Assignment 3 – Frequency, Filters and Noise

This assignment will count towards your grade.

Summary

The aim of this assignment is to perform low-level processing steps using image and signal frequencies, work with de-noising techniques, convolution, and edge detection.

Introduction

The following assignment needs to be presented as a Jupyter notebook. However it is suggested to write as much as possible of code with PyCharm as an external module and use Jupyter notebooks for visualization and code description only. This will allow you to easily debug the code.

A supporting set of functions are provided in the assign03.py library. Load them into your code to complete the assignment (using the techniques that you mastered in assignment 1). In the explanation, it is assumed that the library as been imported as follows:

```
import assign03 as a3
```

Also, we will be using the following shorthand for the standard libraries

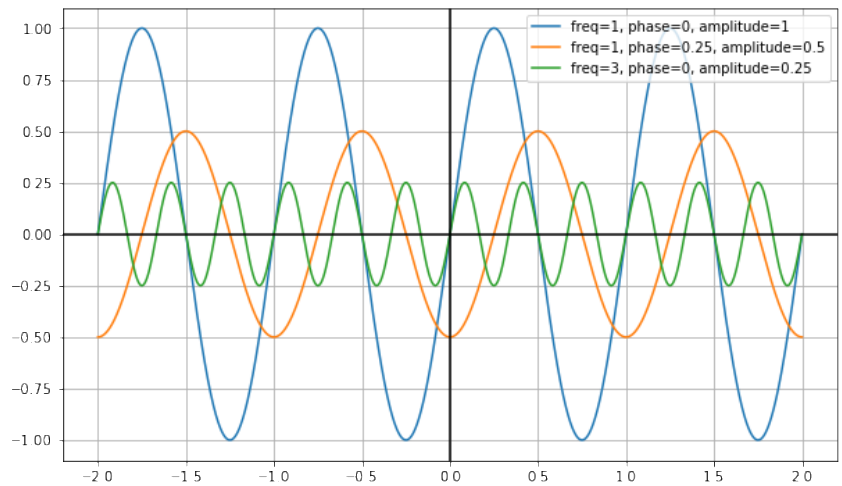
```
import matplotlib.pyplot as plt
import numpy as np
import skimage as sk
import skimage.io as skio
import skimage.transform as sktr
import skimage.filters as skfl
import skimage.feature as skft
import skimage.color as skcol
import skimage.exposure as skexp
import skimage.morphology as skmr
import scipy.ndimage as ndimage
```

Fourier Transform (3 points)

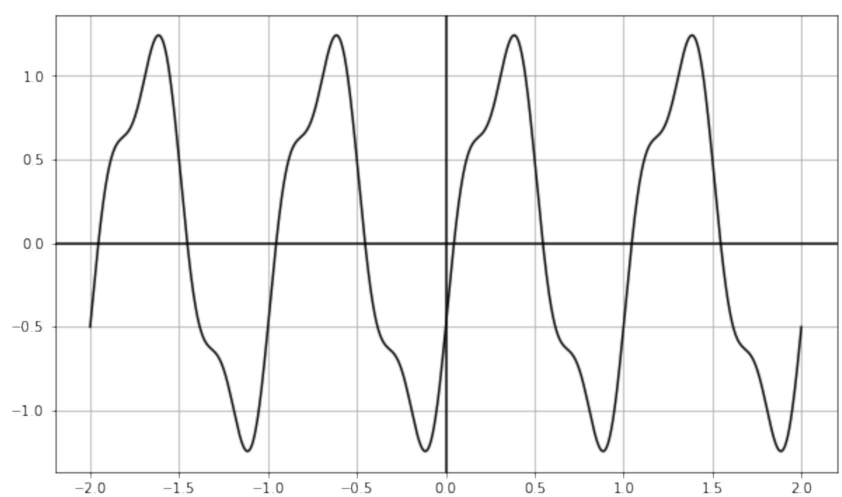
- Compute three sine waves the following parameters (expressed in radians):
  - frequency=1, phase=0, amplitude=1'
  - frequency =1, phase=0.25, amplitude=0.5
  - frequency =3, phase=0, amplitude=0.25
  - Use the a3.sin2 function to compute them. The following code can help you in defining the x domain for the function

```
FS = 1000 # number of samples in the x domain
DOMAIN = [-2, 2] # extent of the x domain
x = np.linspace( DOMAIN[0], DOMAIN[1], 2*FS )
```

