Assignment 4
Mastering Basic Image Restoration and Compression Techniques

Please note:

1. What you must hand in includes the assignment report (.pdf) and – if necessary – source codes (.py or .m). Please zip them all together into an archive file named according to the following template: XXXXXXXX.zip
   Where XXXXXXXX must be replaced with your student ID.
2. Some problems are required to be solved *by hand* (shown by the icon), some need to be implemented (shown by the icon), and some need to be dockerized (shown by the icon).
3. As for the first type of the problems, you are free to solve them on a paper and include the picture of it in your report. Here, cleanness and readability are of high importance.
4. Your work will be evaluated mostly by the quality of your report. Don't forget to explain what you have done, and provide enough discussions when it's needed.
5. 5 points of each homework belongs to compactness, expressiveness and neatness of your report and codes.
6. By default, we assume you implement your codes in Python. If you're using Matlab, you have to use equivalent functions when it is asked to use specific Python functions.
7. Your codes must be separated for each question, and for each part. For example, you have to create a separate .py file for part b. of question 3. Please name it like p3b.py.
8. Problems with bonus points are marked by the icon.
9. If there is ***any*** question, please don't hesitate to contact us through the following email addresses:
   - Atghaei@aut.ac.ir
   - fardin.aiar@gmail.com
   - minoo.dolatabadi75@gmail.com
   - aidin.khalili@ymail.com
   - b.roshanfekr@aut.ac.com
10. Unfortunately, it is quite easy to detect copy-pasted or even structurally similar works, no matter being copied from another student or internet sources. Try to send us your own work, without being worried about the grade! ;)

1. Getting Familiar with the applications of Image Interpolation (15 Pts.)

Image Interpolation can be used in various image processing applications. In this problem, you will get hands-on experience in some of them.

a) **Image Warping** is the process of manipulating a digital image so that different parts in the image have been significantly distorted. It may be used in many purposes; from correcting image distortion to creating funny images. The goal of this part is to apply warping on the input image, "Brad.jpg". You are free to choose any warping pattern you like (apply at least two different types of warping).

b) **Image Morphing** is an image processing technique that changes one image into another through a sequence of intermediate images. It is mostly used to depict one person's image turning into another. In this part, you are to do the same on two images "Clinton.jpg" and "Cruz.jpg". Start with the image "Clinton.jpg" and display 10 intermediate images, before turning into the image "Cruz.jpg".

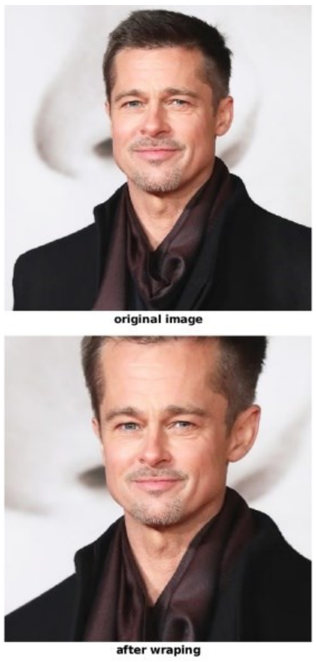


Figure 1: Input image alongside a warping example

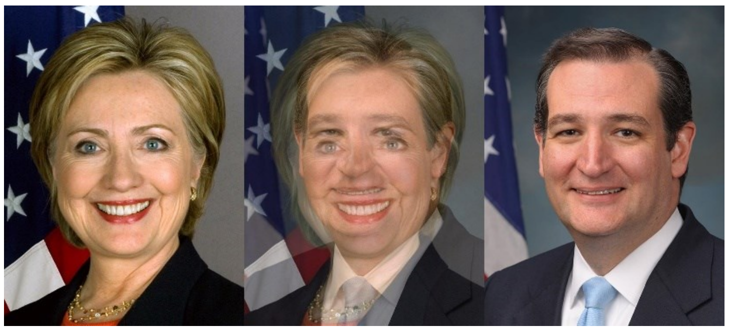


Figure 2: From left to right; starting point, an intermediate state and the target image of the morphing process

2. Getting Familiar with the Wiener filtering (15 Pts.)

When a camera or scene undergoes linear motion over a distance of L pixels during the exposure time T, each object point is *smeared* along a straight-line segment:

$$(0,0) \rightarrow (L\cos\theta,\ L\sin\theta),$$

where θ is the direction of motion, measured counter-clockwise from the positive x-axis.

