

Introduction to Image and Video Processing

spatial filtering, processing

due June 2

Alexia Briassouli (alexia.briassouli@maastrichtuniversity.nl)

Spring 2022

Instructions:

Each of these projects will count for 1/2 of your final grade. They will be checked for plagiarism (software + text). **You are requested to hand in one zip file with title “YourLastName_project1” (e.g. John Doe should submit “Doe_project1.zip”).** The zip file should contain:

1. An about 10 page report with your answers to the questions and figures with your results.
2. The code for producing these results *with clear comments in the code!* You can use any programming language you are comfortable with (preferably Python or Matlab for this class, but we can discuss others). You should explain what you think are important parts of the code in the report (e.g. if you use a special trick that you are proud of).

Your grade will depend on how clearly you present and explain your results in the report and code. You are allowed some freedom to explore solutions (e.g. if you are asked to come up with your own method), so there is no one correct answer. However, you should demonstrate you have understood the class material and how it applies to these projects.

1 Image degradation with motion blur and additive noise

In this exercise you will work on noisy color images, so all the operations described below should be applied to color images and not grayscale. Choose one of the color images below OR an image of your liking where you will add **diagonal motion blur and additive noise**.



Degrading the image and adding noise:

1. Create a diagonal motion blurring degradation function using the definition:

$$H(u, v) = \text{sinc}(\alpha \cdot u + \beta \cdot v) \cdot \exp(-j\pi(\alpha \cdot u + \beta \cdot v)).$$

For $\text{sinc}(x) = \sin(x)/x$ use an inbuilt function. Apply this motion blur to your image in the frequency domain. Choose your own α and β so that you can get a realistic diagonal blur - i.e. not too noisy, but visible.

2. Apply additive Gaussian noise *to the motion blurred image* in the Fourier domain using an inbuilt function. Choose a mean and standard deviation for the noise that produces noise that is visible but does not completely degrade the image.
3. Display (1) the image degraded by motion blur and (2) the image degraded by motion blur and additive noise.

Removing noise: Assume you know $H(u, v)$ (use the one you added motion blur with).

1. Apply direct inverse filtering to the image after it has undergone only motion blur. Display your denoised image and explain the result.
2. Apply direct inverse filtering to the image after it has undergone motion blur and additive noise. Display your denoised image and explain the result.
3. Write the equation for the MMSE filter $H_W(u, v)$ when the image undergoes only motion blur. Apply this MMSE filter $H_W(u, v)$ to the image after it has undergone only motion blur. Display your denoised image and explain the result.
4. Write the equation for the MMSE filter $H_W(u, v)$ when the image undergoes motion blur and additive noise. Approximate the ratio of noise power spectrum to original image power spectrum $S_{nn}(u, v)/S_{ff}(u, v)$ by a constant K . Calculate K from the average of their true ratio (use the additive noise you added and the original image to find the average K). Apply the resulting MMSE filter to the image after it has undergone motion blur and additive noise. Display your denoised image and explain the result.

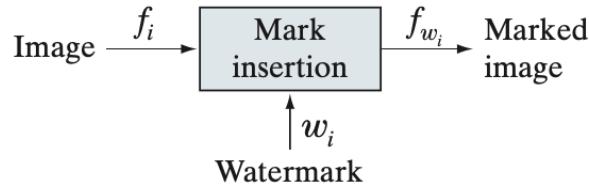
2 Hide a Secret Message in an Image DCT

This exercise is based on example 8.30 in Sec. 8.12 from Gonzalez (4th ed.). First read Sec. 8.12 and example 8.30. Then follow the instructions below (which are based on example 8.30).

A watermark can be inserted and extracted in the spatial or transform domain. Here you will add and detect your own watermark in the DCT domain, on a *grayscale* image of your choice.

2.1 Watermark Insertion

Choose a *grayscale* image you like, to which you will insert an invisible watermark following the steps below.



1. Compute the 2-D DCT of the image to be watermarked in a block-wise manner. You can use 8×8 or 16×16 blocks.
2. Choose its K coefficients, c_1, c_2, \dots, c_K with the largest magnitude. You have to determine what is a good value for K after some testing.
3. Create a watermark by generating a K -element pseudo-random sequence of numbers, $\omega_1, \omega_2, \dots, \omega_K$ that follow a Gaussian distribution with mean $\mu = 0$ and variance $\sigma^2 = 1$. (Note: A pseudo-random

number sequence approximates the properties of random numbers. It is not truly random because it depends on a predetermined initial value.)

4. Embed the watermark from Step 3 into K largest *non-DC* DCT coefficients in each block from Step 2. This means you should embed the watermark in what you consider are non-DCT important mid-frequencies as follows:

- (a) choose K so that the DCT coefficients for $i \geq K$ are significantly lower than for $i < K$
- (b) do not add a watermark to the DC coefficient ($i = 0$)

Add the watermark using the following equation:

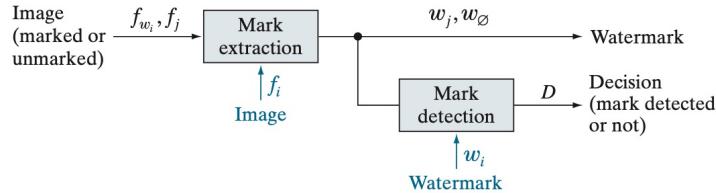
$$c'_i = c_i \cdot (1 + \alpha \omega_i), \quad 1 \leq i \leq K$$

for a constant α , $\alpha > 0$ that controls the strength of the watermark. Test different values of α to find the one that best suits you.

