Task 1

Filtering and Edge Detection

Team members:

NAME	SECTION	BN
Abram Gad	1	1
Mena safwat	2	44
Mahmoud Hamdy	2	24
Mariam Hossam Eldin	2	31
Mariam mohammed Mahmoud	2	34

Contents:

GUI:

Tab1:

The user can browse for image and choose between three options, first one is adding noise to the image second is apply one of three filters, final option is to choose from 4 types of edge detector

Tab2: Here we browse the image and perform the following on it Drawing histograms and histogram equalization, Normalizing the image, Thresholding algorithms, conversion to gray scale

Tab3: We use it to make a hybrid image

The user will browse for two images as an input, the program will then apply low and high frequency filters on the images such that we get both low and high frequency components one from each image, these frequency components will be combined to form new hybrid image.

Code Functionalities:

1)Add additive noise to the image.

2)Filter the noisy image.

3)Detect edges in the image.

4)Draw histogram and distribution curve.

5)Equalize the image.

6)Normalize the image.

7)Local and global thresholding.

8)Transformation from color image to gray scale image and plot of R, G, and B histograms with its distribution function.

9) Frequency domain filters (high pass and low pass).

10)Hybrid images.

Filters tap:

After choosing image you need to choose the operation

First operation:

Adding noise -> we have two methods salt & pepper and gaussian

salt & pepper function

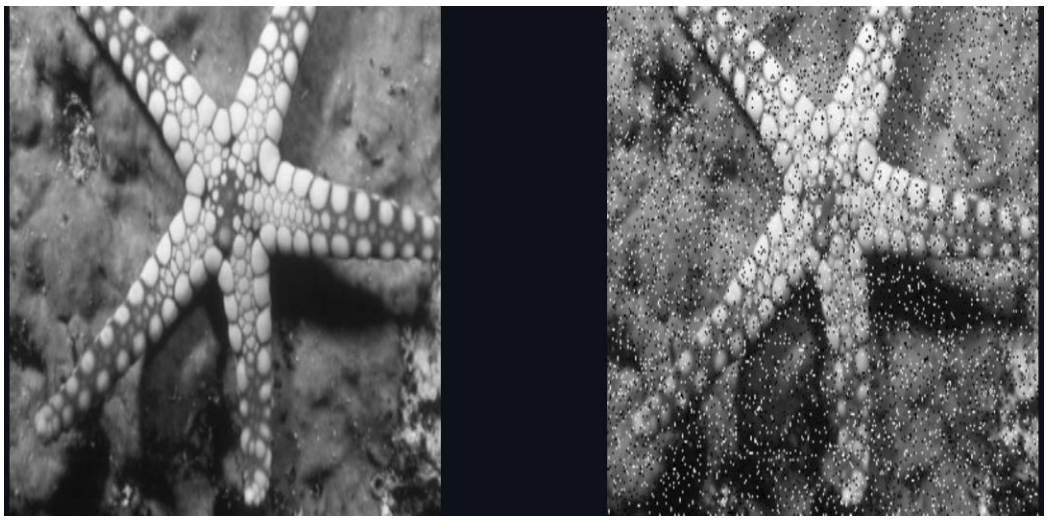
Parameters:

it takes two parameters (NumPy matrix [image you want to process over it], floating number from 0 to 1 [Percentage of noise])

It returns NumPy matrix [result image]

Algorithm:

Create a matrix with the same size of image and iterate over it randomly filling it with wight, black, and image pixels



Gaussian function

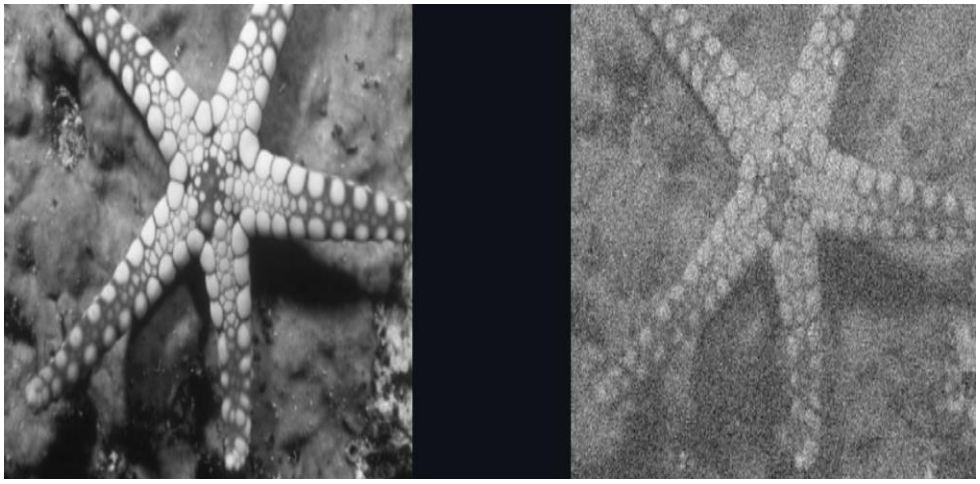
Parameters:

it takes two parameters (NumPy matrix [image you want to process over it], floating number [standard deviation])

It returns NumPy matrix [result image]

Algorithm:

Create a matrix with the same size of image and iterate over it filling it according to gaussian distribution and add it to the image



Second operation:

Applying filter-> we have three types of filters Average filter, gaussian filter and median filter

Average filter function:

Parameters:

it takes two parameters (NumPy matrix [image you want to process over it], integer number [kernel dimension])

It returns NumPy matrix [result image]

Algorithm:

Create a kernel of ones and divide it over the number of elements with the dimension given and apply convolution between the image and the kernel.

Gaussian filter function:

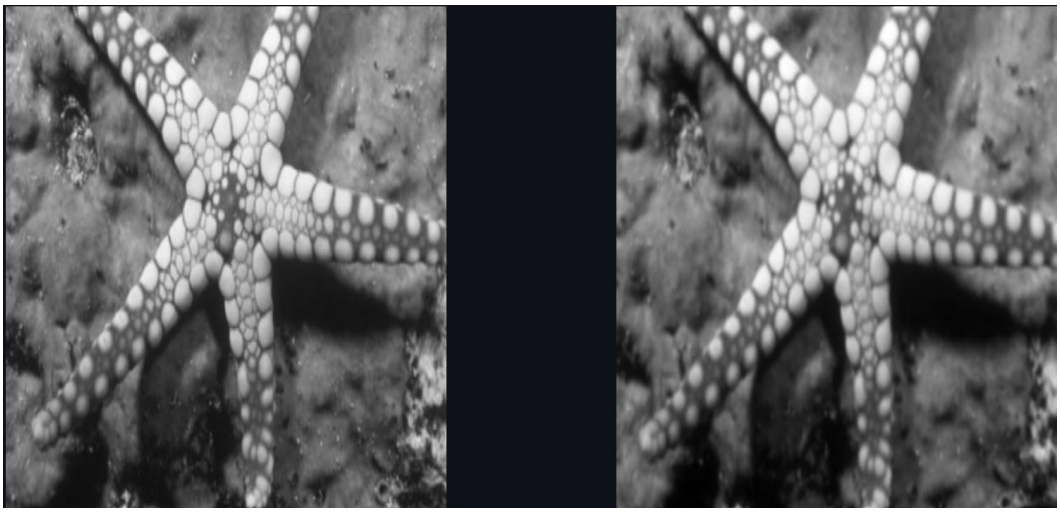
Parameters:

it takes two parameters (NumPy matrix [image you want to process over it], integer number [kernel dimension])

It returns NumPy matrix [result image]

Algorithm:

Create a kernel of gaussian distribution with the dimension given and apply convolution between the image and the kernel.



Median filter function:

Parameters:

it takes two parameters (NumPy matrix [image you want to process over it], integer number [kernel dimension])

It returns NumPy matrix [result image]

Algorithm:

Create a kernel of ones with the dimension given and apply convolution between the image and the kernel but after convolution did not store the sum its store the median of elements.

Third operation:

Applying edge detector -> we have four types of edge detector Prewitt Operator , Sobel Operator, Roberts Operator and Canny Operator.

Prewitt, Soble, Robert's function:

Parameters:

It takes one parameter (NumPy matrix [image you want to process over it])

It returns NumPy matrix [result image]

Algorithm:

Create a kernel of suitable operator and apply convolution between the image and the kernel.