In the continuous domain, this degradation can be modeled as a convolution:

$$g(x,y) = [h * f](x,y) + \eta(x,y), \qquad \eta(x,y) \sim \mathcal{N}(0,\sigma_\eta^2),$$

where:

- $f(x,y)$ is the original (undegraded) image,
- $h(x,y)$ is the point spread function (PSF),

- $\eta(x, y)$ is additive white Gaussian noise.

In the discrete case, the PSF becomes a finite kernel of size $P \times P$, with $P \geq L$, defined as:

$$h[i,j] = \begin{cases} \frac{1}{L}, & \text{if } (i,j) \text{ lies on the line of length } L \text{ and direction } \theta, \\ 0, & \text{otherwise,} \end{cases} \qquad \sum_{i,j} h\,[i,j] = 1.$$

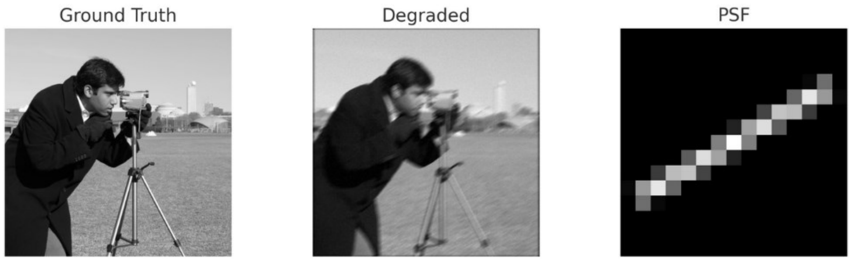


Figure 3: Example of linear motion blur

a) **Simulate motion blur and noise.** Implement a function that constructs a motion blur kernel of size $L \times L$ given a blur length L and an angle θ, as described above. Use $L = 15$ pixels, $\theta = 30^\circ$ for all simulations in this assignment. Convolve the provided ground-truth image `cameraman.png` with this PSF. Then, add white Gaussian noise with small variance to the resulting image. Report the final degraded image.

b) Compute the centred 2-D DFTs of both the original and the degraded images. Display the *log-magnitude spectra* side-by-side (use the usual $\log(|F| + 10^{-3})$ visualisation).

c) **Manual measurement of L and θ.** The motion PSF acts as

$$H(u,v) = \frac{\sin[\pi(u\cos\theta + v\sin\theta)L]}{\pi(u\cos\theta + v\sin\theta)L},$$

whose numerator is zero whenever $u\cos\theta + v\sin\theta = k/L,\ k = \pm1, \pm2, \ldots$. These zeros appear as *dark, parallel lines*. Explain why

$$\theta \ = \ \phi_{\text{lines}} + 90^\circ, \quad L \ = \ \frac{1}{\Delta u},$$

where ϕ_{lines} is the angle of the dark lines and Δu is their spacing in the frequency axis. Measure θ and L *with an image-viewer's tool* on your spectrum plot and report the values you obtain.

d) **Naïve Inverse Filtering.** Using your hand-measured L, θ, build the PSF h and perform plain inverse filtering in the frequency domain. Re-transform to obtain the spatial image $f_{\text{inv}}(x, y)$. Include this restoration in your report and comment on the residual blur, ringing, and noise amplification.

