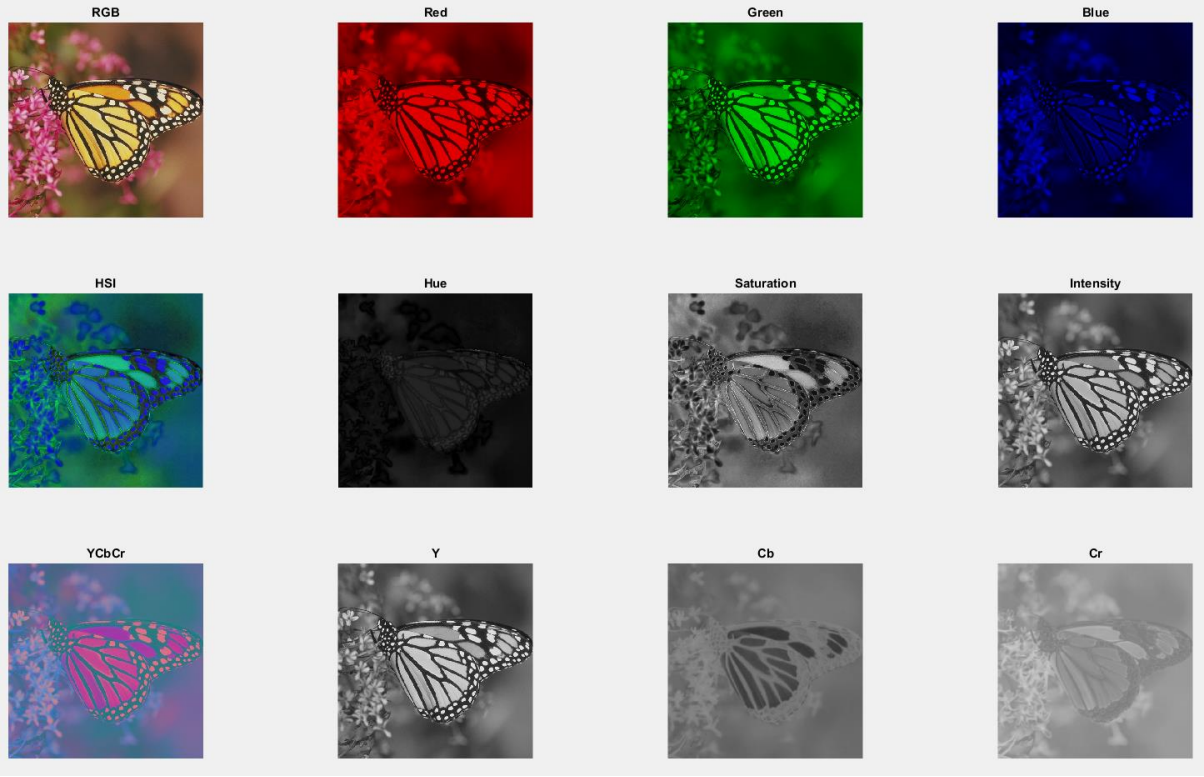


Figure 2.2. The result of Problem 2

For RGB system, Red component is most bright and clear, and Green component is more detail than Blue component. So, the average value of Red component is highest and the average value of Blue component is lowest.

For HSI system, Hue picture is all black because of the overwhelming of Red component. The Saturation picture focus on the changing of color, the wings of butterfly is more bright than other part. Intensity picture is gray scale of RGB picture.

In YC_bC_r system, Y picture looks similar to I picture because both of them are present gray scale of the original picture. Where C_b higher, Blue gets stronger such as inside the wings of butterfly. Where C_r higher, Red goes stronger like in the boundary of the wings.

3. Geometric Transform

3.1. Theoretical background and programming strategy

In affine transformation, there are 3 different types of transformation: translation, scaling and rotation. Using homogeneous coordinates, each transformation can be presented by formula below:

$$P' = T \times P, \qquad P' = S \times P, \qquad P' = R \times P$$

Where:

$$T = \begin{bmatrix} 1 & 0 & Tr\_x \\ 0 & 1 & Tr\_y \\ 0 & 0 & 1 \end{bmatrix}, \quad S = \begin{bmatrix} Sc\_x & 0 & 0 \\ 0 & Sc\_y & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad R = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

To implementing this transformation, it is required 3 functions: translation, scaling and rotation. In order to matching the pixel in image after transform with the original pixel, the inverse matrices of T, S and R are used.

