EBU6230 – Image and Video Processing – 2022/23

Coursework report and exercises

Name: Zhengxiao Wu

Username: jp2020213235

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 EBU6230 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. Open the file containing the image using a file I/O function.
2. Read the image header to determine the image format, width, height, and maximum pixel value.
3. Allocate memory for the image data based on the width and height.
4. Read the pixel data from the file and store it in the allocated memory.
5. Close the file.

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

1. Open a file for writing using a file I/O function.
2. Write the image header to the file, including the image format, width, height, and maximum pixel value.
3. Write the pixel data to the file in the appropriate format.
4. Close the file.
5. Free the memory allocated for the image data.

What is the difference between the identifiers P3 and P6?

The difference between the identifiers P3 and P6 is that P3 indicates a PPM image in ASCII format, where each pixel value is represented by a decimal number in the range 0-255, separated by whitespace. P6 indicates a PPM image in binary format, where each pixel value is represented by a single byte in the range 0-255. PGM images use the same identifier format, but represent grayscale images instead of color images.

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?

In the YUV color space, the Y component represents the luminance or brightness of an image. It is the component that is used to represent the grey-levels of an image. The U and V components represent the chrominance or color information of the image.

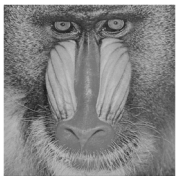
Use thee boxes to display the results for the colour to grey-scale conversion.

Lena colour (RGB)

Lena grey

Baboon colour (RGB)

Baboon grey



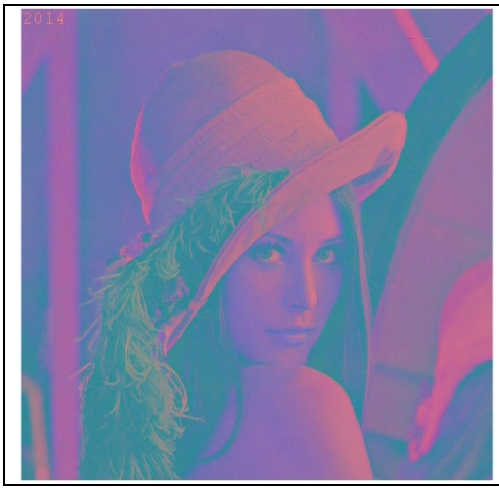
Is the transformation between the two colour-spaces linear? Explain your answer.

The transformation between the RGB and YUV color spaces is linear.

A linear transformation preserves addition and scalar multiplication. In the case of the YUV conversion, the transformation can be expressed as a matrix multiplication followed by an addition of an offset vector.=

The transformation satisfies the properties of linearity because it preserves addition and scalar multiplication. Therefore, the transformation between the RGB and YUV color spaces can be considered a linear transformation.

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



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

When converting the Lena image to the YUV color space, the resulting image will have three channels: Y, U, and V. The Y channel represents the luminance or grey-levels of the image, while the U and V channels represent the chrominance or color information.

If display the YUV image directly without any further processing or conversion back to the RGB color space, the colors will appear distorted. This is because the YUV color space separates the luminance and chrominance information, and displaying the U and V channels without considering the proper color decoding can result in a distorted representation.

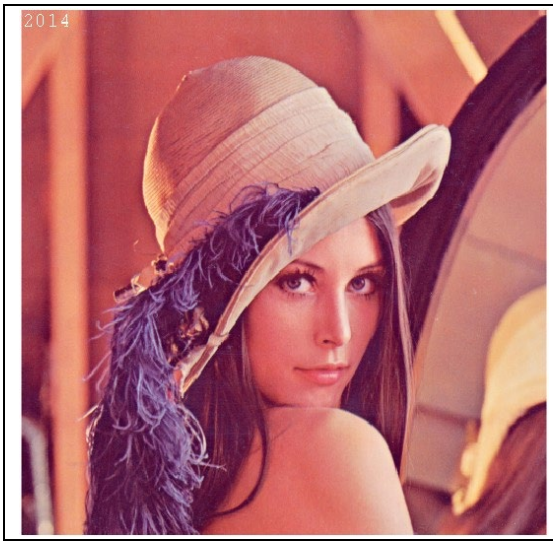
Correctly visualize the colors needs to convert the YUV image back to the RGB color space by applying the inverse transformation.

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

$$M_1 = \begin{bmatrix} 0.257 & 0.504 & 0.098 & 16 \\ -0.148 & -0.291 & 0.439 & 128 \\ 0.439 & -0.368 & -0.071 & 128 \end{bmatrix}$$

$$M_2 = M_1^{-1} = \begin{bmatrix} 1.164 & 0 & 1.596 & -222.912 \\ 1.164 & -0.391 & -0.813 & 135.488 \\ 1.164 & 2.018 & 0 & -276.928 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

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

Aliasing is a phenomenon that occurs in signal processing when the high-frequency components of a signal are incorrectly represented or distorted. It happens when the sampling rate used to capture or process a signal is insufficient to accurately capture the details or high-frequency information present in the signal.