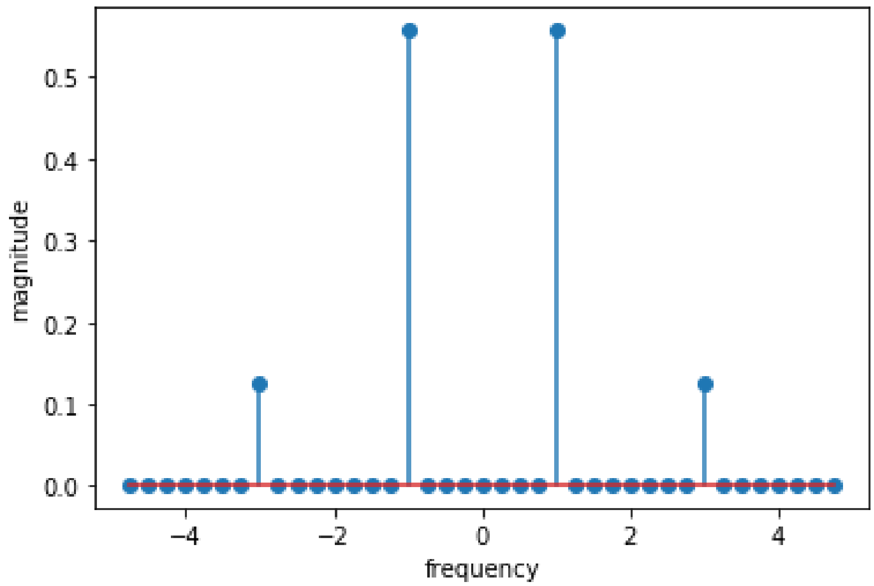
- Plot the three sine signals separately



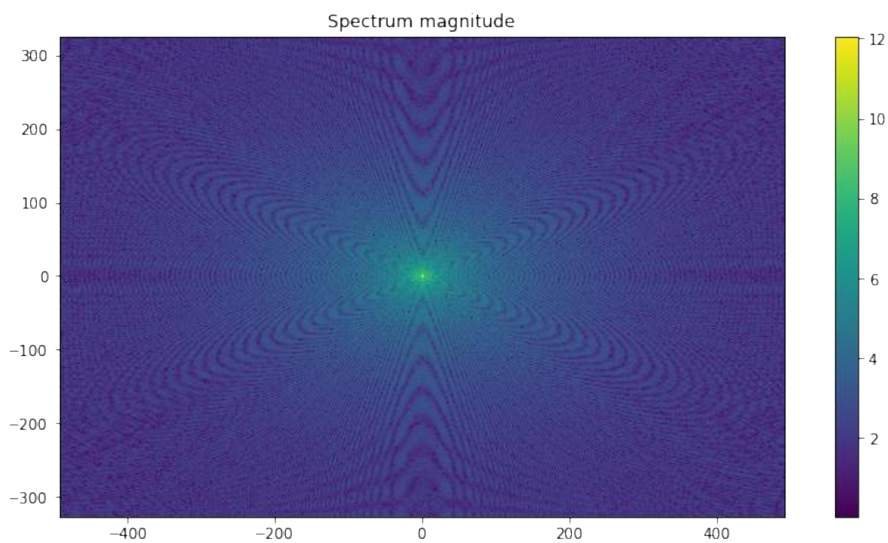
- Sum the three sine signals together into the variable yAll and plot



- Run the FFT of yAll, compute the magnitude and display the results. You can use the e function a3.fft1D Make sure you look at the internals of the code of such function. You should obtain a figure like the following



- Note that in here the plt.stem function is used to plot the results. Check how the Fourier transform shows symmetric negative frequencies, this is normal. You will have to slice the arrays returned by a3.fft1D in order to display the frequencies as above.
- Load 'data/68_left.jpeg', convert to grey scale, resize to 20% of its original resolution and store the image into a variable called imgRet68. Compute the 2D FFT and display the magnitude component.
- The computation and display of 2D FFT can be done by calling a3.plotFFT2d



Convolution and Edge Detection (3 points)

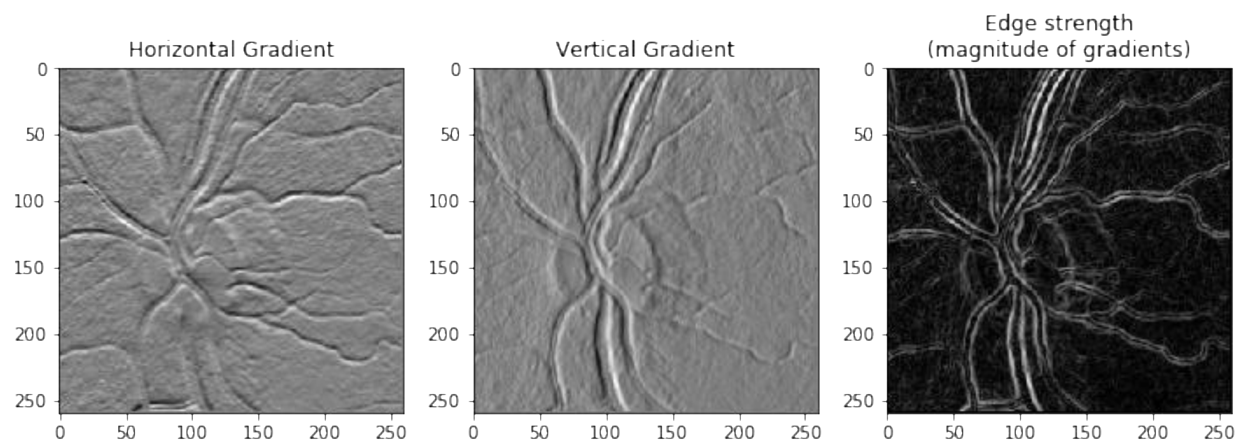
- Extract an area close to the optic nerve in imgRet68 as follows:

```
# 20% of height
borderPx = int(imgRet68.shape[0] * 0.20)
# coordinate of ON
coord = (313, 357)
# slice image
imONest = imgRet68[coord[0]-borderPx:coord[0]+borderPx, coord[1]-borderPx:coord[1]+borderPx]
```

