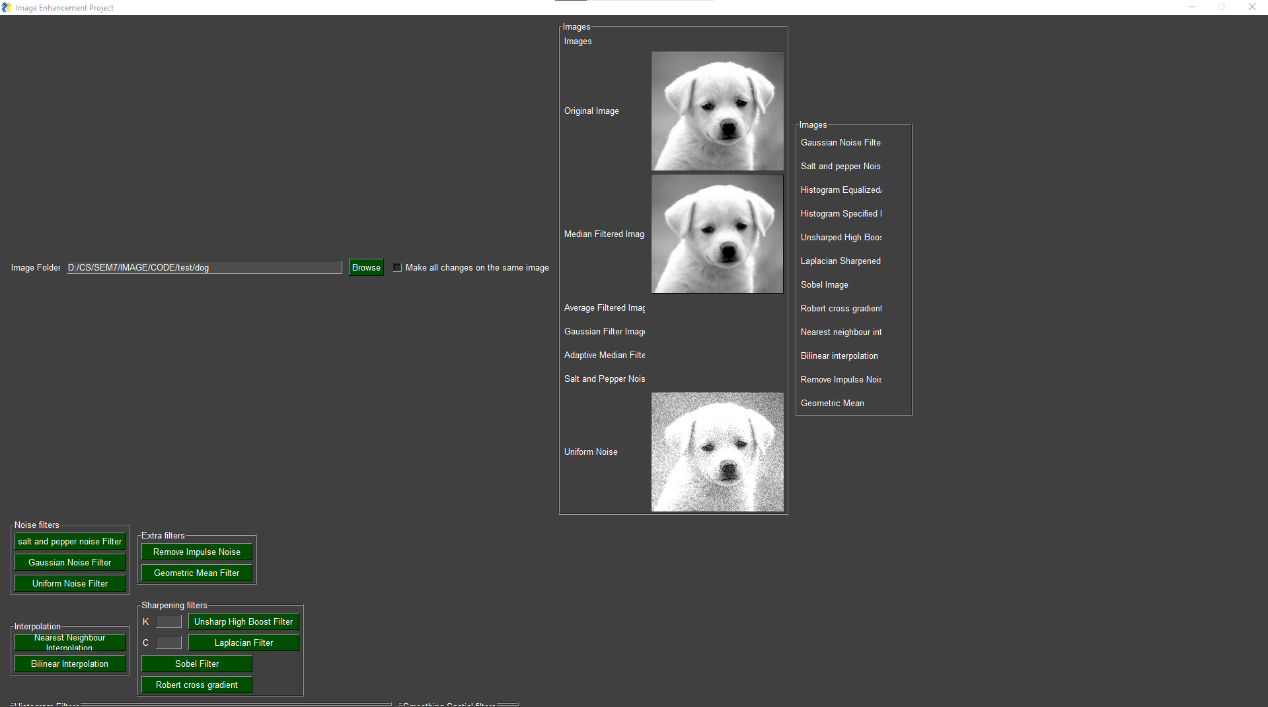


This application is an image enhancement application using multiple filters. To use any filters, click on the filters button and the filter will be triggered. Also, you can choose between applying all changes on the same image or do each filter separately eg:

In both images below uniform noise was added first, then the medina filter



Here we unchecked the box, so all the filters work on the original image.

Text

Description automatically generated

Where here we checked the box, so the filters worked after each other on the same image

