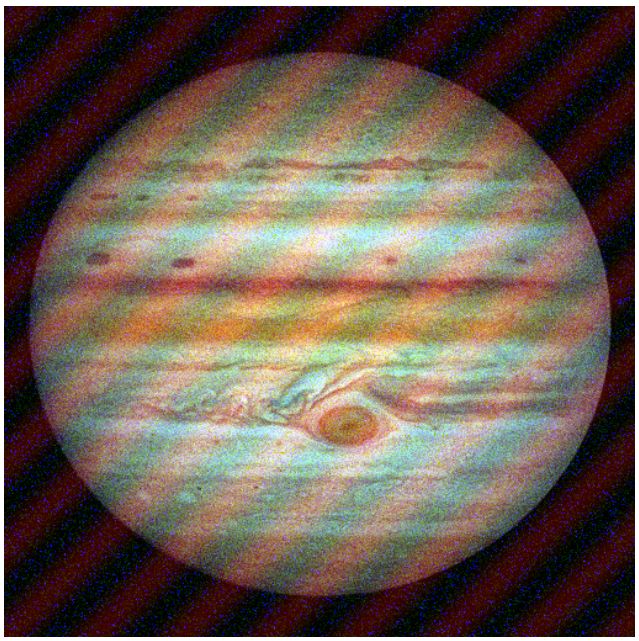
Take-Home Exam 2023
FYS 2010 Digital Image Processing

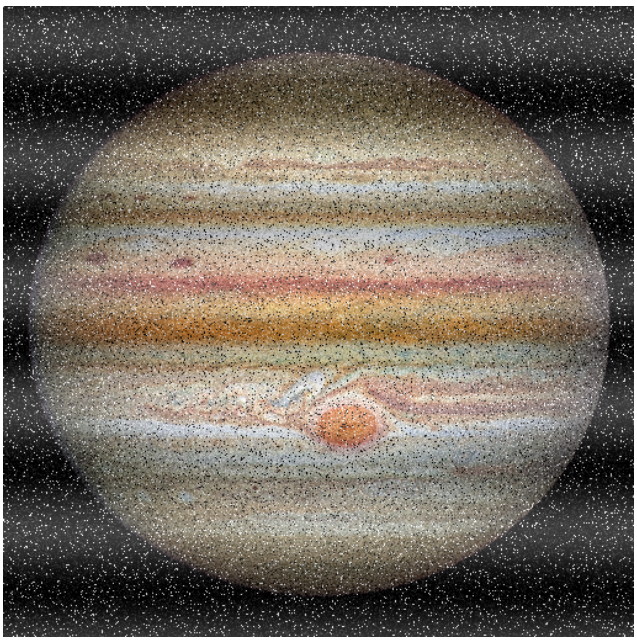
Start date: **20th of February 9:00**, deadline for submission: **13:00 the 13th of March**.

- You should provide a Zip folder containing your code, data and your report. Your report must be a pdf file or a Jupyter notebook. For long scripts of code you may put them in a separate file and refer to them in the report. You may write small (or large) parts of the code inside your report to make your point and make it easy to follow. You must as well convert your report to a pdf file (if it is not already one) and submit it separately in Wiseflow.
- Wiseflow checks automatically for plagiarism, so make sure you do not have identical, copy-pasted, answers from others. You may work in groups but the report you give must be your own personal, individual report.
- Images that should be analyzed will be provided on Canvas.
- All figures that you produce while solving the exercises should be included in the report. These should be annotated by labelled axes, captions and legends explaining the graphs when it is appropriate.
- Explain your approach to solving the problems, the assumptions you make, and how you reason. Lack of explanations will result in a lower score.
- If you can't actually solve a problem, due to time or programming problems, you may want to write down some notes about how you would have done it. Your intentions may gain you some partial score for the question.
- Some of the exercises may require you to make your own implementational choices. That is, the implementation of the algorithms may not be completely specified. In such cases, state your choice and argue for it. It does not need to be a lengthy argument.
- The exercises are weighted as follows:
  - Task 1 : 30% ($a : 5\%, b : 10\%, c : 25\%, c : 25\%, e : 35\%$)
  - Task 2 : 30%
  - Task 3 : 15%
  - Task 4 : 25%

  Questions inside Task 2, 3 and 4 have equal weight.



(a) Jupiter1.png



(b) Jupiter2.png

Task 1

You've taken two pictures of the planet Jupiter on a clear night. However, due to some unknown software error both images have been degraded by some unknown noise. Furthermore, the images have not been degraded in the same manner. In this task we'll attempt to restore both images.