- Compute the horizontal and vertical gradient images using skfl.sobel_h and skfl.sobel_v
- Compute the magnitude of the edges (i.e. the edge strength) using

$$|L_{xy}| = \sqrt{L_x^2 + L_y^2}$$
$$L_x = \frac{\partial L}{\partial x} \quad L_y = \frac{\partial L}{\partial y},$$

- |L_xy| is the magnitude of the edges, L_x is the horizontal gradient and L_y is the vertical gradient. All of the variables are 2D images.



- Compute the convolution on imgRet68 by manually defining the horizontal Sobel Filter as follows:

```
sobelHfilt = np.array([[ 1,   2,  1],
                       [0,   0,  0],
                      [ -1, -2, -1]]) / 4.0
```

  and using the scipy.ndimage.convolve function
- You should get a result that is visually identical to the first figure on left above

1 BONUS POINT:

- Measure if there is any difference between the two horizontal gradient images (i.e. the one computed with the sobel_h and convolve functions)

Noise (3 points)

- Generate images with Gaussian and Salt and Pepper noise as follows

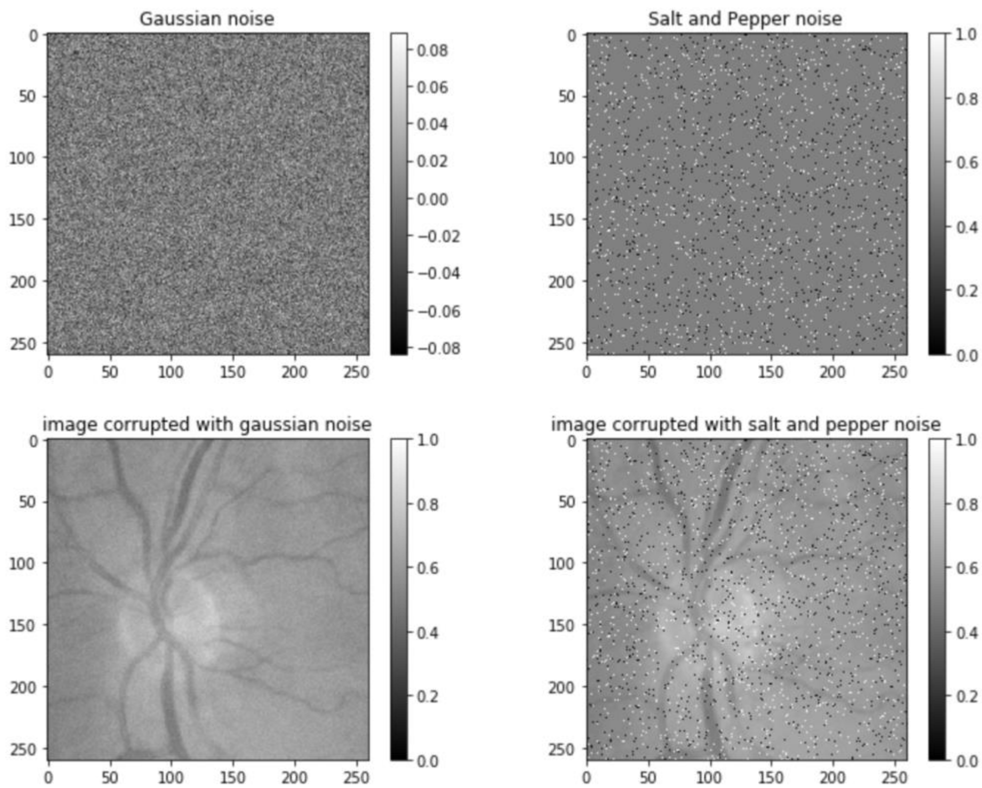
```
# Gaussian noise example
noiseGauss = np.random.normal( np.zeros(imONest.shape ), scale=0.02 )
# Salt end pepper noise (example)
noiseSalt = np.random.normal( np.zeros(imONest.shape ), scale=1 ) > 2
noisePepper = (np.random.normal( np.zeros(imONest.shape ), scale=1 ) > 2) * -1
noiseSalAndPep = (((noiseSalt + noisePepper) / 2.) + 0.5)
```