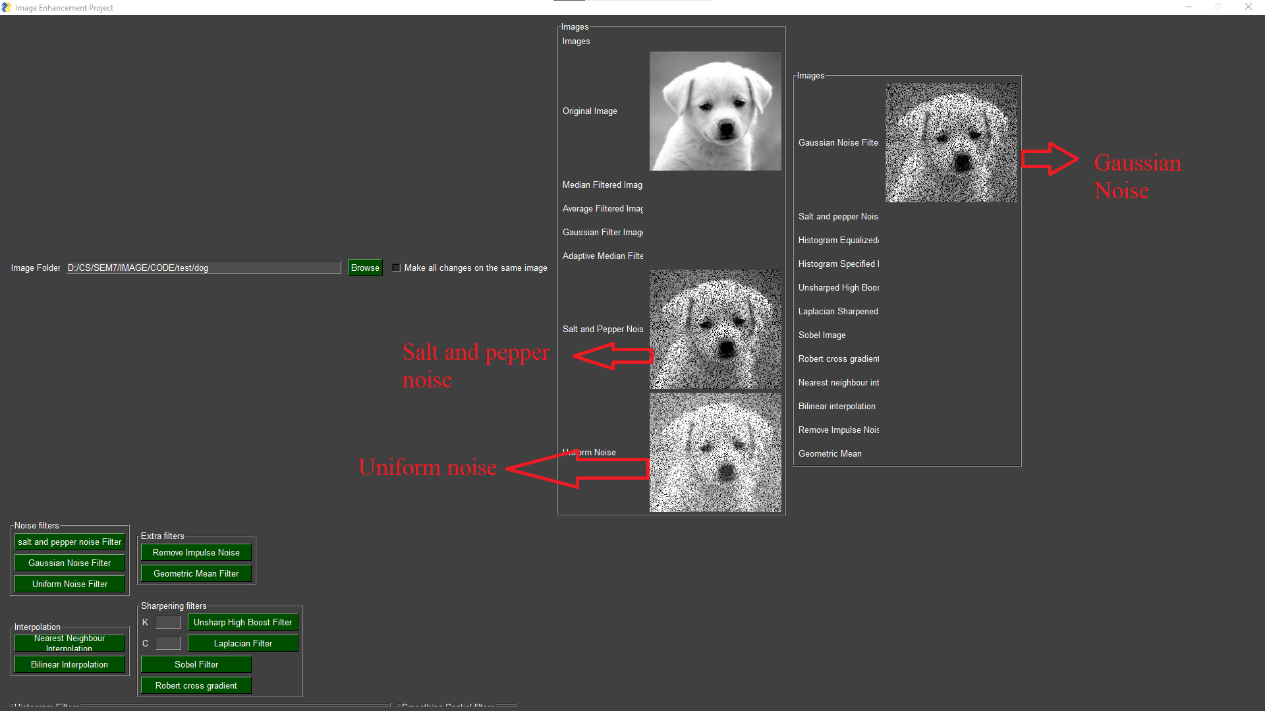
The used filters are:

* **Noise filters**
* **Extra filters**
* **Interpolation**
* **Sharpening filters**
* **Histogram filters**
* **Smoothing filters**
* **Fourier transform filter**

In order to use the application an image path is needed, by clicking browse you can choose the folder containing the image.

Also note that the kernel size used in all filters is a [3x3] kernel

**Noise Filters**

****

These filters are used to add noise to the image. The First step in all the filters is dividing image intensities by 255 so the intensities vary from (0,1) then we add the filters then multiply the intensities again by 255

**Salt and pepper**

Salt and pepper algorithm works as follow:

An image is created with the same size of the image, and a random function is called to determine where and how many black, and white pixels added. After that the noise is added to the image

