# Introduction to Image and Video Processing
# spatial filtering, processing
# due June 3

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** in the frequency domain. If your original image FT is $F(u, v)$, the motion blur filter is $H(u, v)$, the noise FT is $N(u, v)$, the noisy image FT $G(u, v)$ will be:

$$G(u, v) = F(u, v)H(u, v) + N(u, v) \tag{1}$$

**Degrading the image and adding noise:**

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

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

   For $sinc(x) = sin(x)/x$ use an inbuilt function. Apply this motion blur to your image in the frequency domain. Choose your own $\alpha$ and $\beta$ 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 there is only additive noise (no motion blur). Apply this MMSE filter $H_W(u,v)$ to the image after additive noise only (no motion blur). Display your denoised image and explain the result. *Hint: $H(u,v)$ is not zero! It is going to leave F(u,v) unchanged in eq (1).*

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.

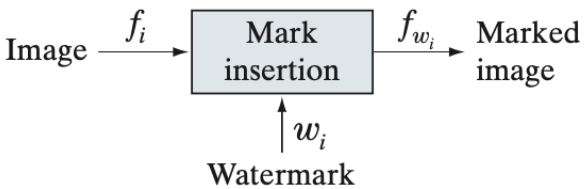
# 2 Hide a Secret Message in an Image DCT

This exercise is based on example 8.30 in Sec. 8.12 from Gonzalez ($4^{th}$ 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.




The steps below describe how you will insert a watermark. You will have to empirically choose the parameters $K, \sigma^2$ and $\alpha$ so that the watermark is invisible (see below for more on these parameters).

*This means you should choose some initial realistic values for $K, \sigma^2$ and $\alpha$, insert the watermark (as described below) and display the watermarked image to check if the watermark is visible. If the watermark is invisible, you can use these values. If not, you should tweak them (most likely reduce $\alpha$) and check again.*

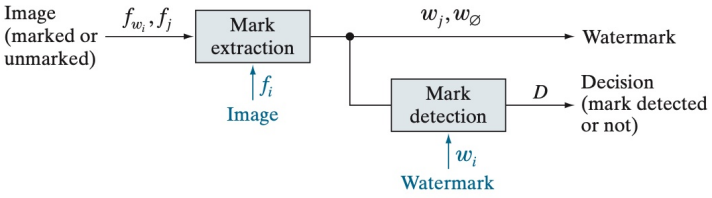
1. Compute the 2-D DCT of the image to be watermarked in a block-wise manner. You can use $8 \times 8$ or $16 \times 16$ blocks.
2. Choose its $K$ coefficients, $c_1, c_2, ..., c_K$ with the largest magnitude. You have to determine what is a good value for $K$ after some testing. *This means you should keep find the inverse DCT for various values of $K$ (keep $K$ highest DCT coefficient and set the rest to zero) and choose the smallest $K$ that creates an image that still looks like the original one, just using fewer DCT coefficients.*
3. Create a watermark by generating a $K$-element pseudo-random sequence of numbers, $\omega_1, \omega_2, ..., \omega_K$ that follow a Gaussian distribution with a mean $\mu = 0$ and variance $\sigma^2$ that you will choose.
4. Embed the watermark from Step 3 into $K$ largest *non-DC* DCT coefficients in each block from Step 2. Add the watermark using the following equation:

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

   for a constant $\alpha$, $\alpha > 0$ that controls the strength of the watermark. Test different values of $\alpha$ to find the one that best suits you. *This means you should insert the watermark using the next steps and display the watermarked image. If it looks like the original, then you keep the $\alpha$. If not, choose a smaller one so that the watermark is invisible.*
5. Create the DCT of your watermarked image by replacing the $K$ DCT coefficients $c_i$ that you kept 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. *Note: you will have to compute the inverse DCT in a blockwise manner, since you added it in a blockwise manner.*
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.




**Consider 2 cases for the mystery image**:

1. Use the watermarked image from the previous exercise as your mystery image (pretend you don't know the watermarked image has an invisible watermark).
2. Use the original image as your mystery image (pretend you don't know the original image doesn't have an invisible watermark).

Detect if the mystery image has a watermark **for these 2 cases** following the steps below.

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

$$\hat{\omega}_i = \frac{\hat{c}_i - c_i}{\alpha c_i},\ 1 \leq i \leq K \tag{2}$$

   where $c_i, \alpha, K$ are known from above and $\hat{c}_i$ are the $K$ non-DC coefficients of your mystery image (now the watermarked image).
4. Assume you know the mean $\bar{\omega}$ of the watermark sequence $\omega_1, \omega_2, \ldots, \omega_K$ and $\bar{\hat{\omega}}$ the mean of the approximated watermark sequence $\hat{\omega}_1, \hat{\omega}_2, \ldots, \hat{\omega}_K$. Measure the similarity of $\hat{\omega}_i$ with $\omega_i$ using the correlation coefficient:

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

5. Compare the measured similarity $\gamma$ 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 $\omega_1, \omega_2, \ldots, \omega_K$ is present in your mystery image. There is no watermark if $D = 0$. 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 that will be needed yourselves, using combinations of the inbuilt functions for dilation and erosion. You can use the provided images or your own that are similar to the ones provided.

## 3.1 Count oranges

**Pre-processing:** 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).

### 3.2 Granulometry

Figure 3: Lights

**Pre-processing:** Convert the images in Fig 3 into grayscale, so that the round objects appear lighter and the rest is darker. This is 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 $N$ variations (*at least 5*, and the same number of variations for each of the 3 images) 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 $N$ 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) all eigenfaces and (2) 2 (or another small number of) eigenfaces. Display and explain your results.
3. Reconstruct each face using 'wrong' PCA weights, corresponding to different ones. Display and explain your results.