Now, we try the *Frequency-Domain* Wiener Filter for better restoration. It assumes that the *signal* f is a wide-sense stationary random field with flat power spectrum of variance σ_f^2 and is uncorrelated with the noise. Taking the 2-D discrete Fourier transform (DFT) gives

$$G(u,v) = H(u,v)\,F(u,v) + N(u,v),$$

where capital letters denote DFTs. The minimum-mean-square-error *linear* estimate of F in the frequency domain is the classical **Wiener filter**

$$\boxed{\hat{F}(u,v) = \frac{H^*(u,v)}{|H(u,v)|^2 + K}\, G(u,v), \qquad K = \sigma_\eta^2/\sigma_f^2}.$$

H^* is the complex conjugate of H. K is a scalar balancing noise amplification against blur suppression.

Choosing K.

Estimate the noise variance σ_η^2 from a visually flat patch of g (e.g., the sky). Estimate the signal variance σ_f^2 from the global variance of g. Typical K values are 10^{-3}–10^{-2}.

e) Implement the above procedure and compare with the last section.

3. Adding Different Types of Degradations to Images (20+10 Pts.) ⭐

The objective of this exercise is to deepen your understanding of image degradation processes and advanced restoration techniques. You will simulate realistic degradation (including motion blur and noise), implement frequency-domain filters, and explore both known and blind restoration approaches to recover the original image.

a) Simulate Realistic Motion Blur

Goal: Understand motion blur in a real-world setting.

Task:

1. Write a function to simulate **linear motion blur**:
   - Blur length L, angle θ.
2. Create a motion blur PSF (point spread function).
3. Convolve an image with the PSF to create a degraded version.
4. Add Gaussian or Poisson noise.

Challenge: Implement the PSF generation without using built-in OpenCV blur functions.

Hint:

- Use a line kernel in a given direction as a PSF.
- `scipy.signal.convolve2d()` or FFT-based convolution.

b) Non-blind Restoration Using Wiener Filtering

Goal: Apply Wiener filtering using a known degradation function.

Task: Given the motion-blurred and noisy image from part a. Assume known PSF and known noise variance. Implement Wiener filter in the frequency domain. Compare restored image quality (e.g., PSNR, SSIM) against original.

Challenge: Visualize magnitude/phase of DFT at each stage.

c) Blind Deconvolution (Iterative PSF Estimation)

Goal: Estimate blur kernel and restore image without knowing the degradation function.

Task:

1. Input: a blurred image only (you know it was blurred using a motion PSF).
2. Use **iterative blind deconvolution** to:
   - Estimate the blur kernel (PSF).
   - Restore the image.
3. Evaluate accuracy of PSF estimation.

Challenge: Initialize with a delta function, update using a known optimization method (e.g., alternating minimization).

Hint: Refer to **Richardson–Lucy deconvolution**.

d) Comparative Restoration Strategy Evaluation

Goal: Evaluate multiple restoration techniques on the same degraded image.

Task:

1. Degrade one image with:
   - Motion blur + Gaussian noise
   - Gaussian blur + Poisson noise
2. Apply and compare:
   - Inverse filtering
   - Wiener filtering
   - Constrained least squares filtering
   - Median filtering (for impulse noise)
3. Quantitatively and visually compare PSNR and SSIM.

e) Custom Edge-Preserving Restoration

Goal: Restore an image without blurring edges.

Task:

1. Degrade an image with Gaussian noise.
2. Implement a **bilateral filter** or **guided filter** from scratch.
3. Compare results with Gaussian and median filters.

Challenge: Optimize for speed using vectorized NumPy or CUDA (optional).

f) Restoration Pipeline for Real Images

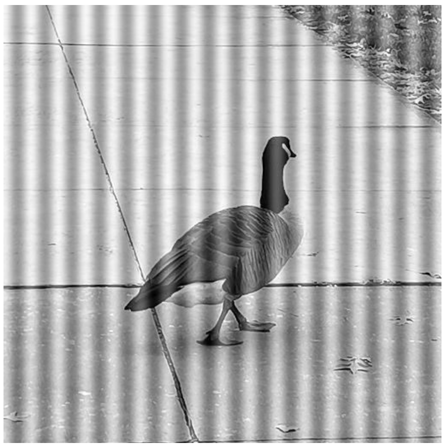
Goal: Build a mini real-world pipeline.