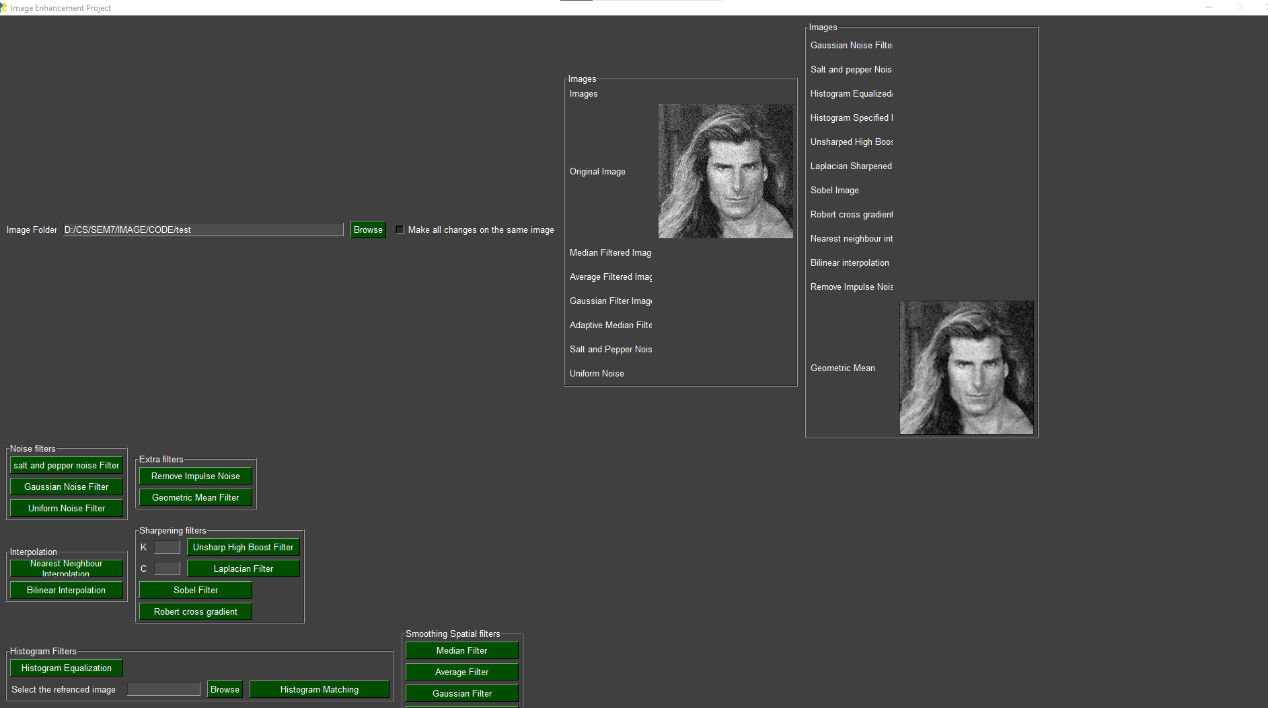
**Gaussian noise**

An image is created with the same size of the original image, a random function is called to determine where and how many noise is added. But unlike salt and pepper noise the range of colours is determined by mean as lower bound (which we set to 0), and the standard deviation as the upper bound. Then this noise is added to the image

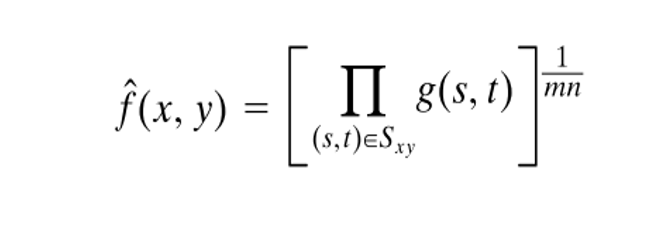
**Uniform noise**

An image is created with the same size of original image, a random function generates noise varying from (0,0.4) to add uniform noise. Then the noise is added to the image.