For every task you may use packages such as *numpy*, *opencv*, *scipy*, etc which contains functions for most filters. You are however required to explain what methods / filters you're using and showing that you understand the reasoning for using said methods. Make sure your explanations and discussions are argued from theory.

a) Explain the general process of image restoration. How can an image become degraded, and what are we trying to achieve through image restoration?

b) Take a look at the images, or analyse them in any way you see fit. Describe the types of noise you see, and if possible provide indicators for the noise being what you see. Have the images been degraded in the same way? If not, how are they different? Be precise, use figures and plots to make your point.

c) Start off with restoring "Jupiter1.png", and try to restore this image. Choose filters you know off, or design your own filter to restore the image. Make sure to explain the filters you use, and your reasoning for choosing them. The explanation of your filters should be grounded in theory.

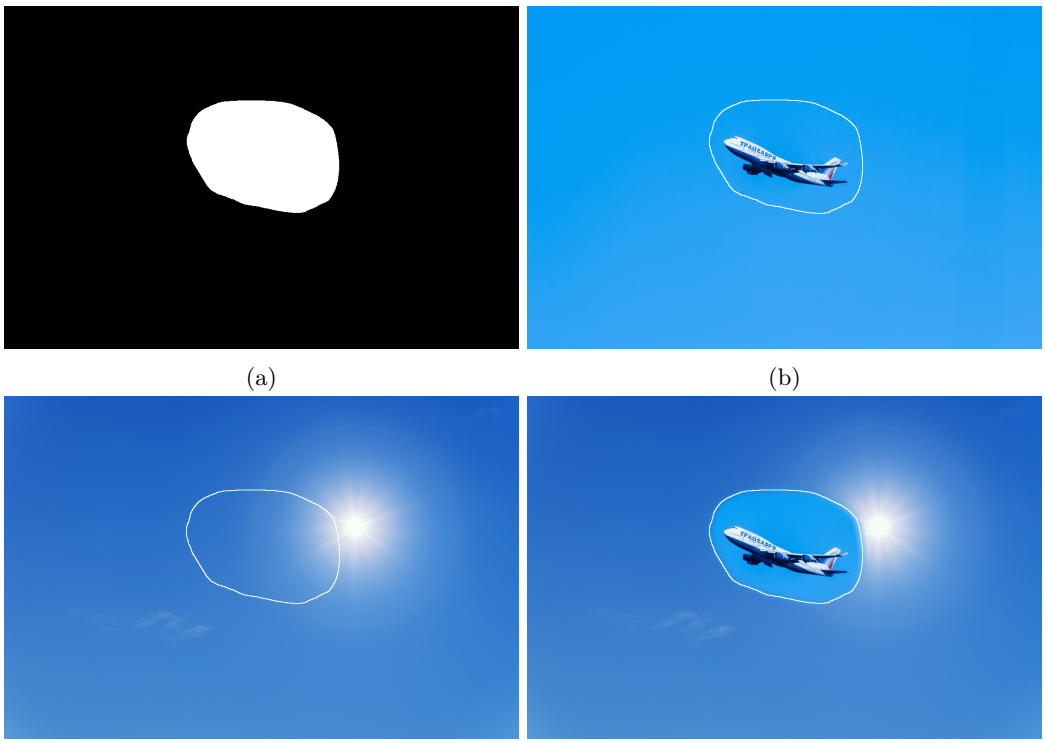
d) Now take a look at "Jupyter2.png". Will the restoration process be different in any way from the restoration process of "Jupiter1.png"? Choose filters or design filters and restore "Jupyter2.png". Explain your choices as you did in task **c)**.

e) Finally we want to see if we can enhance the images. Use what you've learned in this course about image transformations, filtering, and edge enhancing to improve the overall quality of the images. Your solution should contain one spatial, and one Fourier enhancement method (in total, not necessarily for both images). Explain your enhancement process, discuss the chosen methods strength and potential weaknesses, and how enhancing a color image differs from enhancing a grayscale image (if at all). Finally, present your enhanced images and discuss the results.

Task 2

For this question, most of the code has been provided to you. You may work in the Jupyter Notebook file `ImageCompositing.ipynb` or with the Google Colab notebook shared here: `https://colab.research.google.com/drive/1WUtcYNYUk2omAzchI9JsxZJlaDis2DPP?usp=share_link`. The code requires openCV, matplotlib, PIL, and scipy to run. If you have trouble installing any of these (particularly openCV), the Google Colab notebook is recommended. You should only change code in the sections marked for you to do so. (Reducing the variable `scale_factor` to a number less than 1 will scale down the images, which can make the code run faster for debugging.)

