EBU723U – Image and Video Processing – 2017/18

Coursework report and exercises

Name: _____Ziqian Chen_____

Username: ___qm140918481 / jp2014212849____

Exercise 1 (a)

Reading/writing PGM/PPM images: The first step towards image and video processing is reading images from a file and write them to a file. There exist different standards that store the information in different formats; so before opening an image, knowledge of the standard is necessary.

Two widely used image formats are PPM and PGM. The PGM format is a greyscale file format designed to be easy to manipulate. A PGM image represents a greyscale graphic image. For most purposes, a PGM image can just be thought of as an array of integers. The name "PGM" is the acronym of "Portable Grey Map." The name "PPM" is the acronym for "Portable Pixel Map." Images in this format (or a precursor of it) were once also called "portable pixmaps." It is a highly redundant format, and contains a lot of information that the Human Visual System (HVS) cannot even discern. However, as for PGM, PPM is very easy to write and analyse.

The goal of the first part of today's lab is to become comfortable with these two formats. You will implement functions to read and to write PPM and PGM images. The final demonstration of the implemented software will be done using the well-known test images: LENA, BABOON, PEPPERS, etc. You can find PPM and PGM versions of these images in the EBU723U QMplus pages. The writing function must add as a comment in the header: "image created by *your_name*".

Include in your submission the file resulting from reading the images provided and writing them back in their original format.

Summarize in 5 points the operations necessary to read a PGM/PPM image:

1. Read the magic number from the header, i.e., P2, P3 etc.
2. Find out the format by referring to the header.

   P2 – PGM, ASCII; P3 – PPM, ASCII; P5 – PGM, binary/byte; P6 – PPM, binary/byte

3. Read the width and height of the image from the header.
4. Read the maximum value of the colour components from the header, e.g, 255 means every single value of the encoded colour should be within [0,255].

5. Read the data according to the file's format identified in step2, i.e., decode the data by ASCII or Binary approach.

Summarize in 5 points the operations necessary to write a PGM/PPM image:

1. Specify the encoding format, i.e., PPM or PGM? ASCII or binary?
2. Write the corresponding magic number into the first line of the header.

   PGM, ASCII - P2; PPM, ASCII - P3; PGM, binary/byte - P5; PPM, binary/byte - P6

3. Write the width and height of the image into the second line of the header.
4. Write the maximum value of the colour components into the third line of the header.
5. Write in the data by the specified encoding format.

What is the difference between the identifiers P3 and P6?

P3 and P6 are the magic numbers stored in the header of a ppm file.

P3 indicates that the image adopts ASCII to encode the information, where the numerical value of each pixel ranges from 0 to the maximum value given in the header.

P6 indicates that the image adopts byte/binary format to store the data, i.e., one byte for each colour channel (R, G, B). Therefore, ppm format can only apply to single byte colours (0-255).

Moreover, P6 files are generally of smaller size than P3 ones. E.g., value 127 take 3 bytes using ASCII, as '1' '2' and '7' each take one byte. While using binary format it takes only one byte, 0111 1111.

Exercise 1 (b)

Format conversions: in this part of the lab, the images will be converted from colour to grey scale; in other words a PPM image will be converted to the PGM format. You will implement a function called "BUPT_format_converter" which transforms images from colour to grey-scale using the following YUV conversion:

$Y = 0.257 * R + 0.504 * G + 0.098 * B + 16$

$U = -0.148 * R - 0.291 * G + 0.439 * B + 128$

$V = 0.439 * R - 0.368 * G - 0.071 * B + 128$

Note swap of 2nd and 3rd rows, and sign-change on coefficient 0.368

What component represents the luminance, i.e. the grey-levels, of an image?

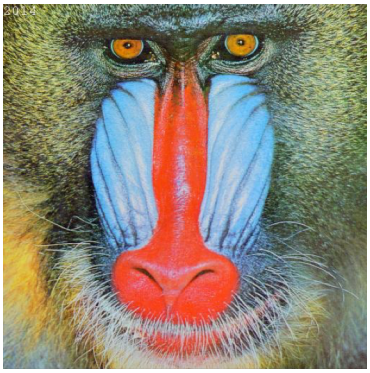
Y represents the luminance. (whereas U and V represent the chrominance)

Use these boxes to display the results for the colour to grey-scale conversion.

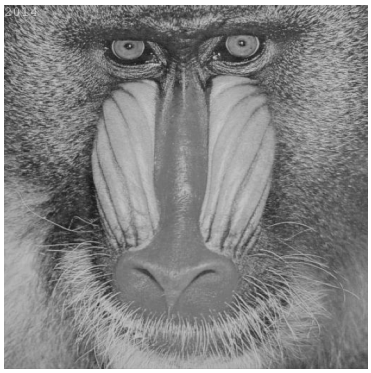
Lena colour (RGB)

Lena grey

Baboon colour (RGB)

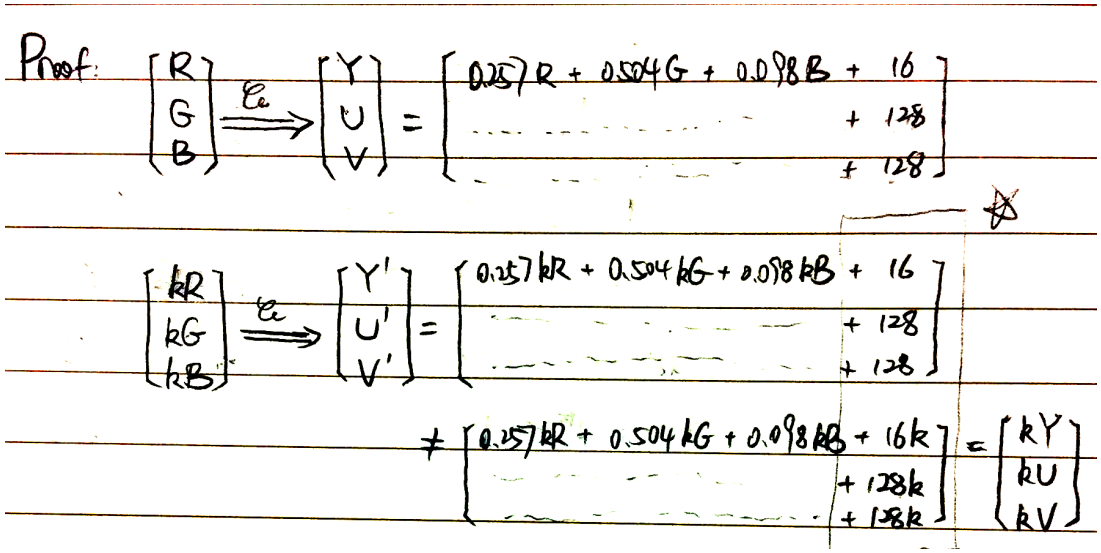


Baboon grey



Is the transformation between the two colour-spaces linear? Explain your answer.

