Name: Guilherme Stabach Salustiano email address: g.stabachsalustiano@student.utwente.nl

Student ID 3301311

Educational Program: EESCM

Exercise 1: Basics of convolution and Fourier transform

Matlab Preamble

- During development of your m-code, you might want to inspect images and image-like data (such as Fourier magnitude spectra) on your screen for inspection. This is done as follows:

```
figure;                  % create a figure window
imshow(im);              % write the image to the current graphical object
```

- If the data is out of range (for type doubles: not between 0 and 1), and it is an intensity (grey) image, you can adapt the range of screen by (this is also useful if the range is so small that contrasts are hardly visible):

```
imshow(im,[]);           % adapt the range
```

- For color images, this does not work. If you want to adapt the scale, you use:

```
imshow(mat2gray(im));% adapt the range
```

- For the report, you have to write the images to file first. This is done as follows:

```
imwrite(im,'fname.jpg');                    % without adapting the range
imwrite(mat2gray(im),'fname.jpg');          % with adaption of the range
```

- For non-image like data, i.e. graphical data such as plots, you have to save the whole figure window to file. This is done as follows:

```
print –r300 –dpng fname.png     % save the current figure window as a  png
                                % file. The resolution ,300 dpi, is an
                                % example.
```

The report and the m-code

You create a **single** m-file for all questions. The code must be such that **all** code can be executed by a single click on the "RUN" button (with the green arrow). Put the following line between each question:

```
%% Question 1  (etc)
```

You copy-and-paste all your code to the end of the report. Make sure that the line length does not exceed 95 characters since otherwise the code does not fit the width of a page.

Questions

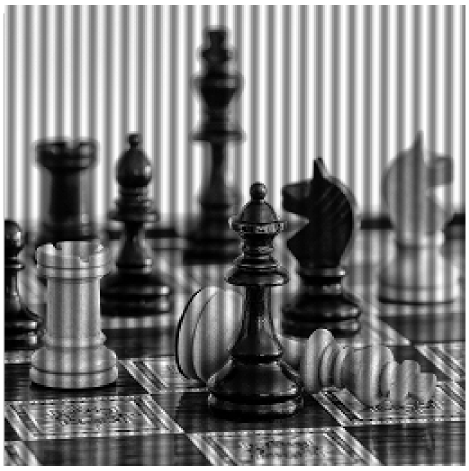
The image, stored in the file `car_dis.png`, is distorted. The objective is to apply a filter that removes the distortion without affecting the original image.

1. Write and execute an m-file that:
   - clears the workspace and closes all possible figure windows (use `clear variables` and `close all`, respectively)
   - reads the image from file (`imread`), converts it to an intensity image of type double (`im2double`), and displays the image in a figure window on the screen (`imshow`).
   - calculates the Fourier transform and shows the log-amplitude spectrum as an image on the screen. Be sure to do so in a controlled manner: (i) the origin should be in the centre, and (ii) the logarithmic scale maps the magnitudes to intensities such that the interesting part of the range of magnitudes is displayed well. Include a title, and the correct x- and y labels to the axes. Write the resulting figure window as a png image[1] to file (use: `print –r150 –dpng Imlogmag.png`). **Do not use the copy-and-paste option, nor the 'save as' of the figure window as the result is not reproducible (it will depend too much on your screen resolution).**

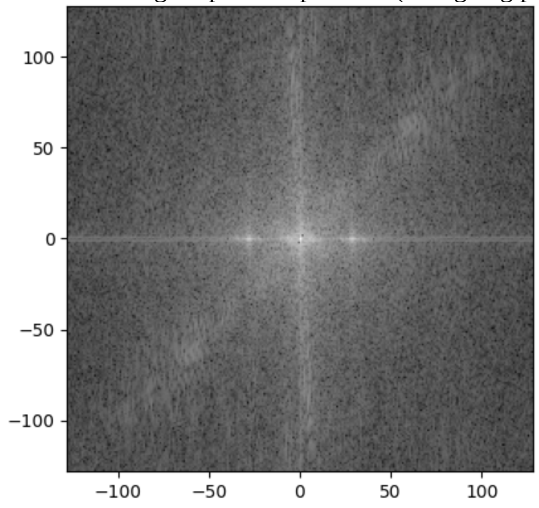
Note: Do not copy-and-paste text from this pdf into your Matlab editor: some fonts in the pdf might slightly differ from the ones that the editor expects

[1] The PNG (Portable Network Graphics) is a lossless image file format. For image analysis, a lossless coding is often preferred as a lossy encoding, such as JPEG, might induce loss of information. Another popular format is TIFF which supports both lossless and lossy encoding.