Image compositing is the process of taking a patch of one image (the source) and pasting it into another image (the target) in a region that is specified by a mask. The images in figure 2 show an example of a composite image. This example doesn't look very good though, and we can do better.



(a) (b) (c) (d)

Figure 2: Example of Image compositing showing the mask outlined in the mask (a), source (b), target(c), and final composite (d) images, all with the mask outline visible.

For the sake of this question, the source S, target T, and mask M images will all have the same size. The mask image is a 2D array of boolean values. Anywhere the mask is equal to 1 corresponds to image regions where we want to use pixel values from the source image.

a) Write an equation that can be used to perform simple image compositing and justify how the equation works. Then implement it in Python by editing the function `basicComposite()` in the marked area. Once you have implemented this compositing equation, include the resulting composites in your report and describe them.

Hint: numpy has specific broadcasting rules for doing mathematical operations with arrays. You can add an arrays with dimensions $(M, N, 3) + (M, N, 1)$*, but not* $(M, N, 3) + (M, N)$*.*

b) The composite produced in the part **a)** can be improved by smoothing the transition between the source and target images. Explain how this can be done using the same equation you wrote for part **a)**. Implement your idea in Python by editing the function `smoothComposite()` in the marked area. Once you have implemented this, include the smooth edge composites in your report and describe how they differ from the ones you produced in part **a)**.

Smoothing the masking edges for image compositing helps, but cannot adjust for differences in background color. Instead, we can introduce a way to compute image composites that unify the color of the composite both inside and outside the mask. This requires adjusting the colors of all the pixels within the mask in the source

image to blend well with the rest of the target image. This means we are trying to keep the edges of the source image while using colors from the target image.

We can do this by enforcing that the gradient of the image within the mask is as close to possible to the gradient of the source image, while the values of the pixels at the edge of the mask have to match the target image. This can be done with a technique called Poisson image editing, which can improve image composites as shown in figure 3.

(a) (b)

Figure 3: Comparison of smoothed-edge compositing (a) and the Poisson edit compositing method (b).

This can be found by solving for I in equation (1). Here I is the final composite image, S is the source image, T is the target image, Ω is the region where the composite mask is 1, $\partial\Omega$ is the edge of the composite mask, and ∇^2 is the Laplacian operator.

$$\nabla^2 I(x,y) = \nabla^2 S(x,y) \text{ in } \Omega \tag{1}$$

$$\text{s.t. } I(x,y) = T(x,y) \text{ on } \partial\Omega \tag{2}$$

c) Finding I that solves equation (1) in practice involves solving a linear system of equations. The number of unknowns and equations in this system is the same as the number of pixels inside the composite mask.

Expand equation (1) using the discrete 3×3 Laplacian for ∇^2. Leave the expanded equation in terms of arbitrary pixel coordinates x and y. You may assume the value of $\nabla^2 S(x,y)$ is a known quantity, because this is the Laplacian of the source image.

Do this once for a pixel that is surrounded by pixels within the mask, and once more for a pixel whose neighbor at $(x, y+1)$ is outside the mask.

$$L_{3\times 3} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Hint: Remember that outside the mask $I(x,y) = T(x,y)$.

d) Based on your expanded equation in **c)**, how would you solve it as a system of linear equations in the $Ax = b$ form? Describe the contents of A, x, and b. What is the largest number of non-zero elements in any row of A? What are the unique, non-zero values in A?

e) Edit the function `populateRow()` in the marked area to complete the code for creating a Poisson editing composite. You will need to define the values that will fill the coefficient matrix A. This function fills the matrix A by looking at one pixel at a time, and its four neighbors.

The variables you will need to complete this are:

- `A` is the coefficient matrix you are filling.
- `b` is the vector of values from the right-hand-side of the linear system. It currently only contains values of the Laplacian of the source image for each pixel within the mask.
- `mask` is the 2D boolean array representing the compositing mask.
- `target` is the numpy array of the target image.
- `ix` is the index of the current row of matrix `A`. This also corresponds to the ix^{th} pixel within the composite mask.
- `(r,c)` is a tuple of row-column coordinates of pixel `ix` in the image.

- `tup2idx` is a dictionary that maps row-column coordinate tuples `(r,c)` to pixel index ix. Ex: The line '`tup2idx[(250,275)]`' will return an integer defining which row of A the pixel at coordinate $(250, 275)$ corresponds to.

After editing `populateRow()`, include the produced composite images in your report and describe them and how they differ from the composites produced in parts **a)** and **b)**.