Canny function:

Parameters:

It takes one parameter (NumPy matrix [image you want to process over it])

It returns NumPy matrix [result image]

Algorithm:

1- applying gaussian filter

2- applying Soble operator

3- applying Non maximum suppression

4- applying double thresholding

5- hysteresis effect

Histograms tap:

As we imported the image, we have many choices in this tab these choices are (Drawing histograms and histogram equalization, Normalizing the image, Thresholding algorithms, conversion to gray scale), all the functions are implemented in **Histograms.py**

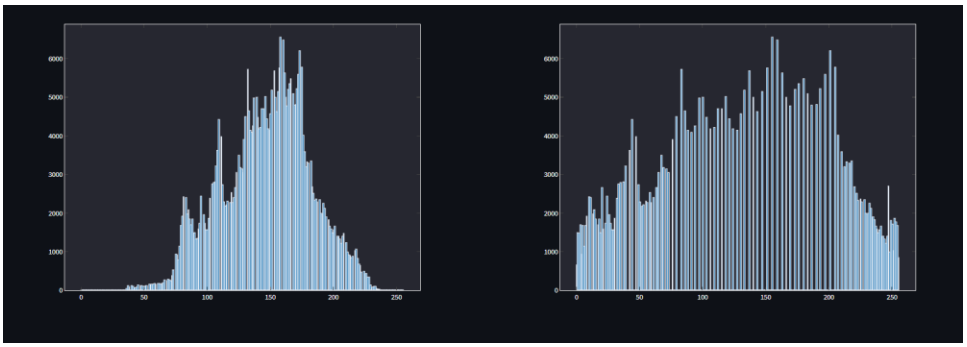
- **Drawing histograms and histogram equalization Option:**

=> In this option we Compare The original image with the output image of histogram equalization and their histograms

The algorithm works as follows: We first Calculate the histogram array using the function `get_histogram_array(array, max_val)` which iterates over the image to calculate different frequencies of image's intensities then we calculate the cumulative distribution of the pixel values using the function `cumulative(array, shape)`

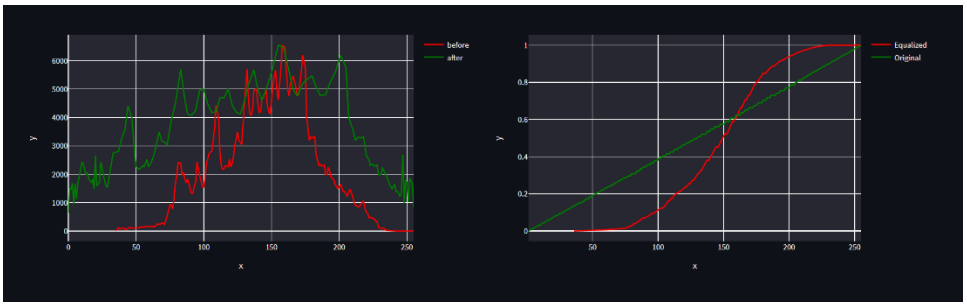
Which depends on calculating Probability distribution function (PDF) after calculating PDF we sum each pixel value intensity with its previous values after that we multiply their values with the maximum value of the pixel we then round the values to the nearest integer after that we map the new values to their frequencies

Comparison between the original image and the equalized image



Comparison between the original image's histogram and the

Equalized image's histogram



The Original and Equalized image's distribution and cumulative curves

- **Normalizing the image:**

  In this function we subtract the min intensity from the whole intensities then divide them by the (max intensity – min intensity) and multiply them back by 255 like in function `Normalize(image)`

Figure 2 Normalize VS Normal Image

Figure 1 OpenCV Normalize

- **Thresholding:**

This is a technique used to partition the object of an image from the background There's 2 types Local and Global Thresholding

1st) Global Thresholding: This method is based on mean method of thresholding which uses the mean to split both the object and it's background as in

```
Thresholding(array, high, low, thresh=0) function
```

2nd) Local Thresholding: This method divides the image into smaller windows(squares) with a set size and at each window we call the global thresholding method independently