No, because there are non-homogeneous terms in the formula, i.e. 16, 128, 128.



Display in the box the Lena image converted to YUV 3 channels format.

(taking YUV values as RGB channel)

Are the colours of the previous picture distorted? If yes why?

No. (Not sure what it means by 'distorted', but if it means 'irreversible', then no.)

The conversion between RGB and YUV is reversible and theoretically lossless. (In practice, minor errors might be seen due to the **rounding** operation)

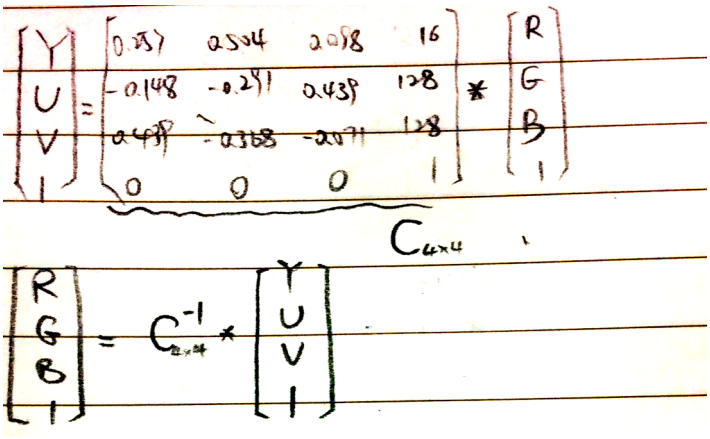
The colours seem different ('distorted') but are actually merely stored in another model, i.e., YUV; all information is kept.

Based on the formula for the RGB to YUV conversion, derive the formula for the YUV to RGB conversion.

The RGB-to-YUV conversion matrix could be written as a 4 by 4 matrix, where the non-homogeneous 4th dimension is added.

```
C = [0.257,  0.504,  0.098, 16;
    -0.148, -0.291,  0.439, 128;
     0.439, -0.368, -0.071, 128;
     0,      0,      0,     1];
```

Thus, the operation and its inverse transformation can be written as:

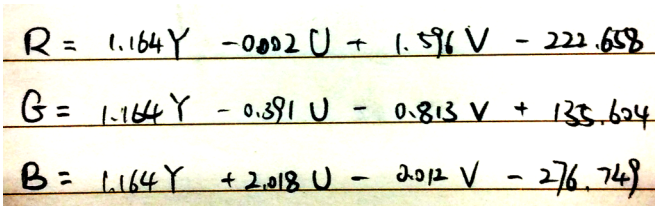



```
>> D = inv(C)

D =

    1.1641   -0.0018    1.5958 -222.6580
    1.1641   -0.3914   -0.8135  135.6041
    1.1641    2.0178   -0.0012 -276.7485
         0         0         0    1.0000
```

Finally, the formula is derived as below:



Use the formula you derived at the previous step to convert the YUV image back to the original RGB format. Display the result in the box.

Exercise 1 (c)

Sub-sampling: The HVS is incapable of perceiving certain details in an image. Therefore high compression ratios can be achieved by exploiting the characteristics of the HVS, thus discarding what has a low visual relevance. However, this process can introduce distortions due to the compression. A simple way to exploit the characteristics of the HVS to give compression is to sub-sample an image. A drawback of this approach is that it is possible to incur the well-known problems of a discrete representation, such as aliasing. This part of the lab covers some simple sub-sampling operations.

Implement a function that sub-samples grey level images by a factor n, with n a multiple of 2. The function should be able to sub-sample independently in the horizontal and in the vertical direction or in both directions at the same time.

Display the results of sub-sampling the image Lena using the following factors: 2 horizontal, 2 vertical, 2 vertical and 8 horizontal, 4 vertical and 4 horizontal. Include the files of the results in the submission.

Box for the 4 images

2 horizontal

2 vertical

2 vertical and 8 horizontal

4 vertical and 4 horizontal

Describe, using your own words, the aliasing problem and how to avoid it, as applied to signal processing

By Nyquist/Shannon Theorem, aliasing is caused by under-sampling, i.e., the sampling frequency is lower than half of the highest frequency in the image.

In other words, sometimes there are highly-frequent changes in an image, which is where aliasing is very likely to happen when subsampling.

To avoid it, we need to increase the sampling rate (to at least twice the highest frequency). Alternatively, some anti-aliasing algorithms/filters could be applied as well.

Given a scene sampled by a ccd sensor with minimum horizontal sampling frequency $10cm^{-1}$, what is the maximum horizontal frequency in the image that can be correctly represented?

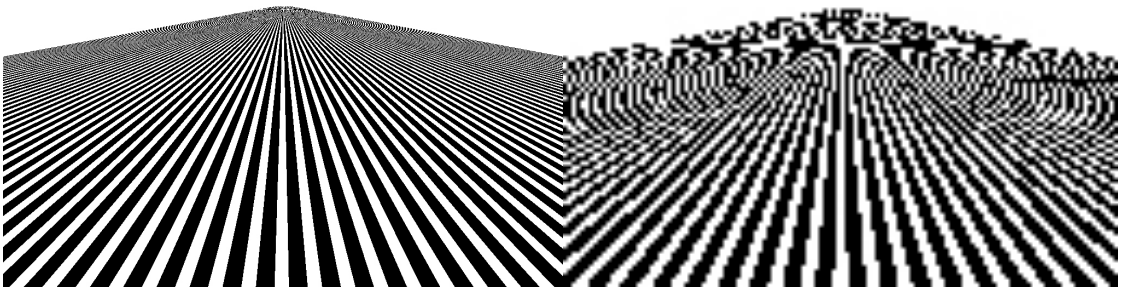
By Nyquist Theorem, $10 \div 2 = 5\ cm^{-1}$.