Hint: The `b` *array has the dimensions* $(n, 3)$ *for* n *pixels in the composite mask. This method solves for each color channel independently, so you will need to edit all 3 channels color channels in* `b`*. All three color channels use the same matrix* `A`*, so it is a 2D array with shape* (n, n)*.*

Hint 2: You should only worry about the 4-neighborhood around a specific pixel. That is, the pixels above, below, left, and right of the current pixel. The diagonals are ignored in this algorithm.

Task 3

Exercise 1

Recover the signal in the time domain from the following functions if:

1.1 the Fourier Transform (FT) in the angular frequency domain is:

$$X(\omega) = 2 + (1+j) \cdot \pi \cdot [\delta(\omega + \omega_0)] + (1-j) \cdot \pi \cdot [\delta(\omega - \omega_0)]. \tag{3}$$

Note: The inverse Fourier transform is here:

$$x(t) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} X(\omega) \cdot e^{j\omega t} d\omega. \tag{4}$$

The symbol δ is Dirac delta function (distribution).

1.2 the Discrete-Time Fourier Transform (DTFT) in the frequency domain is:

$$X(e^{j\omega}) = \frac{1}{(1 - \alpha e^{-j\omega})} \left(1 + \frac{1}{(1 - \alpha e^{-j\omega})}\right), \omega \in \mathbb{R}, \tag{5}$$

where α is real number. In the time domain, present the results using $x[n]$ where $n \in \mathbb{Z}$.

1.3 the Discrete Fourier Transform (DFT) in the frequency domain is:

$$X(k) = 3\frac{(1 - a^N)}{(1 - a \cdot e^{-i2\pi k/N})}, k \in [0; N], \tag{6}$$

where N is the period, a is a real number. In the time domain, present results using the notation $x[n]$.

Transform tables of FT, DFT, DTFT are available in the link: (`https://en.wikibooks.org/wiki/Engineering_Tables`)

Exercise 2

For this exercise, detail all the steps of your computations.

1. Compute the *cyclic* convolution of the following 2D spatial filter:

$$w[x, y] = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 2 \\ 1 & 2 & 3 \end{bmatrix},$$

with the following 2D discrete unit impulse:

$$f[x, y] = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}.$$

2. Compute the convolution $w * f$ using zero-padding.
3. Compare the results of question 1 and 2.

Task 4 Point cloud image and filtering

In this exercise, you are asked to filter an image f which is made of pixels arranged in an irregular manner over a 2D surface. The file `point_cloud.npz` contains the pixels (x,y) coordinates in a matrix X and their respective greyscale value $z(x, y)$ in the vector Z. The convention is that the origin $(0, 0)$ is at the top left corner. From this "point cloud" we have made a graph by connecting the nearest neighbors together in 2D. The Laplacian on this graph L is given in the file `point_cloud_Laplacian.npz` as a matrix.

1. Load the data and use the coordinates in array X and greyscale values Z to plot the point cloud image z. This point cloud was made by subsampling the pixels of the image `coffee.png`. Compare your plot and the original image (in a few words).

2. Display the histogram of the point cloud intensity values. Give a short explanation about the shape of this histogram.

3. Using the graph Laplacian L, compute the Graph Fourier modes of this irregular image domain. *Note:* the Graph Fourier Transform is different from the Discrete Fourier Transform. *Hint:* from the computation you should obtain some eigenvalues and eigenvectors. Let us call the eigenvalues the "frequencies" and the eigenvectors the Fourier modes. Order the eigenvalues and associated eigenvectors in increasing order, if they are not, and plot the eigenvalues. Check that they are all positive (up to Python numerical error).

4. Plot the Fourier mode associated to the first non-zero eigenvalue on the graph. In other words, do a scatterplot of the pixels, as in question 1), where the original image intensities have been replaced by the eigenvector values. What do you observe? Plot a few different other Fourier modes and explain what you see.

5. Perform the Graph Fourier Transform $\mathcal{F}(z)$ of the image z. It should be a vector with the same size as Z. Plot $\mathcal{F}(z)$ on a figure where the x-axis represents the Laplacian eigenvalue "graph frequency". Which frequency band the image have the highest values in?

6. Apply an ideal lowpass filter to $\mathcal{F}(z)$ with a cutoff frequency c of your choice and perform the inverse graph Fourier transform to get a lowpass filtered version of z. Plot the resulting point cloud image. What do you observe? Adjust the cutoff frequency c such that the effect is visible on the image and give the value of c.

7. Same question as in 6), but replace the lowpass filter by an ideal highpass filter.

8. Conclusion: does the filtering on this graph/point cloud domain works as in the standard image domain? Justify your answer in a few sentences.

Remark: If you are curious to know how the point cloud, graph and Laplacian were generated, have a look at the file `code_generating_pointcloud.py`.