Task:

1. Use a real blurred image (capture a defocused or shaken photo).
2. Design a **complete pipeline** to:
   - Estimate PSF
   - Restore image
   - Post-process (sharpen, contrast adjust)
3. Output final image and intermediate steps.

4. Applying Notch Filters to Remove Periodic Patterns (20 Pts.)

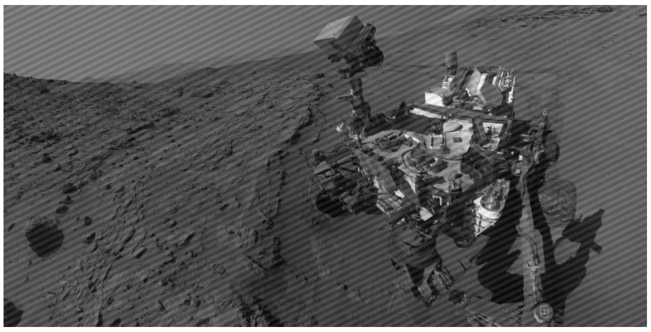
Notch Filters are used to reduce the effect of periodic noise or repetitive patterns from an image. They do so by attenuating a selected frequency – and some of its neighbors – and leave other frequencies of the image Fourier transform unchanged.



In this problem, you will get familiar with notch filters and their capabilities in removing periodic patterns – including noises – from an image.

a) Load the images “curiosity_rover_selfie.png” and “Goose.jpg” and display the corresponding spectrum. As can be seen, the image suffers from a periodic noise. Use notch filters to remove it from the images, and display the result.

b) Load the image “vertical_blinds_garden.jpg” and display the corresponding spectrum. As you can see, the blinds have obscured the view. Because of its natural pattern, one can use image restoration techniques in frequency domain to deal with this problem. Design and apply a notch filter to reduce the effect of this periodic pattern from the image, and display the results.



5. Getting to Know Some of the Simplified JPEG Compression Steps (15 Pts.)

Joint Photographic Experts Group (JPEG) is a common method used for compression of digital images. The JPEG compression algorithm is at its best on photographs of realistic scenes containing smooth variations of color and tone. It is also very popular for web usage, where reducing the amount of data used for an image is vital for responsive representation.

In every variation of it, JPEG compression algorithm consists of several steps, which you will get familiar with one of them in this problem.

Discrete Cosine Transform (DCT) of the input image is an intermediate step in JPEG compression. DCT method for image compression is the most popular technique for image compression over the past several years. Its selection as the standard for JPEG compression method is one of the major reasons for its popularity. Using the instructions given in this link, apply image compression using discrete cosine transform on the image “shahnameh.png”. Set block size as 4, and discard 75% of the DCT coefficients in each block. Display the result, and compute the PSNR related to the original image. Also comment on the visual differences among the input image and the corresponding compressed image.

6. Hands-On Image Processing: Huffman coding (15 Pts.)

Consider the simple 4×8, 8-bit image:

$$\begin{matrix} 21 & 21 & 21 & 95 & 169 & 243 & 243 & 243 \\ 21 & 21 & 21 & 95 & 169 & 243 & 243 & 243 \\ 21 & 21 & 21 & 95 & 169 & 243 & 243 & 243 \\ 21 & 21 & 21 & 95 & 169 & 243 & 243 & 243 \end{matrix}$$

a) Compute the entropy of the image.
b) Compress the image using Huffman coding.
c) Compute the compression achieved and the effectiveness of the Huffman coding.
d) Consider Huffman encoding pairs of pixels rather than individual pixels. That is, consider the image to be produced by the second extension of the zero-memory source that produced the original image. What is the entropy of the image when looked at as pairs of pixels?
e) Consider coding the differences between adjacent pixels. What is the entropy of the new difference image? What does this tell us about compressing the image?
f) Explain the entropy differences in (a), (d) and (e).

Good Luck