If you sub-sample an image, why do you have more problems from aliasing?

Because for some images which contains important details, or neighbour colours varies vastly and frequently, the subsampled image might look essentially different from the original.

(Also see the example provided in the next question)

Paste below a clear example of artefacts generated by aliasing. For this task you can use your own choice of image. Use the box below for the image and comments.



Left: 1024 x 1024 px.

Right: 128 x 128 px. As can be seen, after subsampling, aliasing problem arises with artefacts, especially at the upper part of the image. This is due to the fast change of colours for the upper pixels.

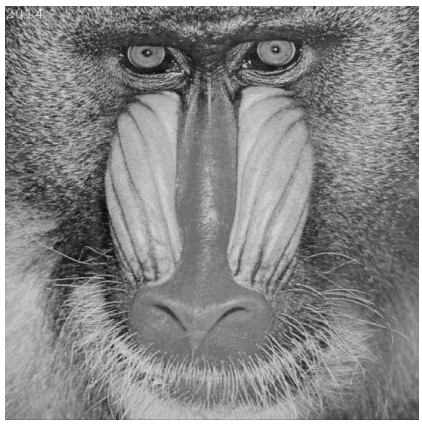
Exercise 2 (a)

Quantize: Quantization is the process of approximating the continuous values in the image data with a finite set of discrete values. The input of a quantizer is the original data and the output is one among the finite number of levels. This process is an approximation, and a good quantizer is one which represents the original signal with minimum loss (quantization error). In this lab, you will work with a scalar uniform quantizer applied to grey-scale images.

Implement a function that uniformly quantizes grey level images. The function will allow the reduction of the number of grey level values by a given factor n (a power of 2). *Note*. To visualize the image, you need to re-map it in the 8-bit-per-pixel representation. Show the results in the boxes below.

Lena, quantization factor 2

Baboon, quantization factor 8



Peppers, quantization factor 32

Peppers, quantization factor 128

Is quantization a reversible process? Can you recover what you discarded? Briefly explain.

No, quantisation is not reversible, i.e., the discarded cannot be recovered.

Because information is lost when the value of the pixel is approximated to the quantisation levels.

Write the results back to PGM/PPM files using the function you created. Make sure that your writing function allocates the correct number of bits per pixel. What is the size of the files compared with the original? Given the results, what is a typical application field for quantization? Include in your submission the output files and comment on the results.

The size remains the same as the original. This is because pgm/ppm files are completely lossless and uncompressed, where each value is stored in ASCII / Binary format with fixed length of bits.

However, for JPG files the size reduction by quantising is obvious, which is because JPG is a compression method.

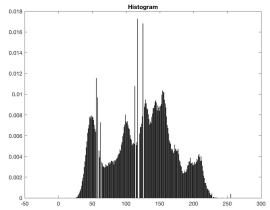
Therefore, quatisation can be applied to compress an image.

Exercise 2 (b)

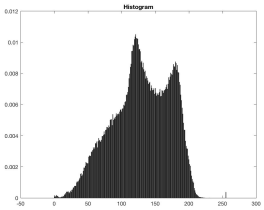
Histograms: This part of the lab is dedicated to image processing using histograms. A histogram is a statistical representation of the data within an image. The histogram can be represented as a plot of the frequency of occurrence of each grey level. This representation shows the distribution of the image data values. By manipulating a histogram, it is possible to improve the contrast in an image and the overall brightness or to segment different areas of the image by applying one or more thresholds to the histogram itself.

Implement a function to output the histogram values of a given grey level image. Display in the boxes the resulting histograms.

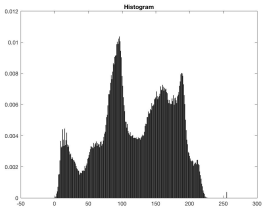
Lena



Baboon



Peppers



If you normalize the values of the histogram so that they sum to 1, what does the value of a bin represent?

The value (<1) of the bin represents the proportion of pixel at this particular greyscale level with respect to the whole image.

e.g., if the bin on 200 gives a value of 0.008, this means that the pixels at greyscale level 200 constitute 0.8% of the whole image.

The sum of all bins is 1.

Exercise 2 (c)

Equalize: Equalization is one of the possible image processing algorithms implemented using histograms. Histogram equalization allows a user to enhance the contrast of images. Histogram equalization employs a monotonic, non-linear mapping which re-assigns the intensity values of pixels in the input image such that the output image contains a uniform distribution of intensities (i.e. a flat histogram).

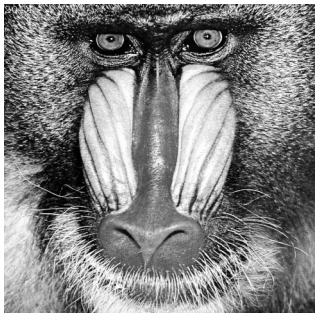
Implement a function that equalizes grey-scale images based on their histogram. The input is a given grey level image; the output is the derived image with uniform intensity distribution.

Display in the boxes the equalized images and their histograms.