In the context of image processing, aliasing can manifest as unwanted visual artifacts such as jagged edges, moiré patterns, or distortion in the image. This is especially noticeable when sub-sampling or resizing an image, where the image resolution is reduced.

To avoid aliasing in signal processing, particularly in image sub-sampling, the process of anti-aliasing is employed. Anti-aliasing techniques aim to reduce or eliminate the effects of aliasing by taking into account the signal's frequency content and applying appropriate filtering methods.

We can apply a low-pass filter before down-sampling the image. The low-pass filter removes or reduces the high-frequency components of the signal, smoothing out the image and preventing high-frequency details from being incorrectly represented. This filtering process helps to avoid the creation of aliasing artifacts during the sub-sampling operation.

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?

According to the Nyquist theorem, the maximum frequency that can be correctly represented in the image is half of the sampling frequency, which is 5 cm-1.

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

The sub-sampling process involves reducing the spatial resolution of the image. As a result, high-frequency details and fine spatial information are lost or not adequately captured in the sub-sampled image.

Aliasing in images occurs when the sampling rate is insufficient to accurately represent the details and high-frequency content of the original image. High-frequency components that exceed the Nyquist frequency (half the sampling frequency) can fold back and appear as lower-frequency components, creating distortions and artifacts in the sub-sampled image.

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.

Original image

4 vertical and 4 horizontal

After performing 1/4 sub-sampling in both the horizontal and vertical directions on the original image of size 950*1300, we obtained a sub-sampled image of size 237*325. The resulting image shows prominent aliasing artifacts along the edges of the window frames, with noticeable jagged lines. Additionally, high-frequency details such as leaves and petals have been significantly lost. The overall image quality has degraded significantly.

As a result, the overall image quality has noticeably deteriorated.

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

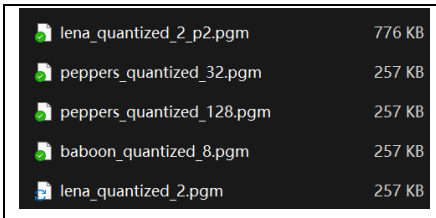
Peppers, quantization factor 32

Peppers, quantization factor 128

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

Quantization is not a reversible process. When we quantize an image by reducing the number of grey level values, we discard some of the original information. The discarded information cannot be recovered exactly because it has been compressed or approximated into a smaller set of discrete values. The quantization process introduces irreversible distortions and loss of information.

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.



For P2 and P3, the quantized image size may be lower (fewer digits) due to the variable length of the values expressed using ASCII;

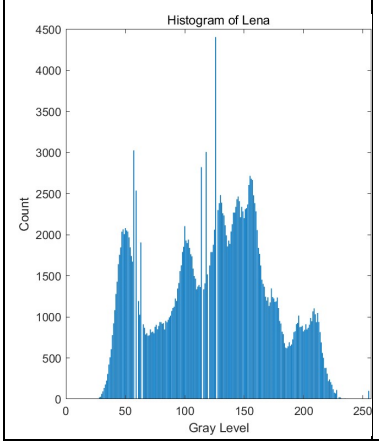
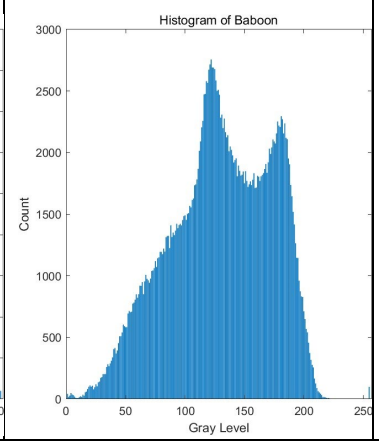
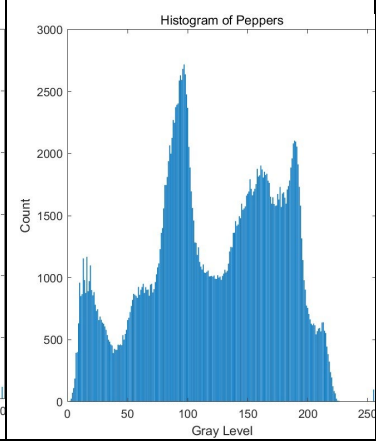
For P5 and P6, the quantized image size is unchanged since both versions of this implementation use the uint8 type to store individual values, each pixel being 1byte (grayscale) and 3byte (tricolor).

If one P5 and P6 implementation uses variable length bits to store the values (e.g. uint4, uint16, uint32, etc.), the image size may become 1/log2(factor) of the original.

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?