Insert the car_dis.png



Insert the log-amplitude spectrum (Imlogmag.png)



2.a What kind of symmetry do you observe in the spectrum and how can you explain this?

They present central symmetry about the origin of the chord system.
This is because normal images are real real-valued and the imaginary part is zero. So, the complex harmonics always come in pairs, to make this sum real-valued.

2.b Suppose that the size of the input image `im` is `N×M.` How large will be the fft `IM` of this image? The function `fftshift` shifts the origin of the u,v frequency domain to the centre. At which row and column index is the origin of this domain (that is, u=0, v=0) located in the Matlab array `IM`, directly after application of `fft2`, and after application of `fftshift`. Give a mathematical expression express in Matlab notation using the given variables in this question AND the final numerical result for this case:

directly after `fft2`:	`column =`	0
	`row =`	0
after `fftshift`:	`column =`	floor(M/2) = 128
	`row =`	floor(N/2) = 128

2.c Suppose that the pixel size of the image `im` is Δ×Δ.The distortion in the image looks like a harmonic function. By using the cursor of the figure window, created in question 1a, determine, i.e. measure, the wavelength of this harmonic function (expressed in pixel size Δ ; that is the result could be, for instance, `4*Delta`). Give the frequency ρ of this harmonic, also expressed[2] in pixel size Δ :

`lambda =`	9
`rho =`	1/9 = 0.111...

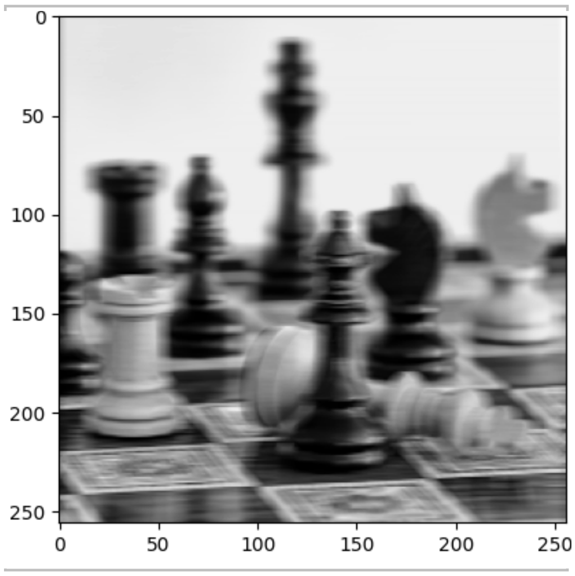
2.d Suppose that the distortion has a vertical orientation, so that in Cartesian coordinates we have $u = \rho$ and $v = 0$. If the image size is N×M, and its fft `IM` is centred with `fftshift`, at what row and column index one of the dominant frequency components of the distortion could be expected in the log-magnitude spectrum? Give Matlab expressions and the numerical result for this particular case:

`col =`	M/2 + M*ρ = 128 + 28 = 156
`row =`	N/2 = 128

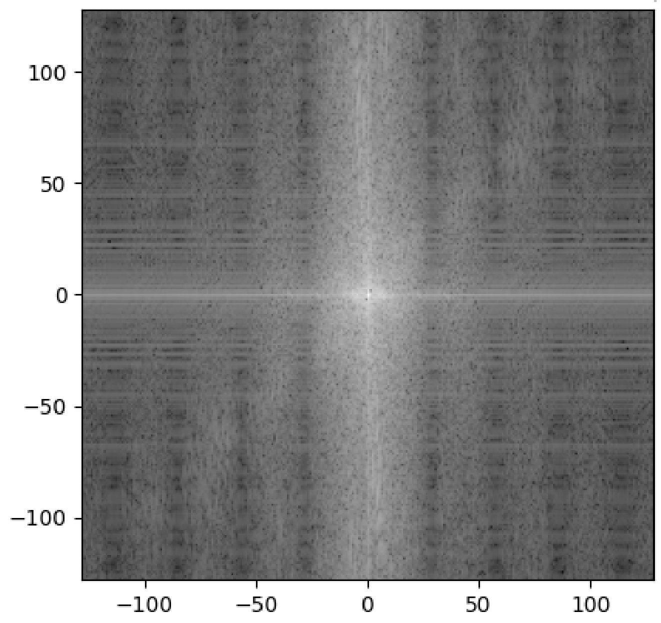
[2] Hint: useful information is on the slide entitled: “Discrete Fourier transform” of the lecture sheets.