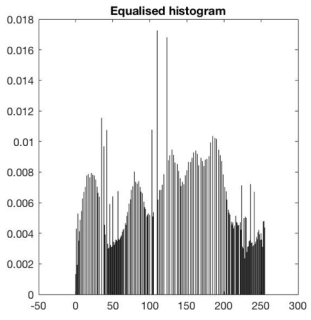
Lena equalized

Peppers equalized

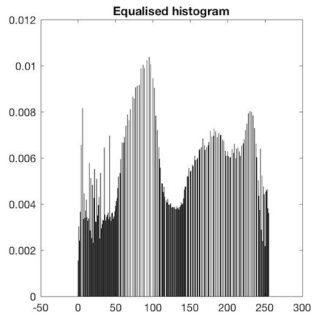
Baboon equalized



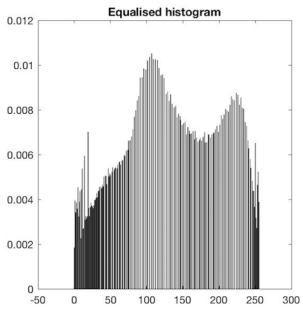
Lena histogram after equalization



Peppers histogram after equalization



Baboon histogram after equalization



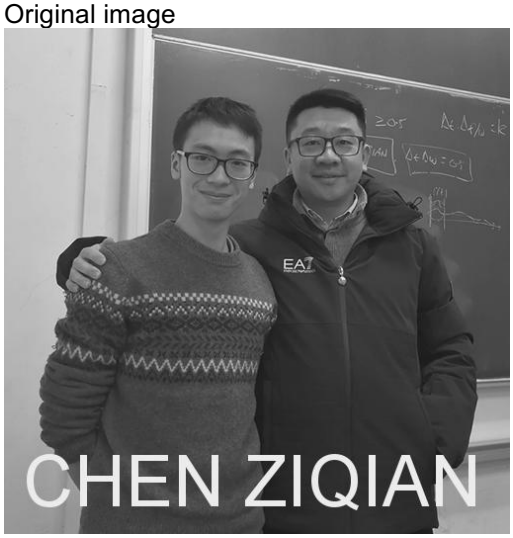
Are the distributions really uniform? Explain your results.

Not really. Equalisation adopts the gamma transformation to redistribute the histogram without changing the number of grey levels. The equalised distribution is the result of conversion through the gamma function, so it is not necessarily strictly uniform.

What is constant is the sum of all values from all bins, according to the conservation of energy.

Show an example of the successful application of histogram equalization to image enhancement. You can use an appropriate image of your choice

Original image



Enhanced image

Comment on the results of the previous step.

Equalisation is effective in raising the quality of low-contrast images, because HVS is practically more sensitive to high contrast.

Exercise 2 (d)

Histogram modelling: Histogram modelling techniques are effective tools for modifying the dynamic range and contrast of an image. Unlike contrast stretching, histogram modelling operators may employ non-linear and non-monotonic transfer functions to map between pixel intensity values in the input and output images. In the first part of this lab you will model the histogram of a grey-scale image.

Implement a point to point operation that maps the input grey level image into an output image which has a predefined frequency distribution. The algorithm is not given explicitly in the lecture slides, you are supposed to derive it. Use as input histogram the histogram of an image A and model the histogram of another image B according to the input.

(A = Lena) (B = Peppers)

Image A

Histo of A

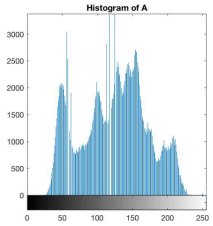
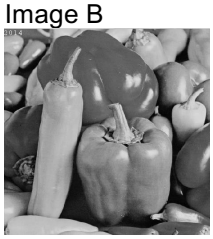
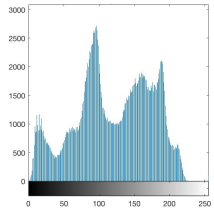


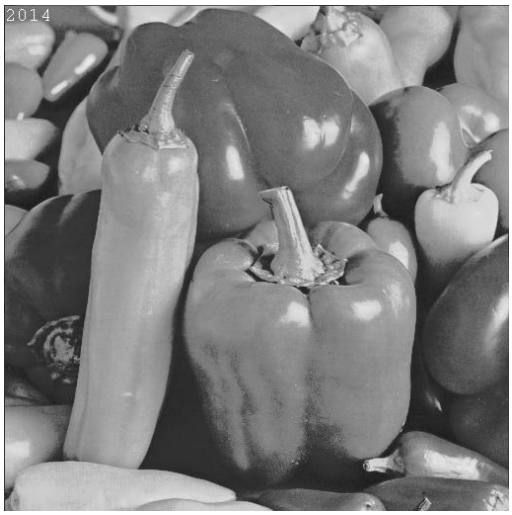
Image B



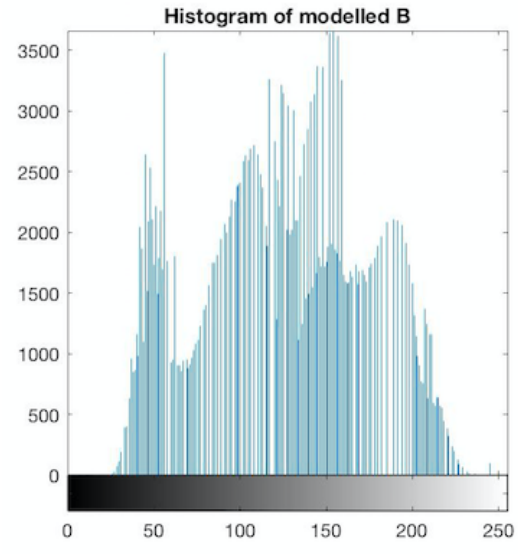
Histo of B



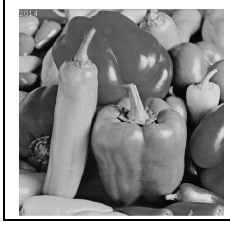
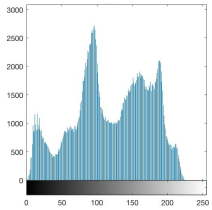
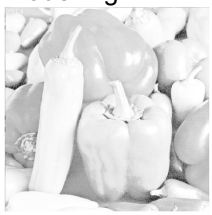
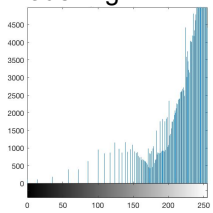
B after Modelling



Histo of B after modelling



Use as input histogram an approximation of the exponential distribution.

Image Peppers	Histo of Peppers	Peppers after modelling	Histo after modelling
			

Write in the box the formulation of your algorithm.

```
for i=-255:0
    e(i+256)=1.6^(i/16);
end;
```

Exercise 3 (a)

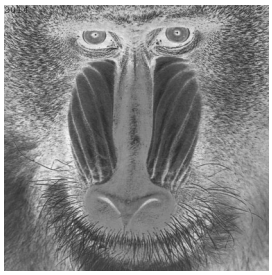
Negatives: We are used to the negative of an image in analogue image processing. It is possible to generate a negative from a digital image too. In the last part of today's lab you will solve a simple exercise on negative images.

Write a function that inverts the grey level of a PGM image (i.e. it creates the negative of the image).

Negative of Lina

Negative of Peppers

Negative of Baboon



Perform the same task with PPM images and comment on the results.