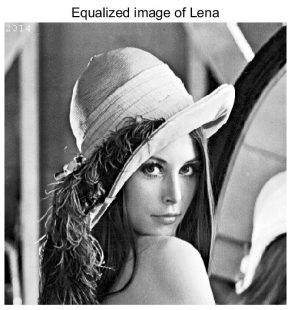
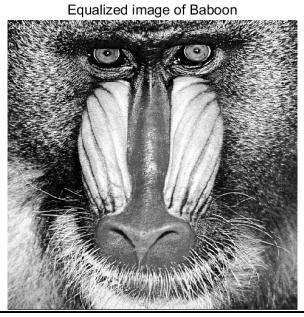
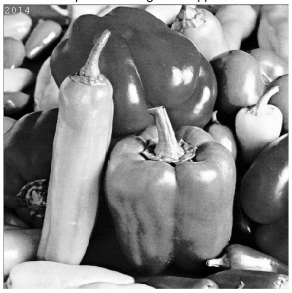
If the values of the histogram are normalized so that they sum to 1, the value of a bin represents the probability density of the corresponding grey level in the image. In other words, each bin represents the relative frequency of occurrence of a particular grey level in the image, taking into account the total number of pixels.

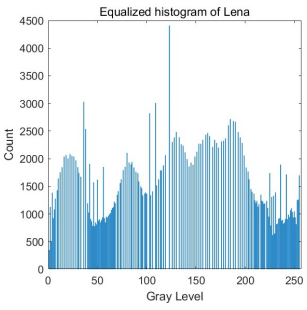
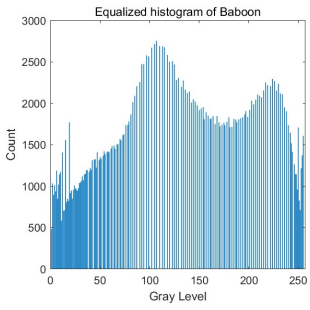
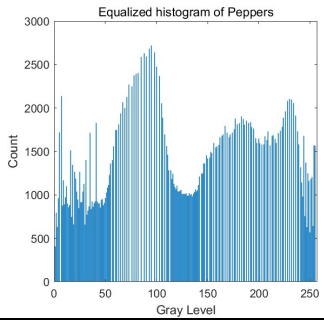
Exercise 2 (c)

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	Baboon equalized	Peppers equalized
		

Lena histogram after equalization	Baboon equalized after equalization	Peppers equalized after equalization
		

Are the distributions really uniform? Explain your results.

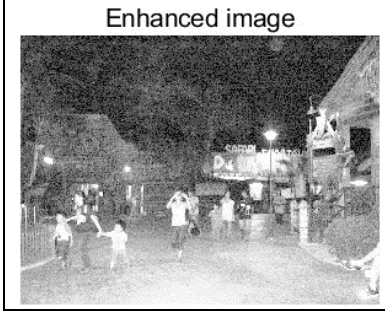
The distributions achieved through histogram equalization may not be perfectly uniform, even though the goal is to achieve a more uniform intensity distribution. This is because histogram equalization redistributes the pixel intensities in such a way that it maximizes the overall contrast of the image. The resulting histogram will appear flatter and more spread out compared to the original histogram, indicating a wider range of intensities. However, this does not guarantee a completely uniform distribution.

The reason for not achieving a perfectly uniform distribution is that histogram equalization operates based on the cumulative distribution function (CDF) of the histogram. The CDF maps the input intensities to output intensities in a way that stretches or compresses the intensity range. It aims to balance out the pixel intensities across the entire range, but this process can result in some intensities being overrepresented or underrepresented in the equalized image. As a result, while the equalized histogram may appear flatter, there may still be some variations and deviations from a perfectly uniform distribution.

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.

The application of histogram equalization on the original dark outdoor night scene image resulted in an enhanced image that exhibits improved brightness and visibility. The equalization process effectively redistributed the pixel intensities, spreading them across a wider range and enhancing the overall contrast. As a result, the equalized image appears brighter, making the details and objects in the scene more visible. However, it is worth noting that the process of equalization can also lead to an increase in noise within the image.

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

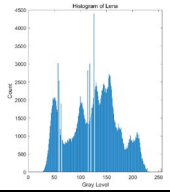
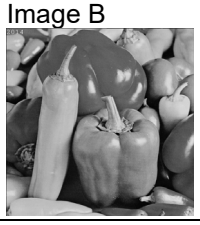
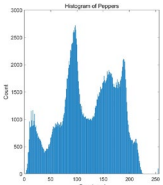