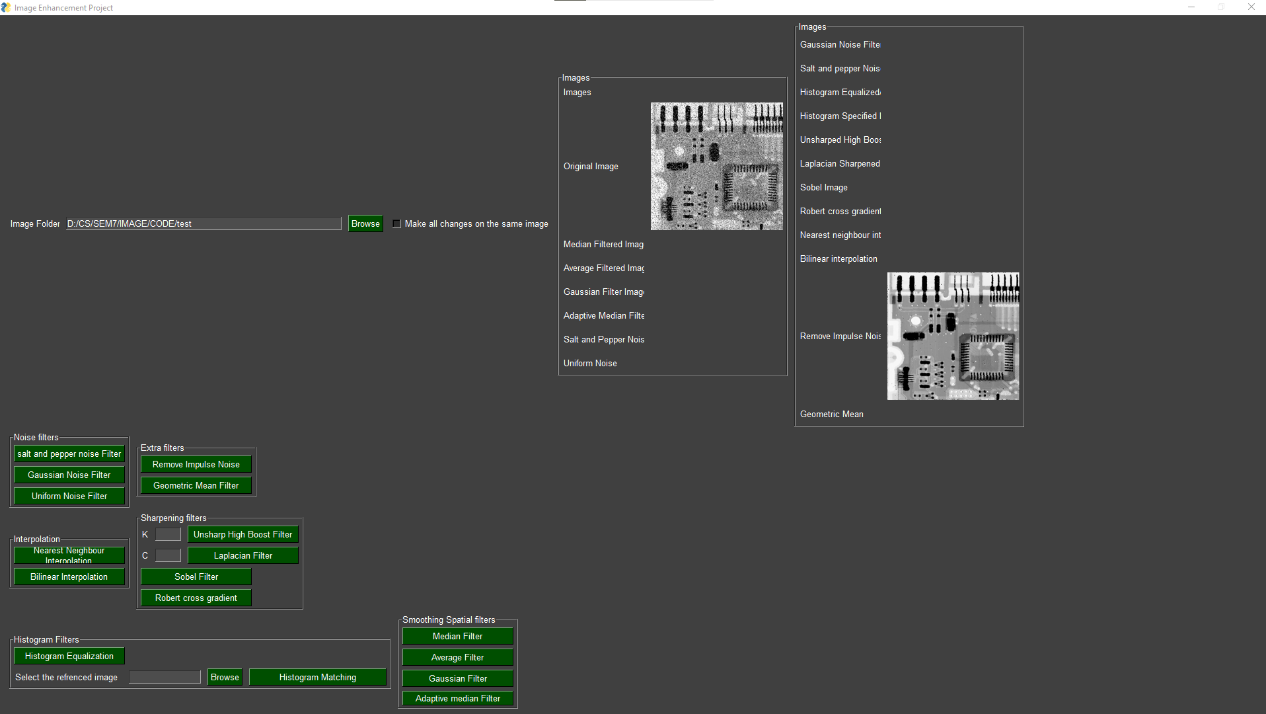
**Extra Filters**

****

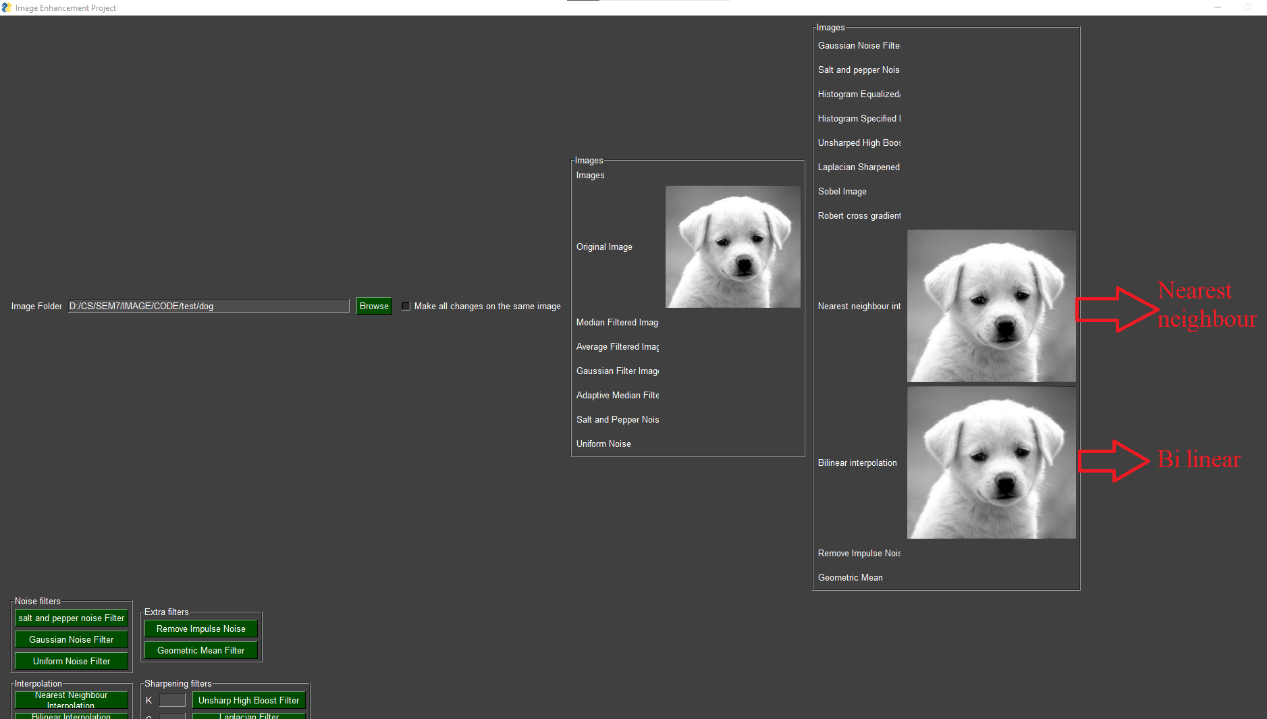
**Geometric mean**

The geometric mean filter works best with gaussian noise images. The filter works by multiplying the neighbourhood intensities with each other. Then square roots the product by the number of intensities.