If you put Tr_x, and Tr_y, Sc_x and Sc_y, Ang values predefined in the function with original image, an image in homogeneous coordinates is created by translation, scaling and rotation. In particular, to obtain the picture size of the resulting image to be rotated, only the vertices of the original image were rotated in advance, and the difference was calculated using min and max between x and y coordinates.

To matching the index of the transformed image and multiply it by the inverse matrix (above three matrices) as mention above. As a result, we were able to transform any floating point parameter, and the image quality was not broken much.

For the sake of convenience, the upper left corner of the image is considered as the origin, and the right direction is positive x direction and the lower direction is positive y direction. The input parameters can be modified in the code.

3.2. Result and analysis result

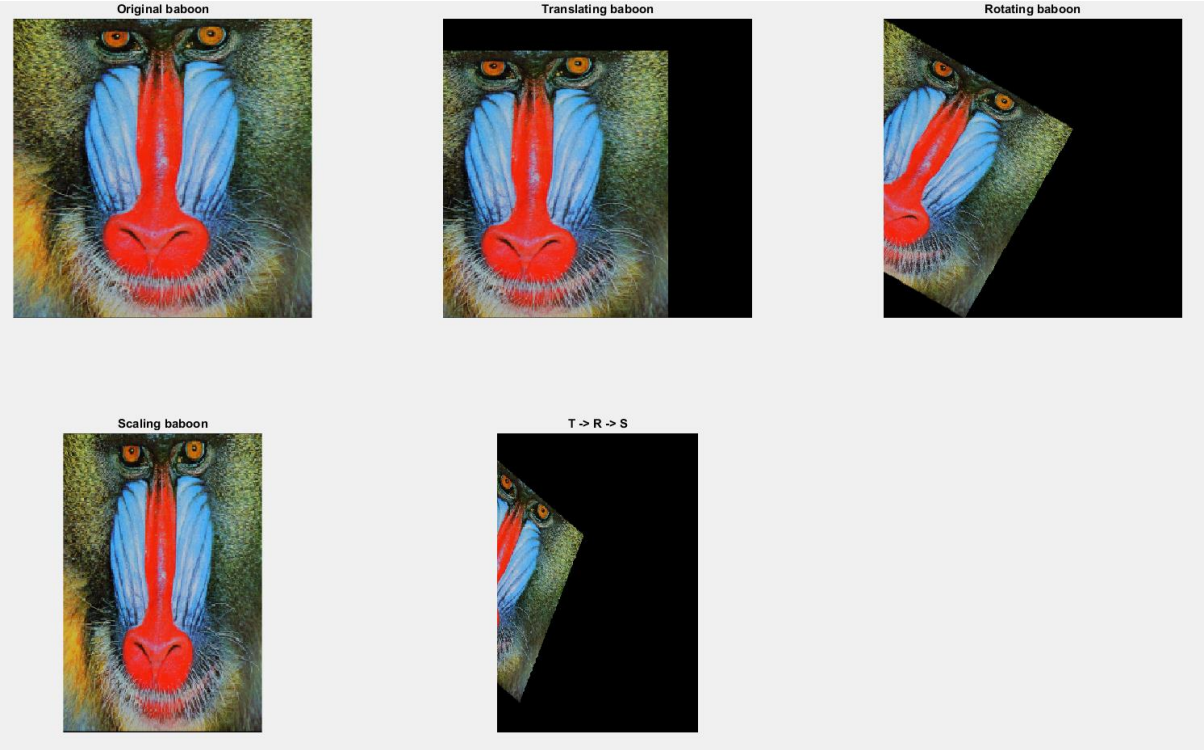


Figure 3.1. The result of Problem 3

The result of Problem 3 is shown in Figure 3.1. Some pixels out of range, is cannot be displayed because in my code, there is not expansion image in case the coordinates of pixel are greater than size of the original image.

4. Image Down and Up Sampling

4.1. Theoretical background and programming strategy

The flow of the solution is shown in Figure 4.1. Before down sampling image, the original image must come to Low-pass Filter (LPF) and before taking the up sampling image from down sampling image, it also needs LPF.