Image B

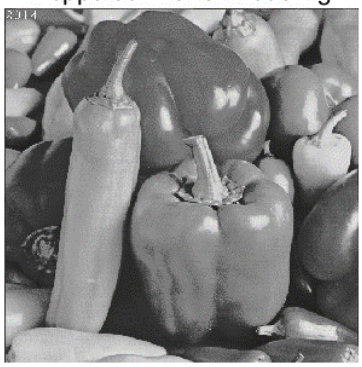


Histo of B

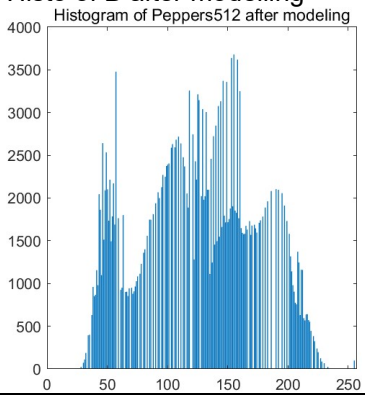


B after Modelling

Peppers512 after modeling

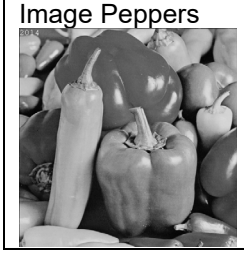


Histo of B after modelling

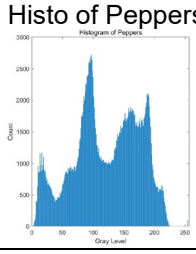


Use as input histogram an approximation of the exponential distribution.

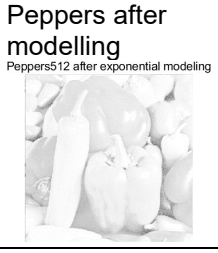
Image Peppers



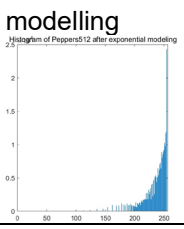
Histo of Peppers



Peppers after modelling



Histo after modelling



Write in the box the formulation of your algorithm.

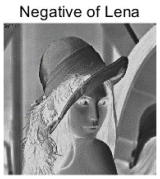
```
% Generate the approximation of the exponential distribution
decay_factor = 0.05;
exp_hist = exp(decay_factor * (1:length(hist_B)));
% Normalize the histogram from 0 to pixel number
exp_hist = exp_hist / sum(exp_hist) * sum(hist_B);

% Compute the histogram modeling
img_B_exp = BUPT_histogram_modeling_by_hist(img_B, exp_hist);
hist_B_exp = BUPT_histogram(img_B_exp);
```

Exercise 3 (a)

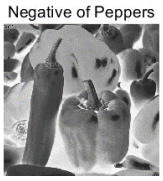
Negatives: We are used to the negative of an image in analogue image processing. It is possible to generate a negative from a digital image too. In the last part of today's lab you will solve a simple exercise on negative images.

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

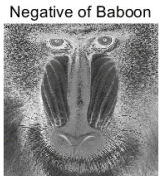
Negative of Lena



Negative of Peppers

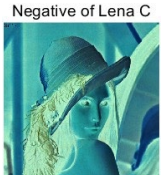


Negative of Baboon

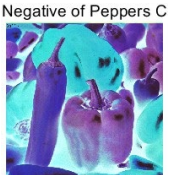


Perform the same task with PPM images and comment on the results.

Negative of Lena



Negative of Peppers



Negative of Baboon

Your comments:

Inverting the grey level of a PGM image involves converting each pixel value to its complement, resulting in a negative image. This can be achieved by subtracting each pixel value from the maximum intensity value of the image.

When performing the same task with PPM images, which are color images, inverting the grey level would require inverting each color channel individually. This means taking the complement of the red, green, and blue values of each pixel.

This leads to interesting and sometimes striking visual transformations.

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).

$$\begin{bmatrix} X_{new} \\ Y_{new} \end{bmatrix} = \begin{bmatrix} \cos\theta_1 & -\sin\theta_1 \\ \sin\theta_1 & \cos\theta_1 \end{bmatrix} \begin{bmatrix} X \\ Y \end{bmatrix}$$

θ is the rotation degree.

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

$$\begin{bmatrix} X_{new} \\ Y_{new} \end{bmatrix} = \begin{bmatrix} 1 & \frac{1}{\tan\Theta} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \end{bmatrix}$$

Θ is the skew degree.

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

Rotate 30

60

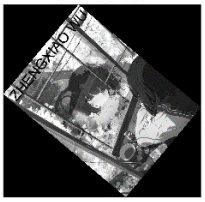
Rotate 60

120

Rotate 120

-50

Rotate -50



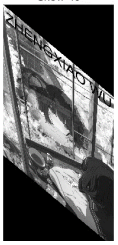
Skew the same image by 10, 40 and 60 degrees and display the results below.