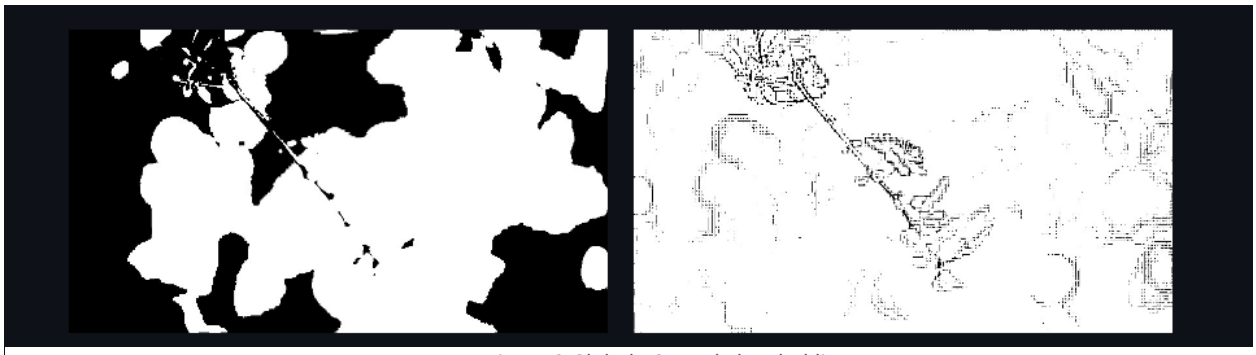


Figure 3 Global VS Local Thresholding

- **Gray Scale Conversion:**

This option transform the RGB image to gray scale image by multiplying each of the RGB component in the image by a certain value (for Red we multiply by 0.299 and Green by 0.587 and Blue by 0.114 as in the function `ToGrey(image)` and

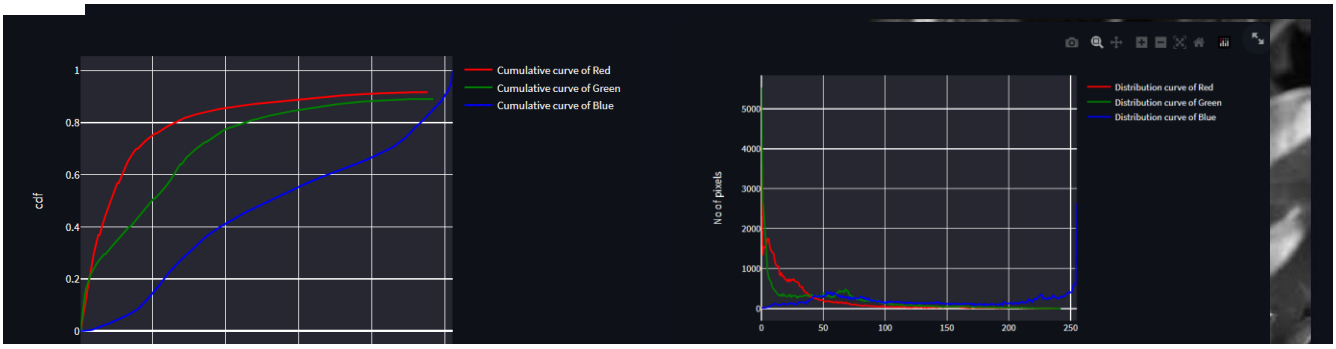


Figure 4 cdf

6 Gr

Figure 5 Distribution function

after the conversion we draw the RGB's histograms and their cumulative and distribution curves.

.

Frequency tap:

Here we will explain how to get hybrid images from two input images and each function to complete the process.

Algorithms implemented:

- Hybrid image
- Low pass filter
- High pass filter

Starting from reading the two input images and passing them with the sigma value to **fft_hyprid_image()**, where it first resizes both images to match to be able to perform summation of the two of them, then it passes the images to the low and high pass filters functions (fft_low_pass(), fft_high_pass()) that will filter the images returning both low and high frequency components , then these components will be added to get the output image(Hybrid image).

The inputs: Two input images

Input images

Expected output:

Output image

For the low pass filter:

In **fft_low_pass(),** it takes the image and sigma value then apply Fourier transform to the image and then send the image and sigma value to **`create_gaussian_filter()`**, where here it substitutes in this formula `np.exp(-((Y-center[0])**2+(X-center[1])**2)/(2*sigma**2))`and return the kernel that will be then multiplied by the Fourier of the image , thee output of this multiplication is the low pass filtered image but in frequency domain so taking inverse Fourier of it will give us the final desired output .

Low pass output:

Low pass image

The low pass image doesn’t show edges of the image (high frequency component)

Note: sigma is a parameter that controls the contribution of low and high frequency components.

For the High pass filter:

In **fft_high_pass(),** it has the same working idea as low except that here we want simple the opposite (high frequency components) for that the only difference is the kernel, which will be the remaining after subtraction low pass kernel from 1.

High pass output:

High pass image

The high pass image shows the structure of the image