3. Extend the m-file with code that filters the image with a rectangular PSF. In Matlab the corresponding filter is called ‘`average`’. Select the horizontal and vertical sizes of the PSF such that the distortion is suppressed, but the image details are preserved as much as possible. (use `fspecial` and `imfilter`). Write the resulting image to file in JPEG format, e.g., `imwrite(imfil,’IMfil.jpg’)`; Calculate, display and store the log-magnitude spectrum of the filtered image, as was done in Q1.

Insert the filtered image here:



Insert the log-amplitude spectrum of the filtered image:



3.a What is the exact size of the PSF and motivate its specific width and length.

`Nhorz =`	9
`Nvert =`	1

The noise is a horizontal sinusoidal signal and the mean of the sinusoidal signal is zero. So I designed a filter with a horizontal size like the lambda of the sine wave, to remove it

3.b Explain what this filter should suppress in the spectrum. Does the filter also affect the informative part of the image? And is this inevitable, or could another choice of the PSF circumvent this?

The filter should suppress the horizontal sine wave. At the same time, it affects all the image information, which is inevitable in average filters.

3.c What do you observe at the border of the image, and how do you explain this?

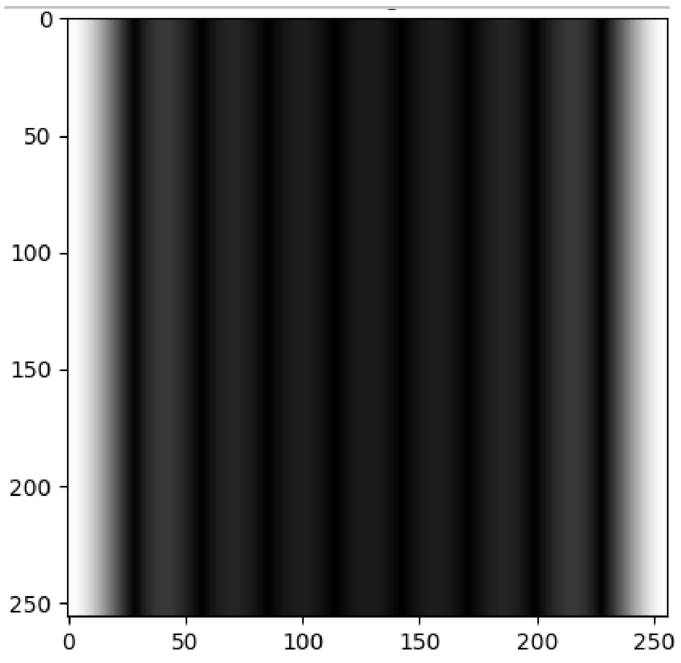
The border on the left is a little darker, and the border on the right is a little lighter. This happens because by default opencv uses the BORDER_REFLECT_101 border.

3.d Add one or option(s) to '`imfilter`' (refer the documentation of this function) such that the result is as natural as possible (the natural result would be obtained if image data outside the image plane would have been available). Note: in future, use these option(s) whenever you want to use `imfilter`.

Option(s) used: borderType=cv.BORDER_REPLICATE

4. Extend the m-file with code that calculates the otf associate to the filter found at Q3 and stores this in an array `OTF`. For that, use the function `psf2otf`. See the documentation. If you use this function with only one argument being the psf matrix, then the size of `OTF` will be equal to the one of the psf matrix (you can inspect that easily). With an additional option, you can calculate the otf at a much higher resolution. Select the second option such that the resulting array has the same size as the image in Q3. Next, shift the origin to the centre of the graph, and show the magnitude of `OTF` as an intensity image in a figure window. Apply a linear scale to map the magnitudes to intensities. Don't forget to add the title and the labels at the axes.