10

Skew 10

40

Skew 40



60

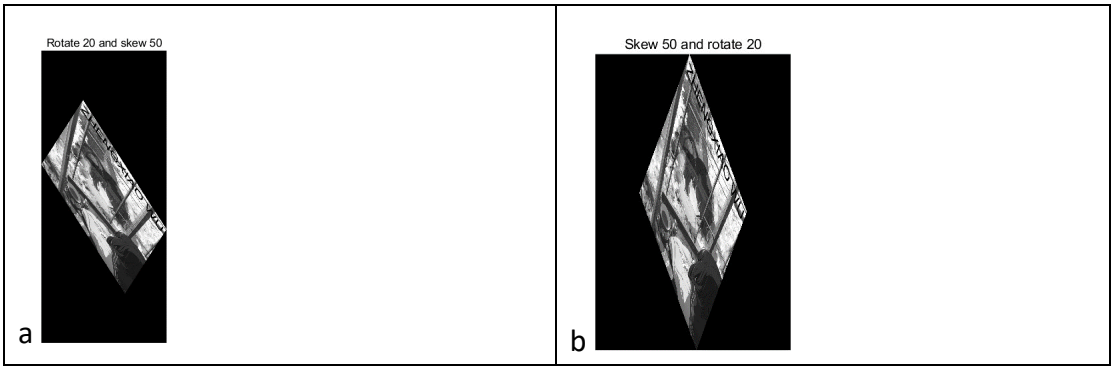
Skew 60

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.

I have not encountered this issue. My speculation is that it may occur when stretching and rotating the image, as the resulting image may have a larger display area with more pixels compared to the original image. If each pixel of the original image is directly mapped to a pixel in the new image, not all pixels in the enlarged display area will have valid values. However, in my mapping method, I transform the coordinates of the display area in the new image back to the coordinates in the original image and then perform sampling. This ensures that all pixels in the display area have valid values, avoiding this problem.

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’.



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?

The results (a) and (b) are not the same. The reason for this difference is that the order of operations affects the geometric transformation applied to the image. Rotation followed by skew applies the rotation first, which changes the orientation of the image, and then applies the skew, which distorts the rotated image. On the other hand, skew followed by rotation applies the skew first, which stretches or compresses the image along a specific axis, and then applies the rotation, which rotates the skewed image.
Since rotation and skewing are non-commutative operations, changing the order of these operations leads to different transformations and, consequently, different results.

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.

$$MSE = \left(1/(M * N)\right) * \Sigma\left[\Sigma\left[\left(I(x,y) - K(x,y)\right)^2\right]\right]$$

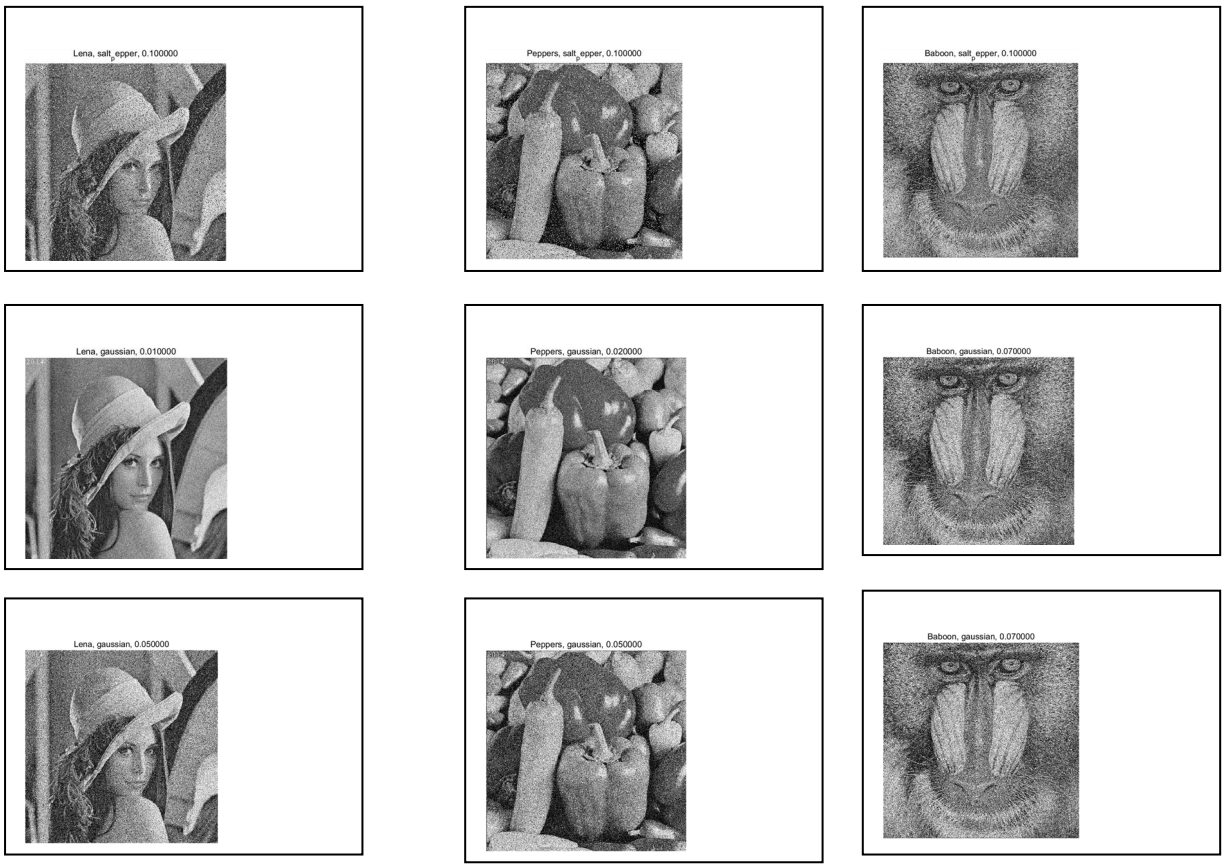
$$PSNR = 10 * log10\left((Pmax^2)/MSE\right)$$