Negative of Lina

Negative of Peppers

Negative of Baboon

Your comments:

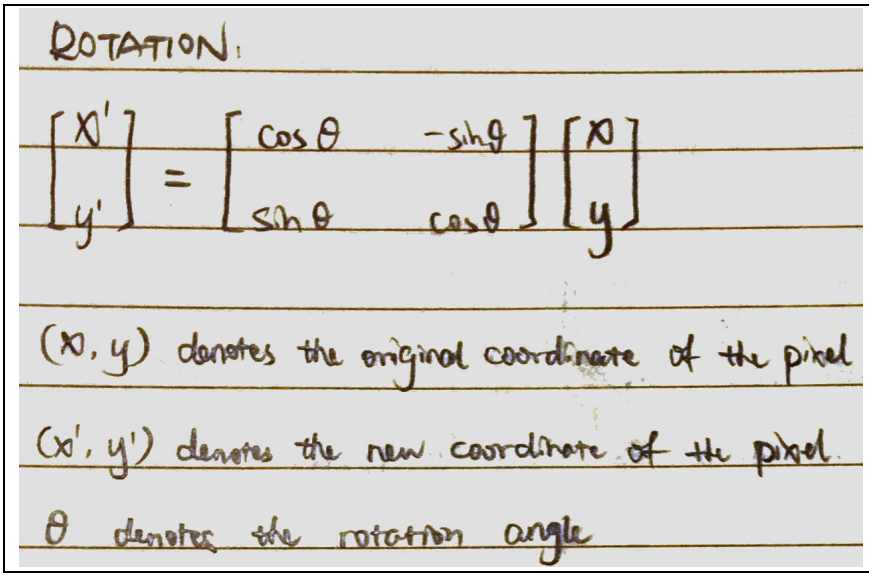
The negative inversion is basically its original pixel value subtracted from the maximum colour level (255 in this case).

Exercise 3 (b)

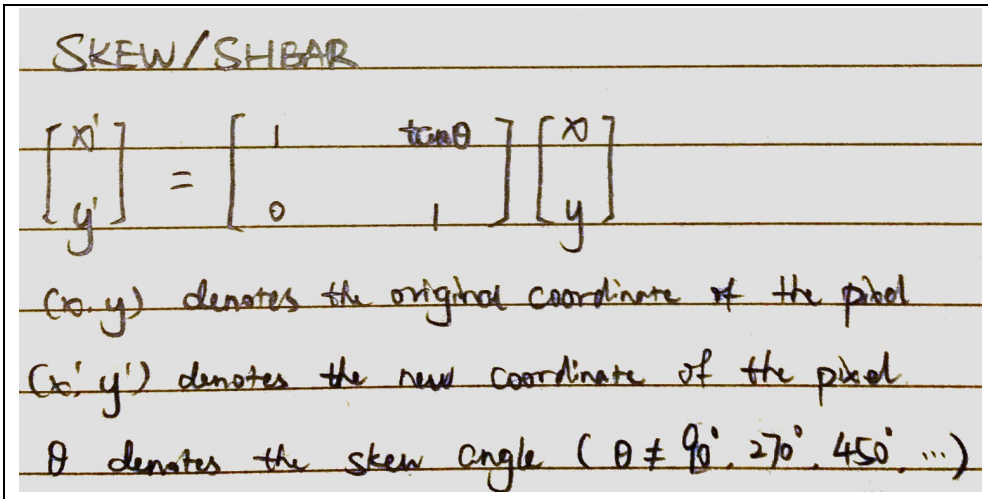
Rotation and translation. Image processing toolboxes allow a user to rotate, translate and skew images. These are very useful operations for image composition, for example. The first exercise will cover the implementation of two such transformations.

Write a function *BUPT_transform* that takes as input an image *I*, rotates it with an angle θ_1 and skews it with a second angle, θ_2.

Write the matrix formulation for image rotation (define all variables).



Write the matrix formulation for image skewing (define all variables).



Create and paste below a PGM image containing your name written in Arial font, point 72, uppercase letters.

Your image

Rotate the image you created by 30, 60 120 and -50 degrees clockwise and display the results below.

30

60

120

-50

Skew the same image by 10, 40 and 60 degrees and display the results below.

10

40

60

During the development process have you experienced the problem of regular patterns of black pixels in the image? If so, explain how you solved the problem. Otherwise imagine what could have generated these artefacts and how you would have worked around them.

No. although I did not experience the problem, I suppose this is due to the rounding issues.

To be specific, because the transformation is operated on the coordinates of the image, it is very likely that the outcome number is not an integer. In this case, unless rounding functions like round(), floor() or ceil() are used, the pixel whose position/coordinate is non-integer will be black.

Rotate the image by 20 degrees clockwise and then skew the result by 50 degrees. Display the result in 'a'.

Skew the image by 50 degrees and then rotate the result by 20 degrees clockwise. Display the result in 'b'.

a

b

Analyse the results when you change the order of the two operators i.e. R(S(*I*)) and S(R(*I*)), where R is the rotation and S is the skew. Are the results of (a) and (b) the same? Why?

No, they are not the same. Because the transformations in essence are multiplication of matrices, and matrix multiplication is not commutative. Thus, the order of transformation matters.

Exercise 4 (a)

Noise and PSNR: This part of the lab introduces error metrics for image quality evaluation. Two common metrics are the Mean Square Error (MSE) and the Peak Signal to Noise Ratio (PSNR). The MSE is the cumulative squared error between the processed image and the original image. The PSNR makes use of the MSE. The smaller the MSE, the smaller the error is. The larger the PSNR, the smaller the error is. You will add different amounts of random noise to the test images and measure their MSE and PSNR.

Write the formulas of MSE and PSNR.

$$MSE = \frac{1}{NM}\sum\left[I(x,y) - I'(x,y)\right]^2$$

I denotes the value matrix of the original image

I' denotes the value matrix of the new image

[M, N] denotes the height and the width of the image (in pixel)

$$PSNR = 20\log_{10}\left(\frac{255}{\sqrt{MSE}}\right)$$

Can the PSNR return a negative value? Explain your answer.

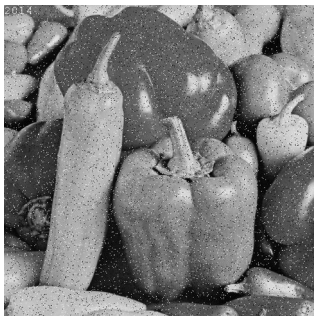
No (on condition that the colour level is 255 of course). If PSNR were smaller than 0, it means the square root of MSE would be greater than 255.