Insert the magnitude of the OTF image:



4.b Inspect the imaginary part of the otf , and explain the result.

`max(abs(imag(OTF(:))))` = 0.0

Explanation:

The OTF is symmetric, so its Fourier transform is just composed of phaseless harmonics

4.b In Q2 you have identified the dominant frequency (and corresponding row- and column-indices) that can be associated with the distortion. Use this information, to extract from the OTF the attenuation factor of this particular frequency.

Attenuation factor is: 87.74 dB

4.c Which type of this OTF represent (low pass, high pass, band pass, high emphasis, etc)?

Type of filter: Low pass filter

5. Instead of spatial convolution we can filter in the frequency domain. The strategy will be to define a filter transfer that blocks a range of frequencies that can be associated with the distortion. All other frequencies will be passed without attenuation.

5.a Suppose that `(row,col)` are the row and column indices of one of the dominant frequency as found in Q2d. We define a rectangular area around this dominant frequency. Suppose the size of the rectangle is set to `s x s`, where `s` is an odd integer. In Matlab, a rectangular range of indices can be represented by `rstart:rend`, and `cstart:cend`. These four parameters must be chosen such that `(row,col)` is in the centre of this rectangle. Give matlab expressions for `rstart, rend, cstart`, and `cend:`

`rstart =` row - floor(s/2)

`rend =` row + ceil(s/2)

`cstart =` col - floor(s/2)

`cend =` col + ceil(s/2)

5.b An OTF matrix H with the same size of the image, and filled with 1's everywhere except for the frequencies of the rectangle can be created with[3]:

```
H = ones(size(im));
H(rstart:rend,cstart:cend) = 0;
```

However, this code will not yield an OTF with the needed symmetry. Extend the code such that the symmetry requirements are met.

Code:

```
H *= H[:,::-1]
```

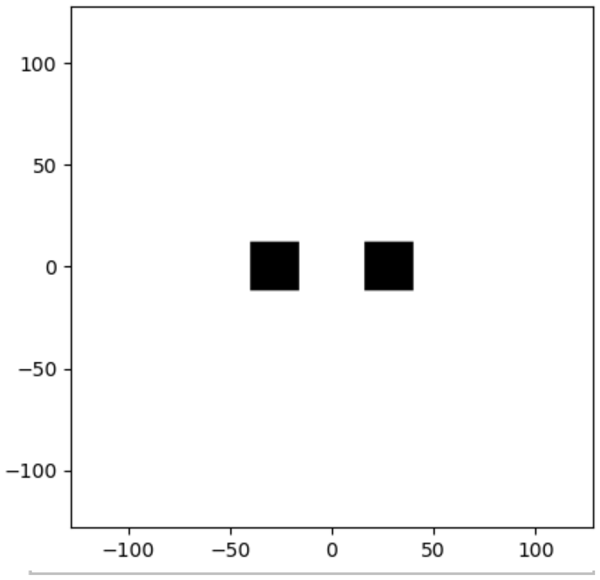
5.c Apply the filter. That is, perform a pixel-by-pixel-multiplication between H and the FT of the original image. Next, calculate the inverse Fourier transform of the resulting image and file this in your report. Very small imaginary parts (that is smaller than 10^{-14}) of the resulting image are due to numerical round-off errors. These parts may be removed. If the imaginary parts are substantial, then the matrix H did not comply with the symmetry requirements. Suppose that `imresult` is the final resulting image, then execute:

`max(abs(imag(imresult(:))))` = 0.0

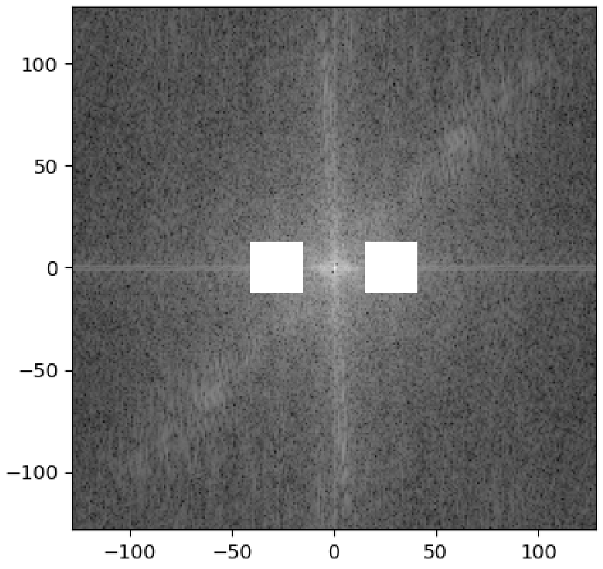
[3] Experienced Matlab programmers always try to avoid for-loops. Here, the colon operator (:) very useful. It produces concise and comprehensible code. Moreover, it may speed up Matlab considerably..