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

No, the Peak Signal to Noise Ratio (PSNR) cannot return a negative value. PSNR is calculated using the MSE (Mean Square Error), which is always a non-negative value. Since PSNR is determined by the ratio of the maximum possible pixel value to the MSE, it will also be a non-negative value.

In the formula for PSNR: PSNR = 10 * log10((Pmax^2) / MSE), both Pmax (the maximum possible pixel value) and MSE are non-negative values. Taking the logarithm of a non-negative value will always yield a non-negative result. Therefore, the PSNR will always be a non-negative value, indicating the quality or fidelity of the processed image in relation to the original image.

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.



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.

Your comments:

The nearest neighbor interpolation algorithm can effectively increase the width and height of an image. However, due to multiple pixels in the upsampled image taking their values from the same pixel in the source image, the resulting image often exhibits jagged edges and blocky artifacts, resulting in a poor overall visual appearance.

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.

Your comments:

Bilinear interpolation is a commonly used algorithm for image resizing or upscaling, and it provides a smoother result compared to nearest neighbor interpolation. The basic principle behind bilinear interpolation is to estimate the values of new pixels by taking a weighted average of the surrounding four pixels from the original image.

Although the upsampled image obtained using bilinear interpolation may exhibit some blurring, it provides better overall visual quality compared to the nearest neighbor algorithm. Bilinear interpolation strikes a balance between preserving image details and reducing artifacts, resulting in a smoother and more natural-looking 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?

When comparing the nearest neighbor technique and the bilinear technique, the nearest neighbor technique is generally faster, while the bilinear technique offers higher accuracy and produces visually superior results.

In terms of speed, the nearest neighbor technique is faster because it simply selects the closest pixel value from the original image and assigns it to the corresponding pixel in the resized image. On the other hand, the bilinear technique involves more calculations to estimate the values of new pixels by interpolating between neighboring pixels. This additional computation makes the bilinear technique slower compared to nearest neighbor.

In terms of accuracy, the bilinear technique is superior. It takes into account the values of neighboring pixels and creates a smoother transition between them, resulting in a more realistic and visually pleasing image. Bilinear interpolation reduces the blocky artifacts and provides better preservation of details and edges.

The nearest neighbor technique, while faster, can produce visually disruptive artifacts. It often introduces noticeable aliasing effects, creating a blocky appearance in the upscaled image. The lack of smooth transitions between pixels can lead to jagged edges and loss of fine details. These artifacts can be visually disruptive, especially in images with sharp edges, diagonal lines, or intricate patterns.

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)

$$kernel(x,y) = exp\big(-(x^2 + y^2)/(2 * sigma^2)\big)/(2 * pi * sigma^2)$$

(ii)

$$rowkernel(x) = exp\big(-x^2/(2 * sigma^2)\big)/(sqrt(2 * pi) * sigma)$$

$$columnkernel(y) = exp\big(-y^2/(2 * sigma^2)\big)/(sqrt(2 * pi) * sigma)$$

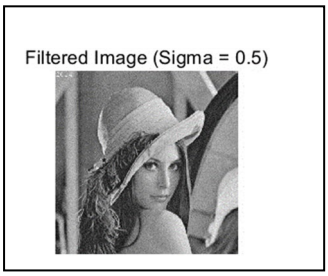
Advantages of using separable filters include:

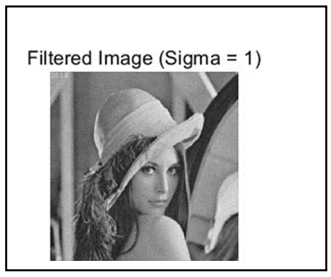
- Reduced computational complexity: Reduces the number of calculations required compared to a non-separable filter, which involves convolving the entire kernel in each direction.
- Faster execution: Since the separable filter requires fewer calculations, it generally leads to faster execution times, especially for larger kernel sizes.
- Memory savings: Separable filters require less memory to store the kernel coefficients compared to non-separable filters, as the kernel can be represented as two separate vectors instead of a 2D matrix.

(iii) The cut-off frequency of the filter is inversely proportional to the standard deviation (σ) of the Gaussian kernel. A larger value of σ results in a smoother Gaussian curve and a lower cut-off frequency, meaning that more high-frequency components are attenuated by the filter.