**Removing impulse noise filter**

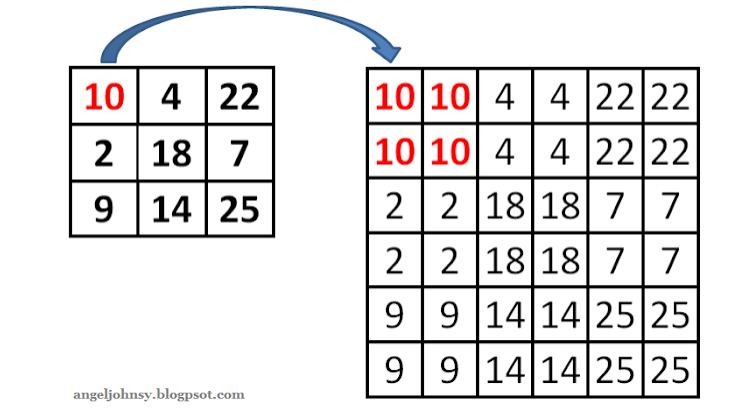
This filter removes impulse noise using median filter.

Its basically a median filter

**Interpolation**

**Nearest neighbour**

This technique matches the missing pixels by looking at their next neighbour and giving them the same pixel as their nearest neighbour.



**Bilinear**

This technique gives the missing pixel the values according to it’s 4 neighbours. And calculate the distance in x-axis and y-axis to get this pixel.

**Histogram**

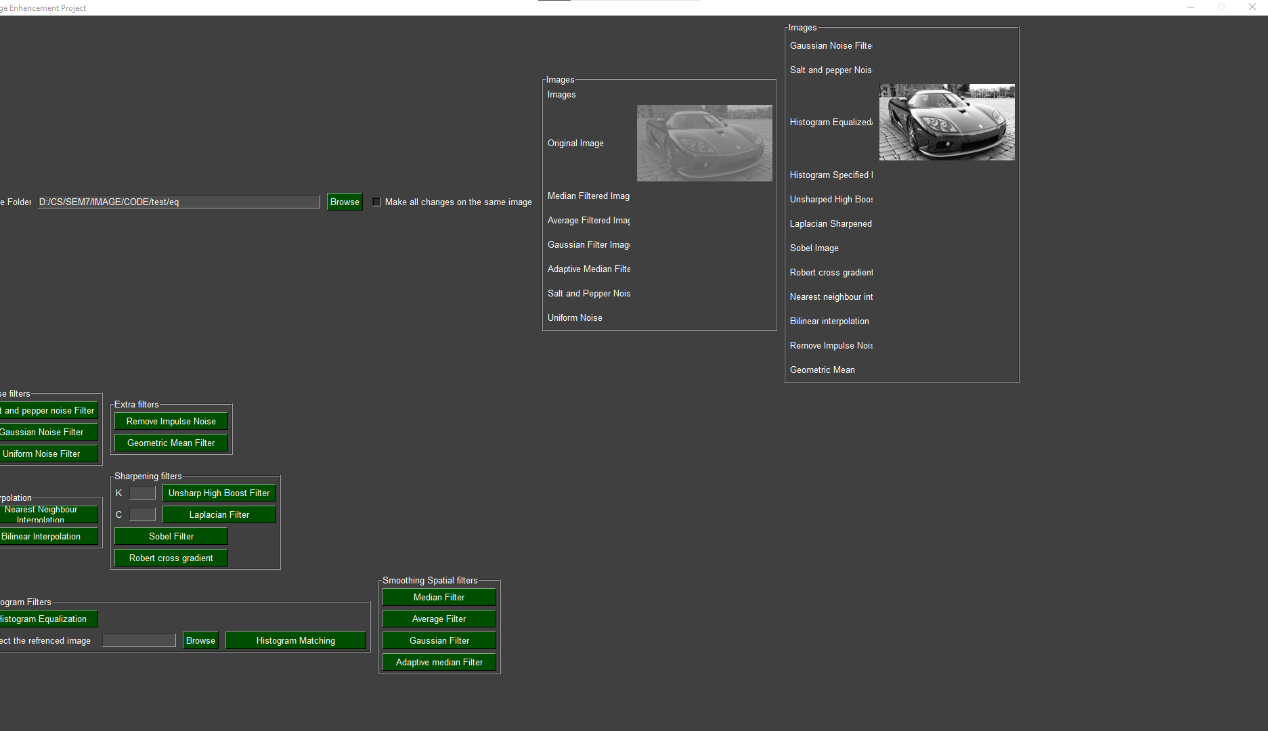
**Histogram equalization**Text, whiteboard

Description automatically generated

Histogram equalization works as follows:

First calculate the frequency of each intensity level,

Then creates a transformation function for all

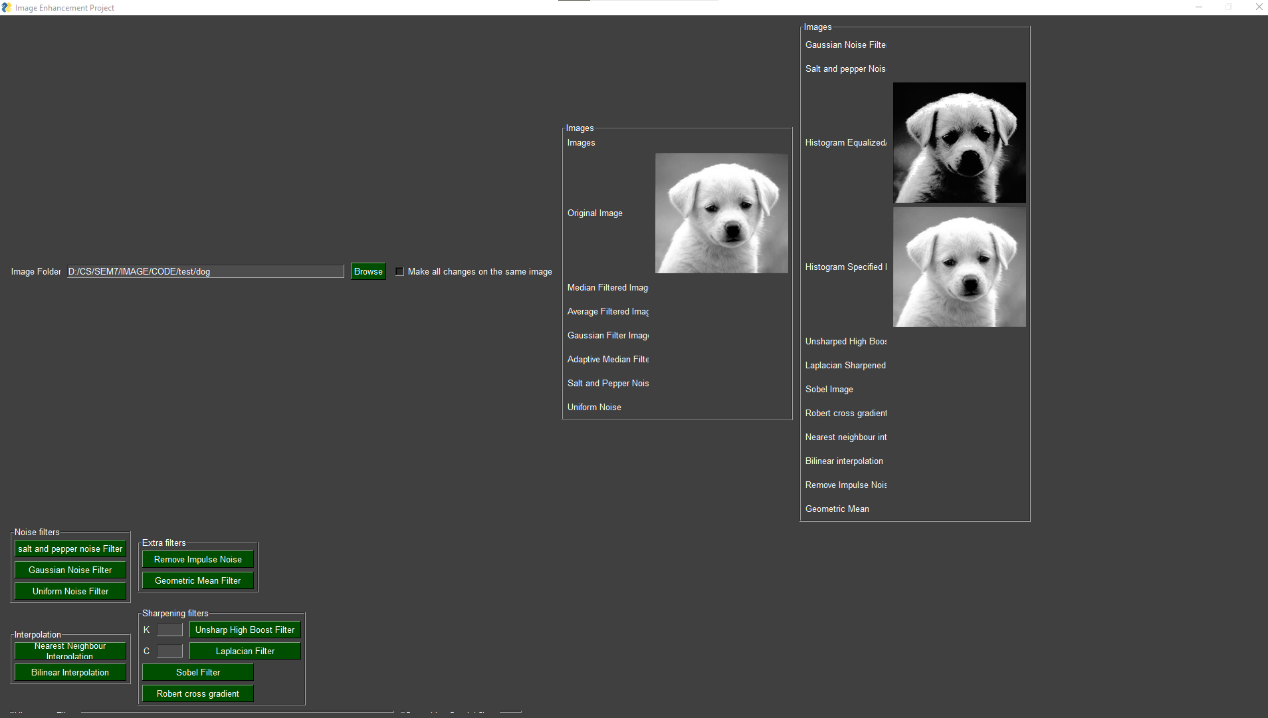
intensities. The last step is matching the intensities to its new positions 

**Histogram matching**

In this technique. We obtain the transformation from 2 images (image, reference image) hence the user is required to enter the location of another picture. The transformation function is obtained for the 2 images then the transformation function of the image is mapped to the transformation function of the referenced image