5.d Make graphical representations of the new OTF and the log-magnitude spectrum of the newly filtered image (as you did before).

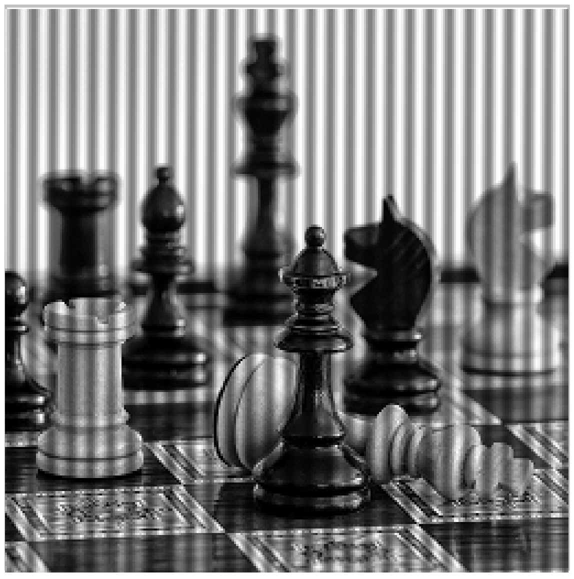
The magnitude of the found OTF image:

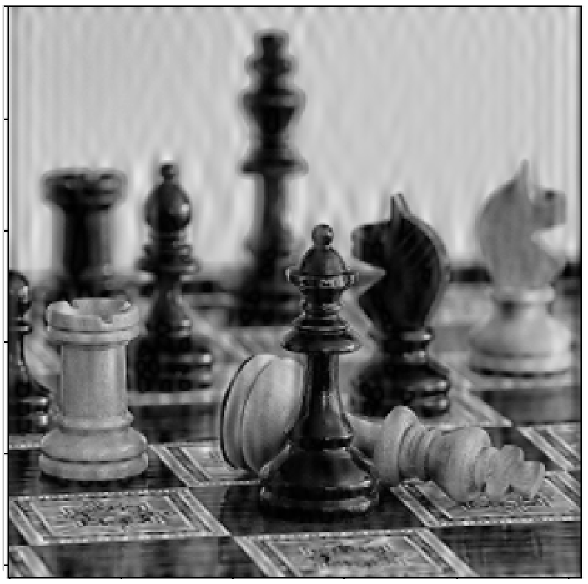


The log-magnitude of the newly filtered image



Show (for comparison) the distorted image and the newly filtered image:





6 In the resulting image, there will be some artefacts at the border of the image. These artefacts depend on the size of the chosen rectangle. Describe how, and why:

By removing the harmonic that was generating the horizontal noise, we also removed part of the signal that, together with others, kept the value constant at the edges.

Make sure the m-code fits within the PDF-margin (also the added comments)!