5. Replace the K DCT coefficients c_i with c'_i .
6. Compute the inverse DCT of the DCT with the new coefficients c'_i and display in the spatial domain: (1) the original, (2) the watermarked images and (3) their difference image. Discuss the appearance of the watermarked and difference images, in relation to the steps you carried out above.
7. Display the histogram of the difference image (between the original and watermarked images) in the spatial domain and discuss its appearance, in relation to the steps you carried out above.
8. Explain why you do not put a watermark on the DC coefficient.

2.2 Watermark Detection

Now you will be given a mystery image and you'll have to find if it is watermarked by comparing it with the known original image, following the procedure described below. Use the watermarked image from the previous exercise as your mystery image (pretend you don't know the watermarked image has an invisible watermark).



1. Compute the 2D DCT of the $M \times N$ mystery image with coefficients \hat{c}_i , $i = 1, \dots, MN$.
2. Keep its K largest non-DC DCT coefficients, now denoted as $\hat{c}_1, \hat{c}_2, \dots, \hat{c}_K$. Use the same $K < MN$, found in the watermark insertion part above.
3. Estimate an approximation of the watermark in your mystery image:

$$\hat{\omega}_i = \frac{\hat{c}_i - c_i}{\alpha c_i}, \quad 1 \leq i \leq K \quad (1)$$

where c_i, α, K are known from above and \hat{c}_i are the K non-DC coefficients of your mystery image.

4. Assume you know the mean $\bar{\omega}$ of the watermark sequence $\omega_1, \omega_2, \dots, \omega_K$ and $\hat{\bar{\omega}}$ the mean of the approximated watermark sequence $\hat{\omega}_1, \hat{\omega}_2, \dots, \hat{\omega}_K$. Measure the similarity of $\hat{\omega}_i$ with ω_i using the correlation coefficient:

$$\gamma = \frac{\sum_{i=1}^K (\hat{\omega}_i - \bar{\hat{\omega}})(\omega_i - \bar{\omega})}{\sqrt{\sum_{i=1}^K (\hat{\omega}_i - \bar{\hat{\omega}})^2 \sum_{i=1}^K (\omega_i - \bar{\omega})^2}}, \quad 1 \leq i \leq K.$$

5. Compare the measured similarity γ with a threshold T of your choice, to make the decision:

$$D = \begin{cases} 1, & \text{if } \gamma \geq T \\ 0, & \text{else} \end{cases}$$

If $D = 1$, an invisible watermark $\omega_1, \omega_2, \dots, \omega_K$ is present in your mystery image. There is no watermark if $D = 0$. Show and discuss your results.

6. Create a second mystery image with a *different* watermark using the same procedure as in question 2.1 as follows:

- (a) Use the same type of random distribution, with $\mu = 0$ but a different σ^2 . Choose your own σ''^2 , making sure the watermark is not too visible (or invisible). This will give you $\omega''_1, \omega''_2, \dots, \omega''_K$.
- (b) Use the same α to create the new watermark:

$$c''_i = c_i \cdot (1 + \alpha \omega''_i), \quad 1 \leq i \leq K$$

- (c) Display the new mystery image.

7. Find its DCT and consider its highest K non-DC DCT coefficients ω''_i , $i = 1, \dots, K$, with the same K as in the previous questions.

8. Use the same procedure as in question 2.2 to detect this watermark, using the same threshold T . What happens to γ and D now? Show and discuss your results.

3 Morphology

In this exercise you are allowed to use inbuilt functions for dilation and erosion only. You have to implement other kinds of morphological operations yourselves, using combinations of the inbuilt functions for dilation and erosion.

3.1 Count oranges

Pre-processing: You should use the images in Fig 2. Convert the images into black and white, so that the round objects appear white and the rest is black. You can use empirically determined threshold to separate and threshold the various colors. Your 3D RGB images should now be black and white with white circles/round objects, so you can implement black and white morphological operations in this part.



Figure 2: (a) Oranges. (b) Orange tree.

Question: Count the number of oranges in Figs 2 (a), (b). Explain all intermediate steps, display and explain your results. Use structuring elements that are appropriate, assuming you know the shape and size they should have (you can measure it from the images you will use).



Figure 3: Lights



3.2 Granulometry

Pre-processing: You should use the images in Fig 3. Convert the images into grayscale, so that the round objects appear lighter and the rest is darker, so you can implement grayscale morphological operations for this part.

Question: Find the frequency of the different sized lights that appear in Fig 3. Explain all intermediate steps, display and explain your results.

4 PCA - Recognition

In this exercise you will implement PCA for faces. You can use the provided images or your own set of images of faces (the ones provided are free from unsplash.com).

Make sure you use images of the same size. You can crop/resize the images so they have the same size. They should all show a face centered.

Use 3 images of different looking faces. Create 5 variations of each image by adding noise, varying illumination/contrast, changing the hair or other parts of the face using any app/software you like (there are many free online).

1. Find the eigenfaces for each of the 3 faces using their 5 variations as samples of each image. Explain step by step how PCA is implemented in your code. Show the resulting eigenfaces and explain their appearance.
2. Reconstruct each face using (1) 5 and (2) 2 eigenfaces. Display and explain your results.
3. Reconstruct each face using the eigenfaces corresponding to a different one, but the same eigenvalues. Display and explain your results.