However, for each pixel, the difference between I(x,y) and I'(x,y) can never be greater than 255, thus ensuring the value of MSE is less than 255^2. Therefore, the value of PSNR cannot be negative.

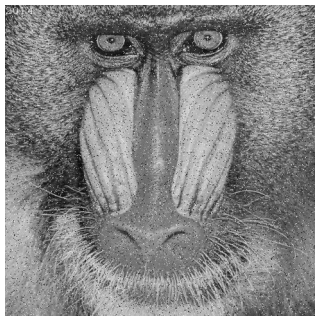
Create a function that can add (i) salt and pepper noise and (ii) Gaussian noise to a PGM image and compute PSNR and MSE. Show the results in the box and write under the box the values of MSE and PSNR comparing the original with the corrupted one.

Lena salt and pepper noise

Peppers salt and pepper noise



Baboon salt and pepper noise

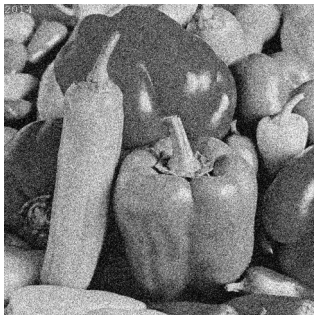


Lena Gaussian noise

σ =1% of the range.

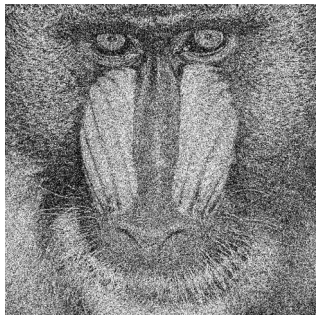
Peppers Gaussian noise

σ =2% of the range.



Baboon Gaussian noise

σ =7% of the range.

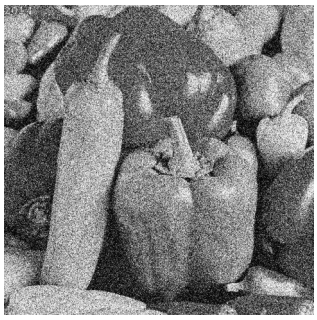


Lena Gaussian noise

σ =5% of the range.

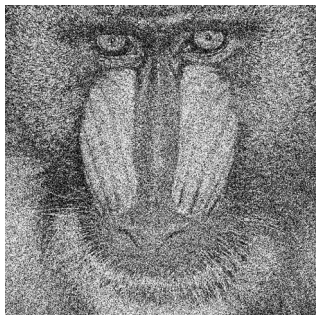
Peppers Gaussian noise

σ =5% of the range.



Baboon Gaussian noise

σ =10% of the range.

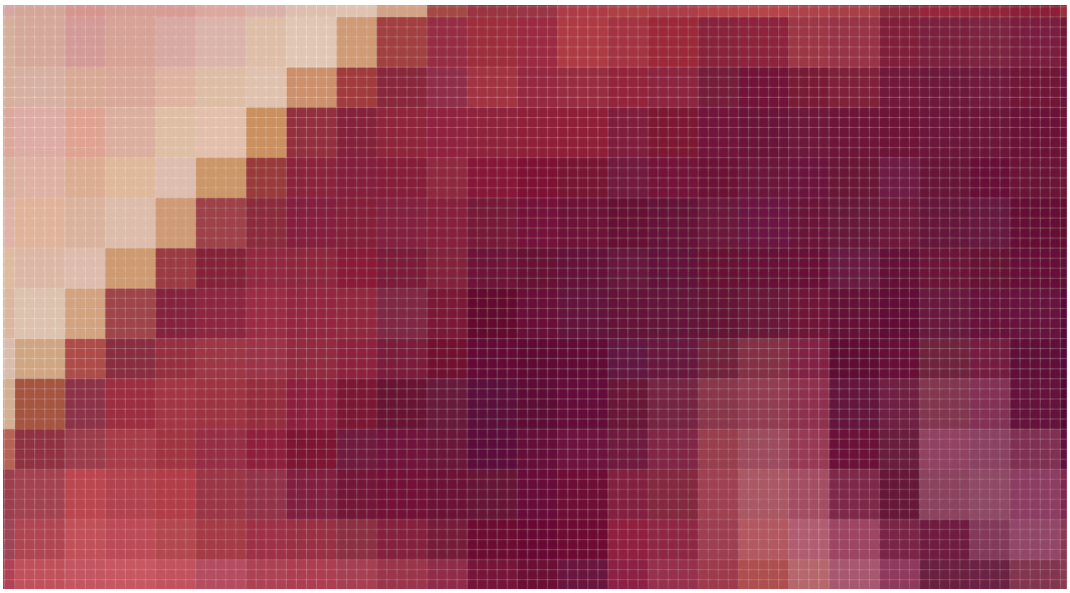


Exercise 4(b)

Up-sampling: Scaling-up an image (up-sampling) requires the filling of the new positions given the original pixels. This filling can be obtained by interpolation. Different interpolation techniques can be used. The choice depends on the quality we want to achieve and on the computation resources we have available. The nearest-neighbour interpolation is the simplest and fastest technique, but it is also a technique achieving low quality results. Bilinear interpolation is computationally more intensive, but it achieves higher quality results.

Implement the function *BUPT_up* that increases the resolution of images by a given factor (also a non-integer one). The up-sampling should be achieved using the nearest neighbour as well as the bilinear interpolation. The function will be able to up-sample independently in the horizontal and in the vertical direction or in both directions simultaneously.

Up-sample the image Lena using nearest neighbour interpolation. Display a blow-up of the image Lena obtained by up-sampling the original image with factor 4.5. The image should clearly show the type of artefact obtained using the nearest neighbour interpolation. Use the box below to display the image and discuss the results.