python-code:

```
def main():
    # read image
    img = cv.imread('car_dis.png', cv.IMREAD_GRAYSCALE)

    # convert to float64
    img = np.float64(img)/255.

    # usefull variables
    [img_size_y, img_size_x] = img.shape
    img_center_x, img_center_y = (img_size_x//2, img_size_y//2)

    # show original image
    show_image(img, 'Original image', filename = 'original.png')

    # Calculate FFT of original image
    dft, magnitude_spectrum = fft2(img)
    show_image(magnitude_spectrum, 'Magnitude Spectrum', 'Imlogmag.png', centrylized=True)

    # Plot one line to easy count lambda
    plt.plot(np.arange(img_size_x), img[0])
    plt.title('First line of original image')
    plt.show()
    lambda_ = 9 # Based on previous plot

    # Calculate noise position
    noise_fft_y = img_center_y
    noise_fft_x = img_center_x + img_size_x//lambda_
    print('Noise position in FFT:', noise_fft_x, noise_fft_y)
    print('Noise value in FFT:', magnitude_spectrum[noise_fft_y, noise_fft_x])

    # Apply average filter
    average_filter = np.ones((1, lambda_))
    average_filter /=  average_filter.sum()

    imgfil = cv.filter2D(src=img, ddepth=-1, kernel=average_filter, borderType=cv.BORDER_REPLICATE)
    show_image(imgfil, 'Image using average filter', filename='ImFil.png')

    # Calculate FFT of filtered image
    dft_fil, magnitude_spectrum_fil = fft2(imgfil)
    show_image(magnitude_spectrum_fil, 'Magnitude Spectrum After average filter', filename='SpectrumFil.png', centrylized=True)
    print('Noise value in FFT after average filter:', magnitude_spectrum_fil[noise_fft_y, noise_fft_x])

    # Calculate OTF
    otf = psf2otf(average_filter, img.shape)
    mtf = np.abs(otf)

    show_image(mtf, 'MTF average filter', 'MtfFil.png', centrylized=True)
    print("sum of otf's imaginary part:", np.sum(np.abs(otf.imag)))

    # Calculate atenuation factor
    atenuation_factor = magnitude_spectrum_fil[noise_fft_y, noise_fft_x] - magnitude_spectrum[noise_fft_y, noise_fft_x]
    print('Atenuation factor:', atenuation_factor)

    # Create FFT filter
    s = 24
    rstart = noise_fft_y - floor(s/2)
    rend =  noise_fft_y + ceil(s/2)
    cstart = noise_fft_x - floor(s/2)
    cend = noise_fft_x + ceil(s/2)
    H = np.ones_like(img)
    H[rstart:rend, cstart:cend] = 0
    H *= H[:,::-1]
    show_image(H, 'Mask "H', filename="H_ftt.png", centrylized=True)

    # Apply FFT filter
    dft_for = dft.copy() * H
    magnitude_spectrum_for = 20*np.log(np.abs(dft_for))
    show_image(magnitude_spectrum_for, 'Magnitude Spectrum after FFT filter', filename="fft_mag.png", centrylized=True)

    # Apply inverse FFT
    img_back_for = ifft2(dft_for)
    show_image(img_back_for, 'Image after FFT filter', filename='forrier.png')
    print("sum of img_back_for's imaginary part:", np.sum(np.abs(otf.imag)))
```

```
from math import floor, ceil
import numpy as np
import cv2 as cv
import matplotlib.pyplot as plt

# From lecture notes
def psf2otf(psf, shape):
  M, N = shape
  # get size of psf
  sz = np.shape(psf)
  # calculate needed paddings
  n = N - sz[1]
  m = M - sz[0]
  n1 = int(np.floor(n / 2) + np.mod(n, 2))
  n2 = int(np.floor(n / 2))
  m1 = int(np.floor(m / 2) + np.mod(m, 2))
  m2 = int(np.floor(m / 2))
  # pad array with zeros
  psf = np.pad(psf, ((m1, m2), (n1, n2)))
  # shift origin
  psf = np.fft.ifftshift(psf)
  # apply DFT
  otf = np.fft.fft2(psf)
  return otf

def show_image(img, title, filename = None, show_windows = True, centrylized = False):
  [img_size_y, img_size_x] = img.shape
  extent =  [-img_size_x/2, img_size_x/2, -img_size_y/2, img_size_y/2] if centrylized else None

  plt.imshow(img, cmap = 'gray', extent=extent)
  plt.title(title)

  if filename:
    plt.savefig(filename)
  plt.show()

# Based on from https://docs.opencv.org/4.x/de/dbc/tutorial_py_fourier_transform.html
def fft2(img):
  dft = np.fft.fft2(img)
  dft_shift = np.fft.fftshift(dft)
  magnitude_spectrum = 20*np.log(np.abs(dft_shift))

  return dft_shift, magnitude_spectrum

def ifft2(dft_shift):
    dft = np.fft.ifftshift(dft_shift)
    idft = np.fft.ifft2(dft)
    img_back = np.abs(idft)

    return img_back

if __name__ == '__main__':
    main()
```