(iv) When choosing the size of the kernel, a common criterion is to consider the extent of the Gaussian function and its significant contributions. Typically, the size of the kernel should be determined such that the values beyond a certain distance from the center contribute negligibly to the overall result. This distance is often defined as a multiple of σ, such as 3 or 4 times σ. By choosing an appropriate kernel size based on σ, we ensure that the kernel captures sufficient information from the image while minimizing unnecessary computations.

Add Gaussian noise to the image Lena with noise power 50 dBm, 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.

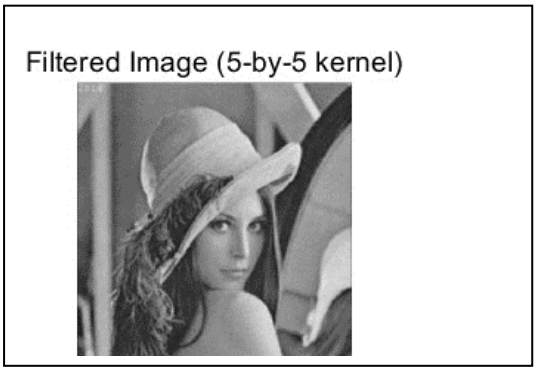


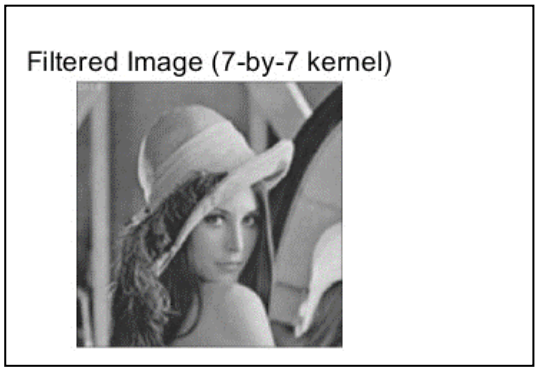


Your comments:

As the σ value increases, the resulting images become progressively smoother. Smaller σ values (0.5 and 1) preserve more of the noise characteristics, while larger σ values (2, 4, 7, and 10) exhibit higher levels of smoothing. The Gaussian filter effectively reduces the high-frequency noise components in the images, resulting in a visually cleaner appearance.

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.





Your comments:

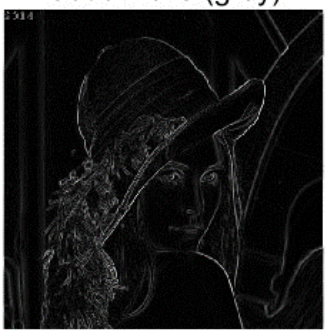
The rectangular-shaped filter, applied with a 5-by-5 and 7-by-7 kernel, produces images with visible blocky artifacts. The rectangular filter lacks the smoothing characteristics of the Gaussian filter, leading to a less visually pleasing result. Compared to the Gaussian filter, the rectangular filter preserves more of the noise and does not effectively reduce high-frequency noise components. The resulting images have sharper edges but lack the smoothness and overall quality achieved by the Gaussian filter. Using 7-by-7 kernel is between 5-by-5 and Gaussian filter.

Exercise 5(b)

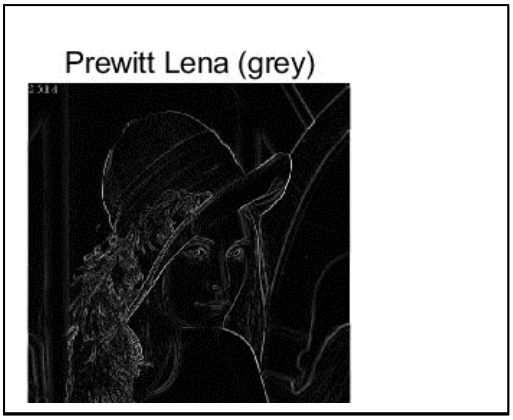
Edge Detection: Edge detection is the process of identifying and locating discontinuities in an image. The discontinuities are sharp changes in pixel intensity which characterise object boundaries. Classical edge detectors convolve the image with a 2-D kernel designed to be sensitive to large gradient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

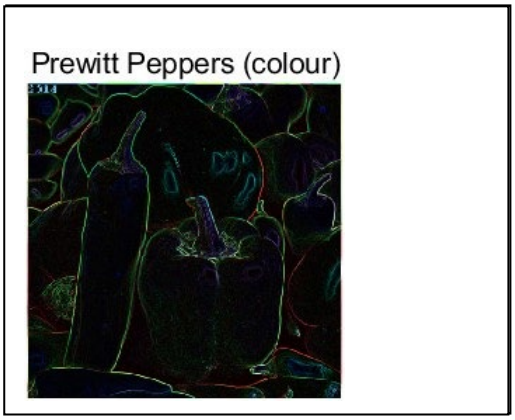
Implement the Sobel, Roberts and Prewitt filters for grey level and colour images. In the case of colour images, you can apply separate filtering of each of the three RGB components. Paste below the images representing the absolute value of the gradient for the three filters. Comment on how you dealt with the borders.