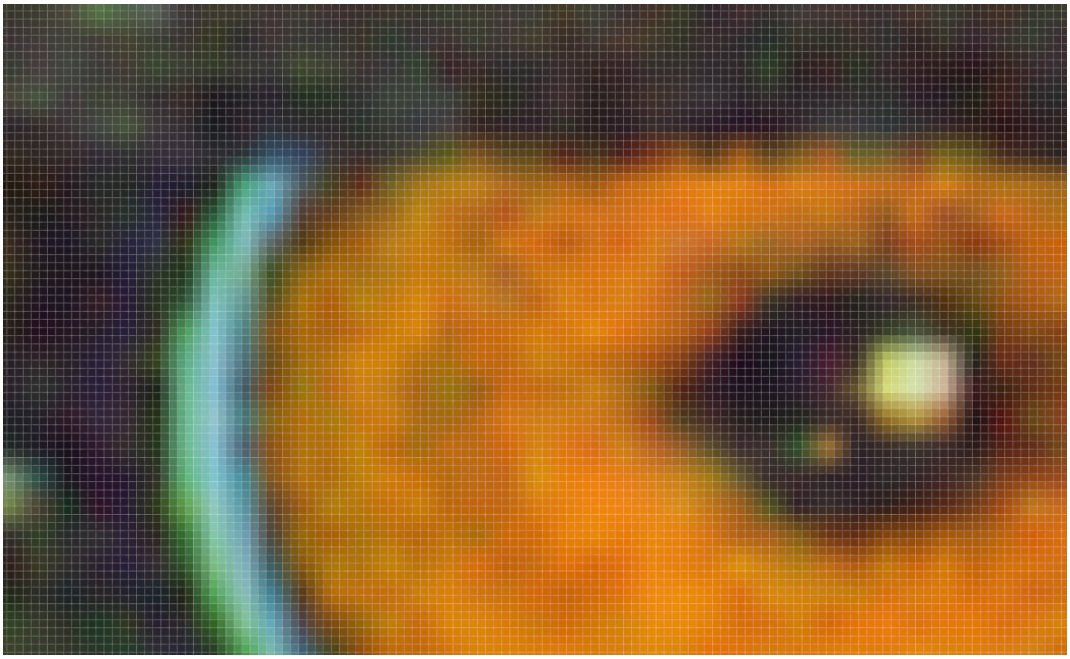


(zoomed)

Your comments:

The algorithm of Nearest Neighbour assigns the value of new pixels to be equal to one of its 4 neighbours (the nearest one), which is why, as can be seen in the zoomed image, the up-sampled result is constituted of blocks of pixels that share the same colour.

Up-sample the image Baboon using bilinear interpolation. Paste below a zoomed portion of the image Baboon obtained by up-sampling the original image with a factor 3.6. Discuss the artefacts obtained using bilinear interpolation.



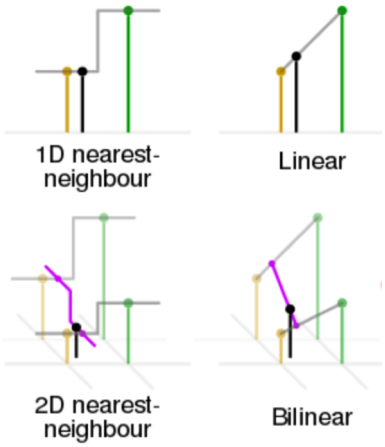
(zoomed)

Your comments:

The algorithm of Bilinear interpolation performs linear interpolation on one axis, then again on the other, thus making the image look very smooth. Unlike in NN interpolation some sawtooth shaped might be observed especially at high-frequency changes, Bilinear gives a soft gradient transition all through the image.

Compare the nearest neighbour technique and the bilinear technique in terms of speed and accuracy. Which technique is faster? Why? What artefacts are more visually disruptive?

Nearest Neighbour is faster.



As illustrated in the picture above (from Wikipedia), for each pixel, NN only needs to approximate to the one of the 4 surrounding pixels, whereas bilinear will have to calculate a smooth function from the 4 surrounding pixels, thus consuming a lot more time.

Artefacts around the edges are more visually disruptive. Using NN method might cause a sawtooth shape. Using Bilinear method tend to smooth out the edges.

Exercise 5(a)

Low pass filtering: Image filtering generates a processed image as a result of certain operations on the pixels of the original image. Each pixel in the output image is computed as a function of one or several pixels in the original image, usually located near the output pixel. The procedure is usually implemented by convolving a kernel with desired properties with the pixels of the input image. If the kernel is a Gaussian kernel, then the behaviour of the filter depends on the variance of the Gaussian.

Write a function BUPT_lowpass that convolves an image with a Gaussian kernel.

(i) Write the formula of the kernel you used.

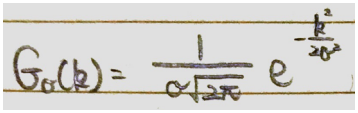
(ii) The 2D Gaussian kernel is separable: write the two separate equations for the rows and the columns, and discuss the advantages of using separable filters.

(iii) What is the relationship between σ and the cut-off frequency of the filter?

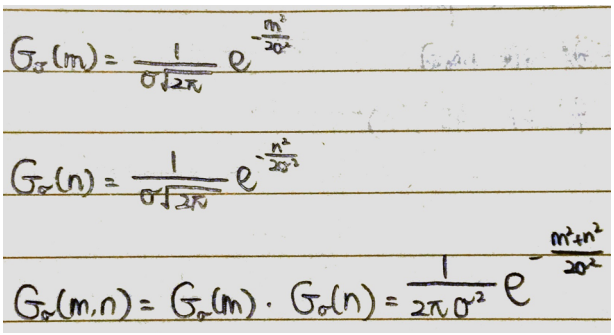
(iv) Given σ, what criterion should be used to choose the size of the kernel? Why?

Your comments:

(i)



(ii)



By separating the calculation for rows and columns, 1) the 2D complexity can be brought down to 1D computation, thus increasing the efficiency, and 2) it is available for us to adjust two parameters individually, which is more adaptable in practice.

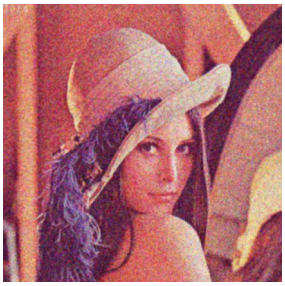
(iii) The cut-off frequency indicates how wide the filter will affect the image. It can also be denoted by the 'radius' of the filter, which is a term usually seen in Photoshop or other image processing applications. Mathematically speaking, these factors are exactly decided by σ, which denotes the variance of the Gaussian function. The larger σ is, the wider the 'radius' will be, i.e., the further the cut-off frequency will get.