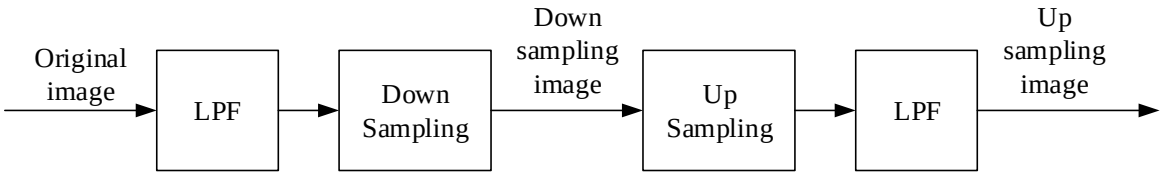


Figure 4.1. The flow of solution

LPF parameters depend on which kernel used. In Problem 4, the ratio of down and up sampling is 4:1, so the kernels of LPF after up sampling is:

$$K = \begin{cases} Peg = ones(4,4) \\ Pyramid = \dfrac{1}{16}(Peg \circledast Peg) \\ Cubic = \dfrac{1}{16}(Pyramid \circledast Pyramid) \end{cases}$$

In order to keep the size of image after filter unchanged, the kernel of LPF before down sampling is:

$$LPF_D = \frac{1}{16}K$$

Down and up sampling follow the ratio 4:1. When up sampling, some pixel have zero value and LPF will interpolate these value by convolution.

PSNR between the original image Orig and the up-sampling image Dist is calculated by the function below:

$$MSE = \frac{1}{m \times n \times p}\sum_{i=1}^{m}\sum_{j=1}^{n}\sum_{k=1}^{p}(Orig(i,j,k) - Dist(i,j,k))^2$$

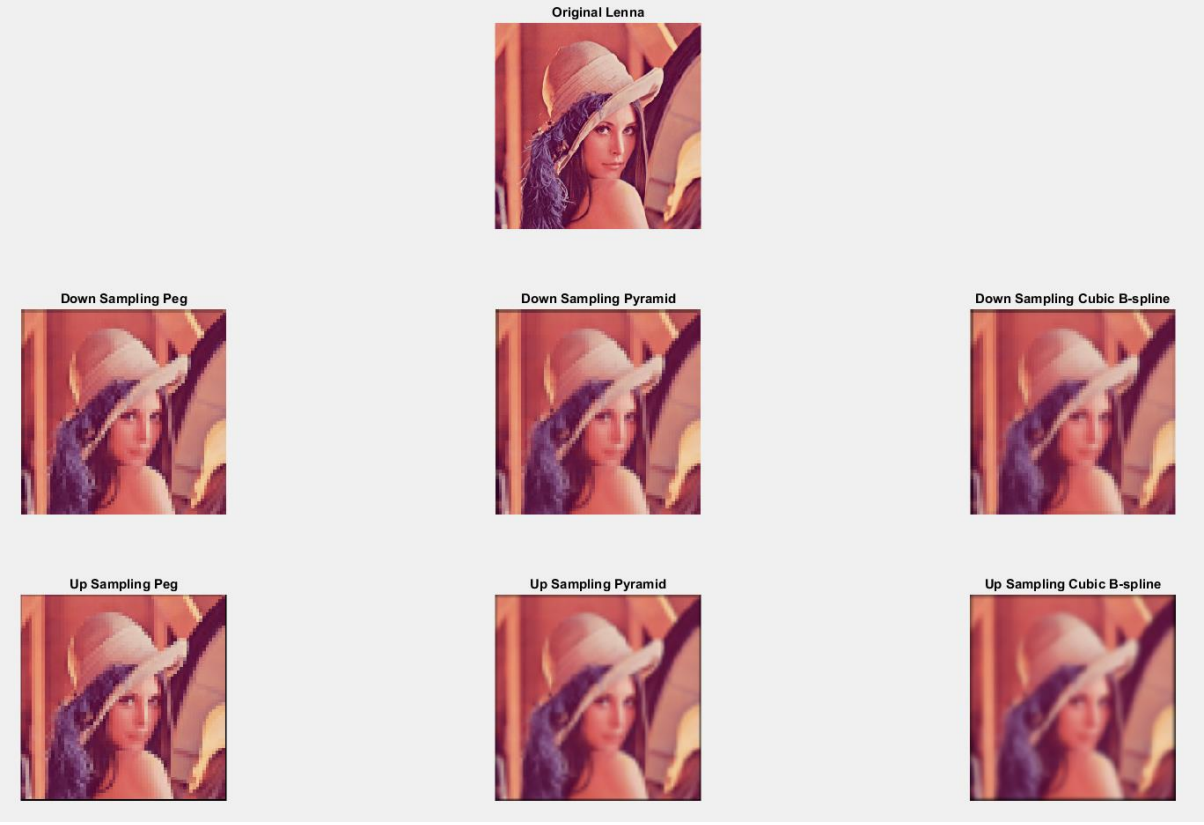
$$PSNR = 20 \times \log_{10}\left(\max_{i,j,k} Orig\right) - 10 \times \log_{10} MSE$$

Where:

- MSE: Mean Square Error
- m, n, p are the height, width and color of image, in this problem m = n = 256, p = 3.