- Add noise to imgRet68 (for example):

```
# add salt and pepper
imONestNoiseSP = imONest.copy()
noiseSalAndPepLbl = (noiseSalAndPep > 0.9) | (noiseSalAndPep < 0.1)
imONestNoiseSP[noiseSalAndPepLbl] = noiseSalAndPep[noiseSalAndPepLbl]

#FOR YOU TO COMPLETE: add gaussian noise
# imONestNoiseGauss =  ?
```

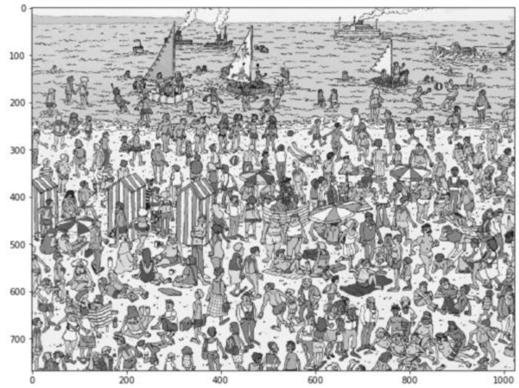
- Plot the resulting images

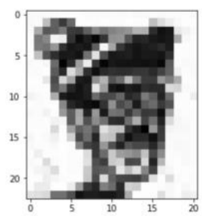


- Try to remove the noise using a Gaussian filter (see skfl.gaussian)
- Try to remove the noise using the median filter
- The median filter can be called as follows:
    - resIm = skfl.median (sk.img_as_uint(image), skmr.square(3))
    - It needs an image represented as integers (as you can see from the code). So remember to turn the result back to float (with sk.img_as_float)

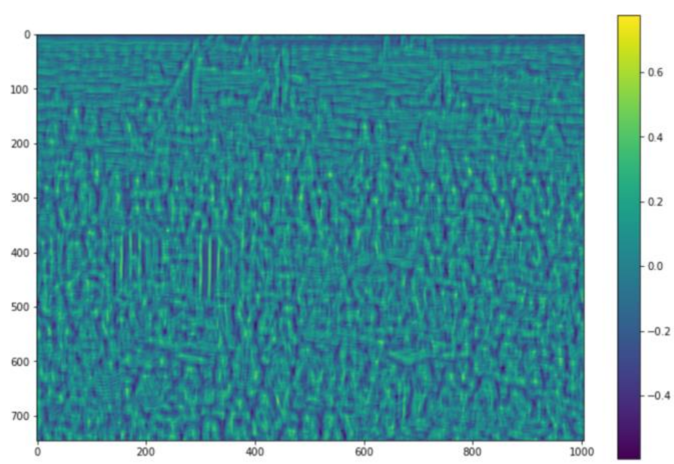
Template Matching (1 point)

- Now, we will try to find Wally on the beach! ☺
- Load 'data/whereiswally.jpg' and 'data/wally_template.png'. The former will be the image of the beach and the latter our Wally template
- Convert to grayscale

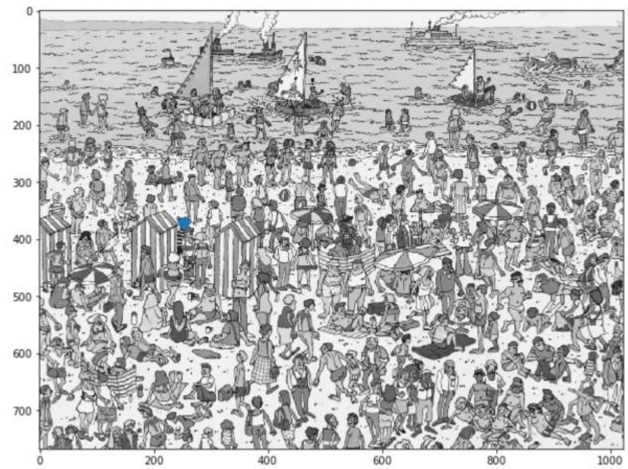




- Use skft.match_template to cross-correlate Wally with the beach image. This should be your resulting image



- Find the highest point in the image and plot Wally's location!



Submission

- Now zip the folder containing your code and the notebook file as you did in assignment 01
- Upload yourname-assignment03-nb.html and yourname-assignment03-src.zip to canvas before the deadline
- Check the readings for more information about the tools you have just installed