(iv) The kernel size decides how many pixels should be involved in the calculation for each new value. Hence, if the σ is fairly large, which means the image will be highly smoothed, we should set the kernel size relatively small so as to keep more detail information. Vice versa, if σ is small, a bigger kernel size should be assigned in order to alleviate the noise.

Add Gaussian noise to the image Lena with noise power 50, then convolve the noisy image with Gaussian kernels with σ = 0.5, 1, 2, 4, 7, 10, respectively. Paste below the resulting images. Comment the results obtained with increasing values of σ.

σ=0.5

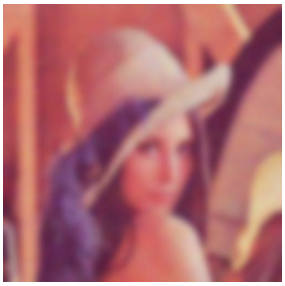
σ=1



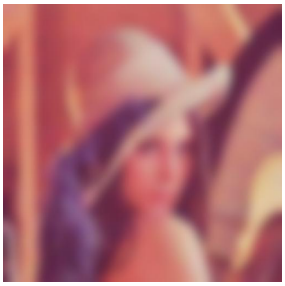
σ=2

σ=4

σ=7



σ=10



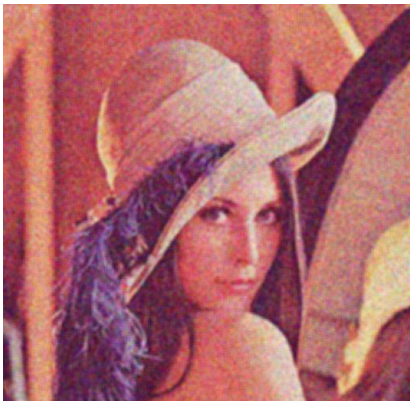
Your comments:

As can be clearly observed, the image gets increasingly smooth and blurry with the rise of σ.

These pictures above tell us that it is a trade-off between the de-noising and keeping details. In order to remove the noise, we raise σ to broaden the filter radius, but this will result a phenomenon of over-smoothing, which blurs the whole image.

Implement a rectangular-shaped filter (*BUPT_rect*). Filter the noisy image Lena with a 5-by-5 and with a 7-by-7 kernel. Paste below the resulting images. Compare these results with those obtained with the Gaussian filter.

5-by-5 pixel



7-by-7 pixel

Your comments:

As can be seen, the 7-by-7 kernel gives a smoother result than the 5-by-5 one, because of a larger size which enables more pixels to be involved in the calculation.

In comparison with the Gaussian ones, these two images appear relatively more moderately de-noised, while keeping a number of details.

This is probably because the rectangular filter is a bi-directional operator, which smoothens the pixels only horizontally and vertically. But Gaussian filter is isotropic, which integrates the smoothing effect equally from all directions.

Exercise 5(b)

Edge Detection: Edge detection is the process of identifying and locating discontinuities in an image. The discontinuities are sharp changes in pixel intensity which characterise object boundaries. Classical edge detectors convolve the image with a 2-D kernel designed to be sensitive to large gradient amplitudes. There exist a large number of edge detectors, each designed to be sensitive to certain types of edges.

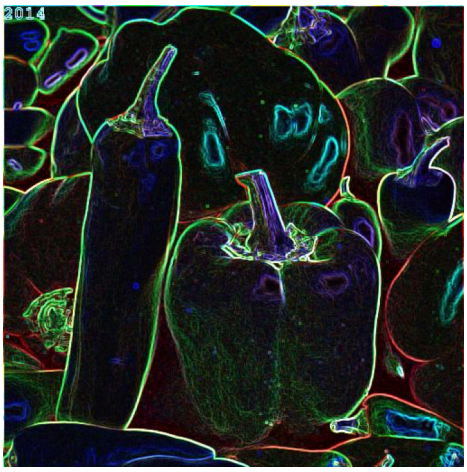
representing the absolute value of the gradient for the three filters.
Comment on how you dealt with the borders.

Sobel Lena (grey)

Sobel Peppers (colour)

Prewitt Lena (grey)

Prewitt Peppers (colour)



Roberts Lena (grey)

Roberts Peppers (colour)

Your comments:

When the operator window comes to the border of the image, some of the values cannot be calculated and a column/row of pixels needs to be padded outside the original.

In this exercise I choose to enlarge the image by assigning the border values equal to their nearest pixel. To be more specific, this is achieved by the function of 'imfilter' with an argument 'replicate'.

Exercise 6

LoG. Laplacian filters are derivative filters used to find edges in images. Since derivative filters are very sensitive to noise, it is common to smooth the image before applying the Laplacian. For example, the image can be smoothed using a Gaussian filter. The two-step process involving Gaussian low-pass filtering followed by Laplacian filtering is called the Laplacian of Gaussian (LoG) operator and will be covered in the first part of the lab.

Implement the LoG operator as a parametric function of the variance, and display the results in the boxes.

Lena σ = 1

Peppers σ = 3

Try the effect of LoG filters using different Gaussian widths by changing the variances. What is the general effect after increasing the Gaussian width?

Baboon σ = 1

Baboon σ = 2

Baboon σ = 3

As the width increases with σ, the image gets more blurred and thus loses more fine edges, which is why the resulted edge map appears more poorly-connected, with more false-negatives.

Construct a LoG filter where the mask size is too small for the chosen Gaussian width (*i.e.* the LoG becomes truncated). What is the effect on the output? Define an empirical or analytical rule to determine how large a LoG mask should be in relation to the variance of the underlying Gaussian if severe truncation is to be avoided.

Lena result

σ = 5

Kernel size = 20 x 20

Peppers result

σ = 4

Kernel size = 15 x 15

Discussion

When the mask size is too small, the edge map will seem nearly empty, because the edge is too smooth to be detected by a small sized window.

Empirically from numerous trials in this exercise, I discovered that when the kernel (suppose it's a square) size reached more than k times the value of σ, the edge map reaches saturation and is able to present all edges it can show.

But this multiplier of k could be different for different images. In here for Lena the empirical number is about 6 and for Peppers it's around 5. But no matter how, the kernel size should be no less than this fixed k times of σ.