Sobel Lena (grey)



Sobel Peppers (colour)





Your comments:

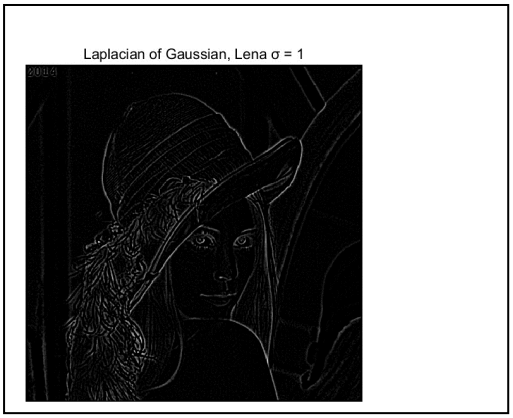
The Sobel and Prewitt filters successfully detect edges in both grey level and colour images. They highlight the edges with strong gradients, effectively capturing object boundaries. Zero-padding is commonly used to handle the borders, reducing artifacts near the image edges. When applied to colour images, the filters can be separately applied to each RGB component and combined to represent colour edges. The resulting images provide informative representations of the edges present in the images.

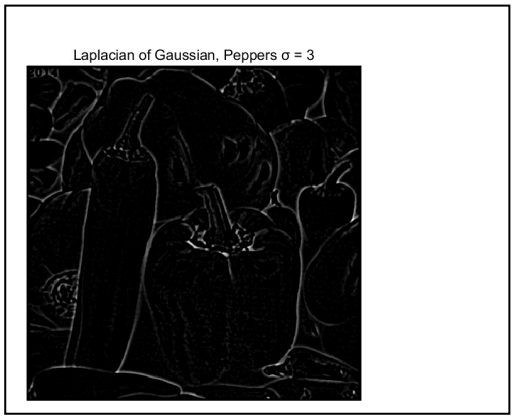
However, compared to the Sobel and Prewitt filters, the Roberts filter may produce slightly less accurate edge detection results due to its simplicity. Nonetheless, it can still provide useful information about diagonal edges in both grey level and colour images.

Exercise 6

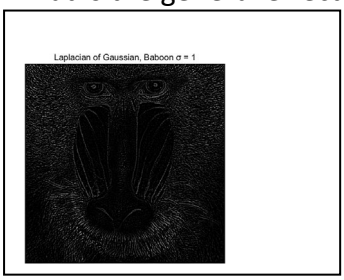
LoG. Laplacian filters are derivative filters used to find edges in images. Since derivative filters are very sensitive to noise, it is common to smooth the image before applying the Laplacian. For example, the image can be smoothed using a Gaussian filter. The two-step process involving Gaussian low-pass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

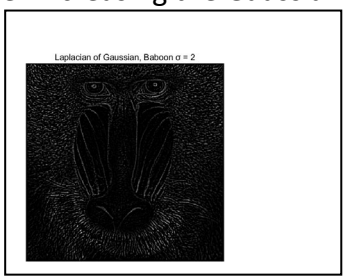
Implement the LoG operator as a parametric function of the variance, and display the results in the boxes.

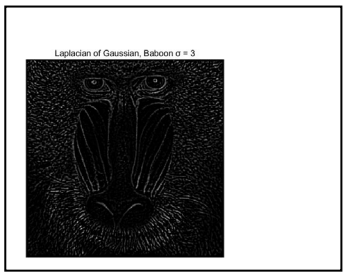




Try the effect of LoG filters using different Gaussian widths by changing the variances. What is the general effect after increasing the Gaussian width?







Add your comments here

Increasing the Gaussian width (variance) in a LoG (Laplacian of Gaussian) filter has the general effect of increasing the scale or size of the detected edges or features in the filtered image.

When the Gaussian width is increased, the filter becomes more sensitive to larger-scale changes in pixel intensity. This means that broader edges and features will be emphasized while finer details may be smoothed or suppressed. The larger the variance, the more the filter responds to gradual changes in intensity, resulting in a broader response.

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 = 3

Kernel size = 19 x 19

Peppers result

σ = 6

Kernel size = 37 x 37

ss

Discussion

When the mask size of the LoG (Laplacian of Gaussian) filter is too small for the chosen Gaussian width (variance), the LoG filter becomes truncated. The effect on the output is that the filter will not be able to capture the complete shape and details of the image features, particularly the smaller-scale details.

To determine how large a LoG mask should be in relation to the variance of the underlying Gaussian to avoid severe truncation, an empirical rule is to ensure that the mask size is at least 6 times the standard deviation (σ) of the Gaussian. This rule is based on the fact that the majority of the energy of the Gaussian distribution falls within approximately ±3σ from the mean. By using a mask size that is 6 times the σ, we can capture most of the important information in the image without significant truncation.

By following this rule, the LoG filter will have a sufficient spatial extent to adequately represent the Gaussian shape and preserve the important image details, minimizing the loss of information due to truncation.