4.2. Result and analysis result

The result of Problem 4 is displayed in Figure 4.2. It can be seen that using Peg kernel made the down and up image still have a lot of noise with PSNR = 19.71 dB. With Pyramid kernel, it seems like it is the most similar one, but the image looks blurrier than of Peg kernel however noise is lowest, PSNR = 22.06 dB. For Cubic B-spline kernel, it is most blurry and with PSNR = 20.92 dB, it is not better than Pyramid but far more than Peg. Conclusion, Pyramid kernel is the best interpolation kernel for down and up sampling.




```
Command Window
  PSNR_Peg = 19.706627 dB
  PSNR_Pyramid = 22.064474 dB
  PSNR_Cubic = 20.925003 dB
fx >>
```

Figure 4.2. The result of Problem 4

5. Quantization

5.1. Theoretical background and programming strategy

In Problem 5, there are two types of quantization implemented: uniform and Lloyd-Max quantization with 2, 4 and 6 bits. Before quantization, the gray image is added random noise range [-16 16] by using function rand(). The intensity of pixel is integer value from 0 to 255 and uniform quantize with L bits. So the quantization step $q = \frac{255}{2^L}$. From

q value, it can detect level of each pixel. After quantization, the image must be remove from noise by subtractor.

5.2. Result and analysis result

Table 5.1. The PSNR of uniform quantizer 2, 4, 6 bits

Quantizer	2 bits	4 bits	6 bits
Uniform	un_psnr_2 = 22.25 dB	un_psnr_4 = 34.54 dB	un_psnr_6 = 46.72 dB

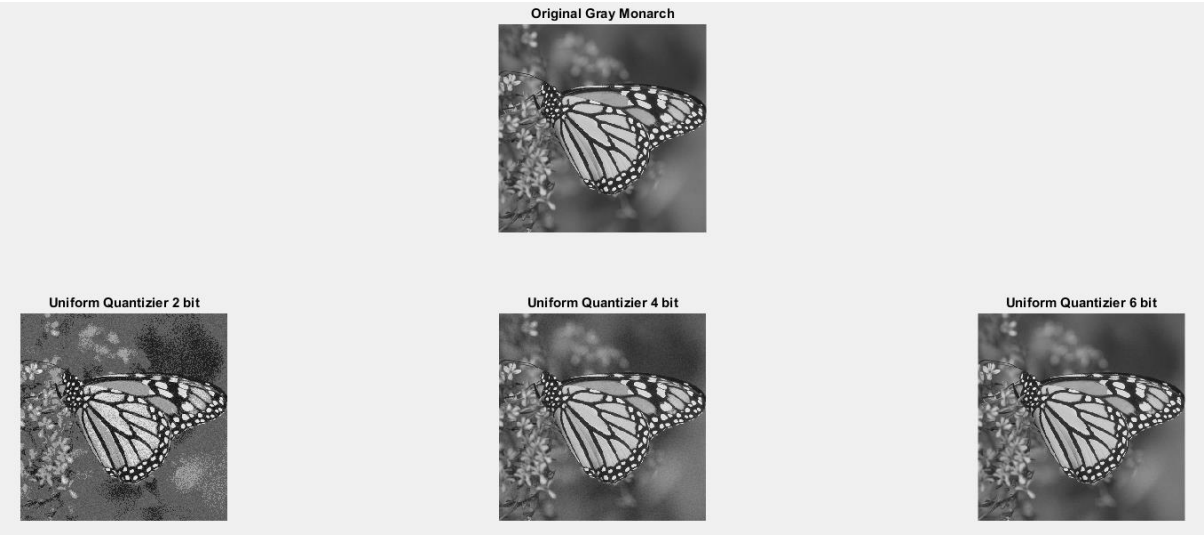


Figure 5.1. Uniform quantization 2, 4, 6 bits

The PSNR and result image is presented in Table 5.1 and Figure 5.1. The 4-bit and 6-bit uniform image look similar to the original image because there is more level near the original value.