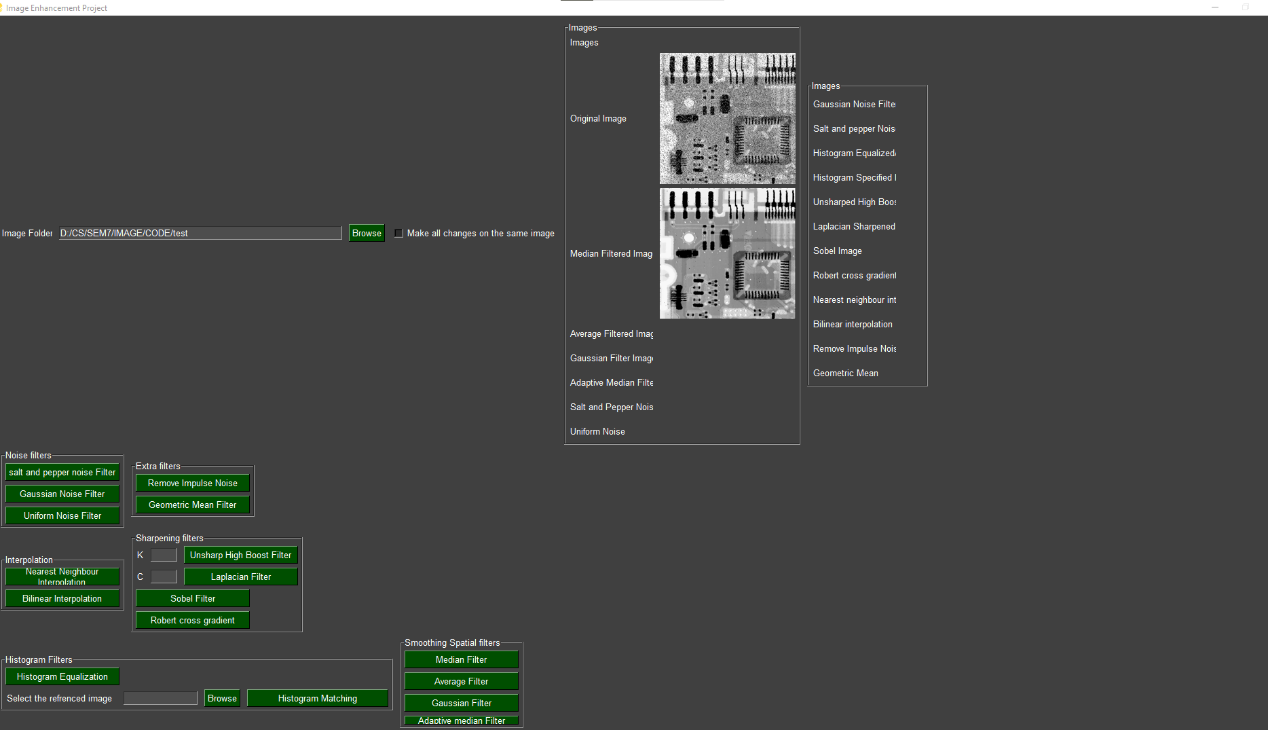
Diagram

Description automatically generated



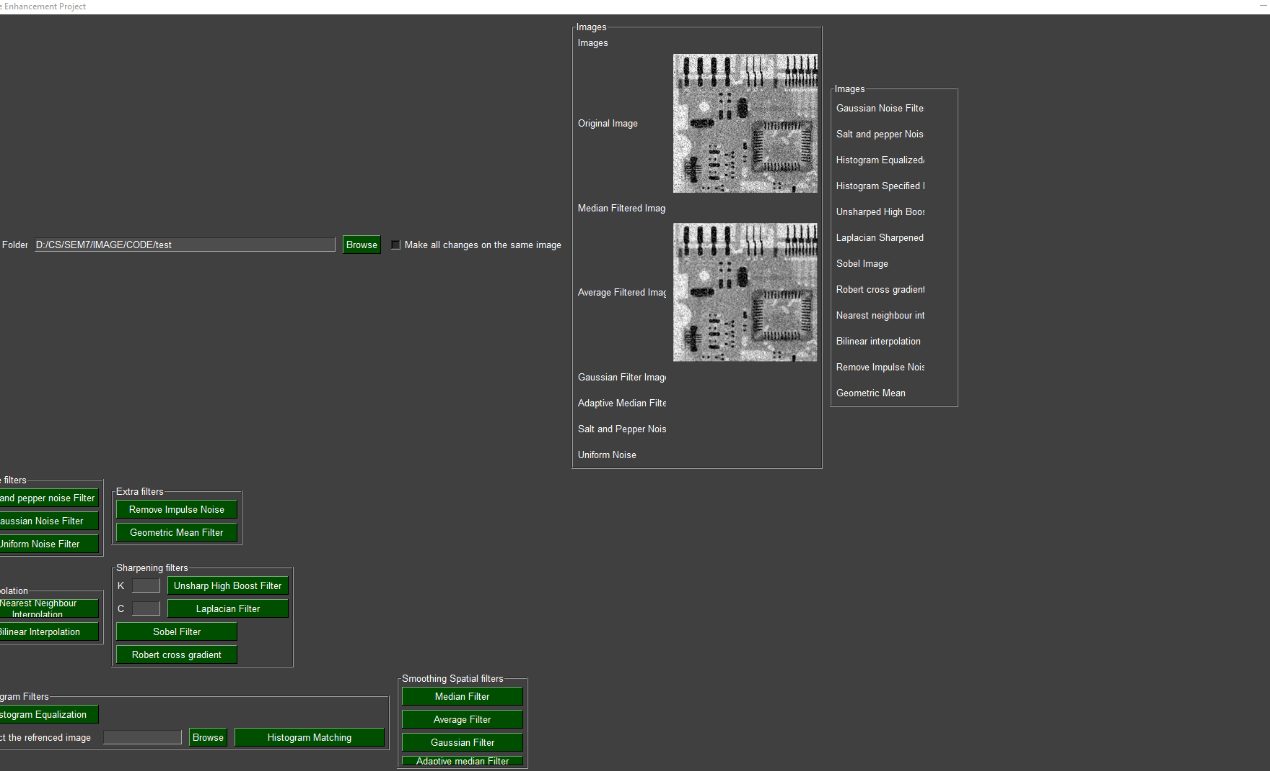
**Smoothing Filters**

**Median filter**

****

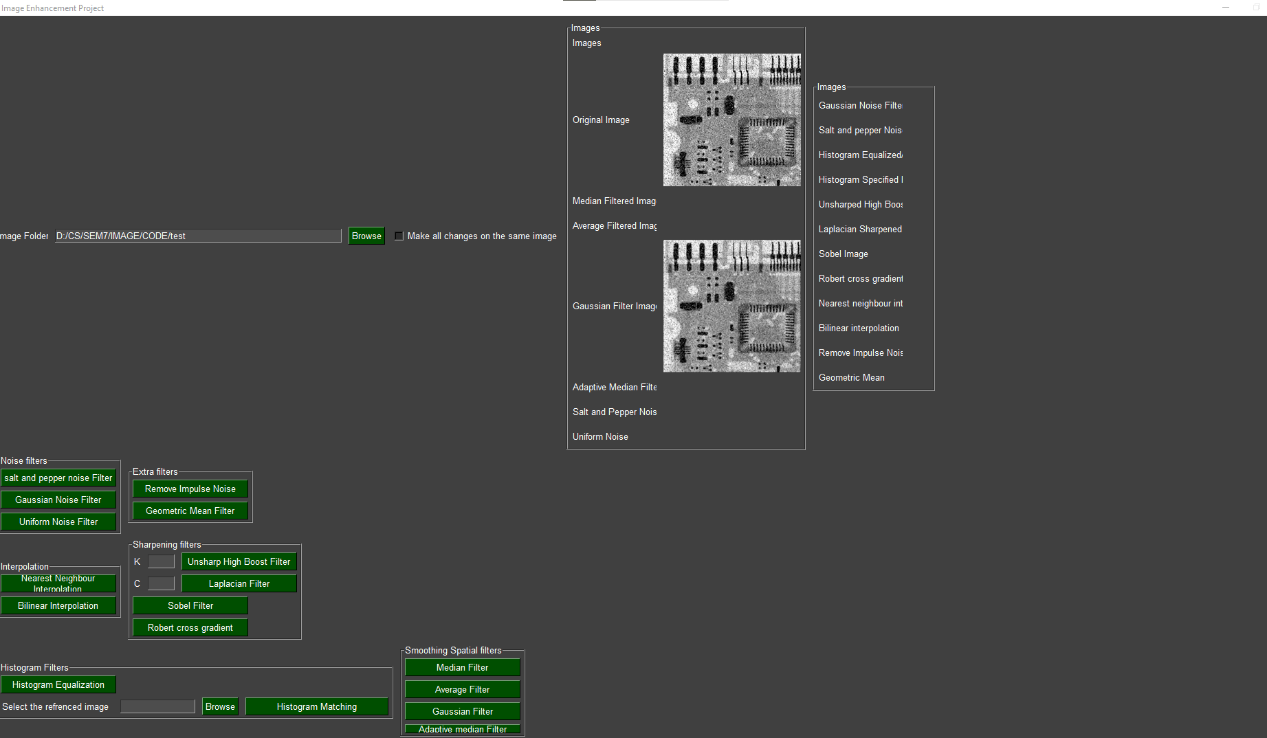
Median filter calculates the median of the neighbourhood and update the pixel with it.

**Average filter**

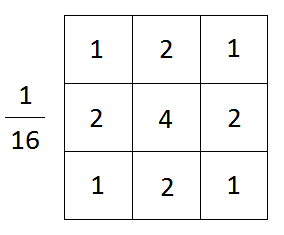
****

Multiply neighbourhood by 1/9 and sum it and update pixel with it

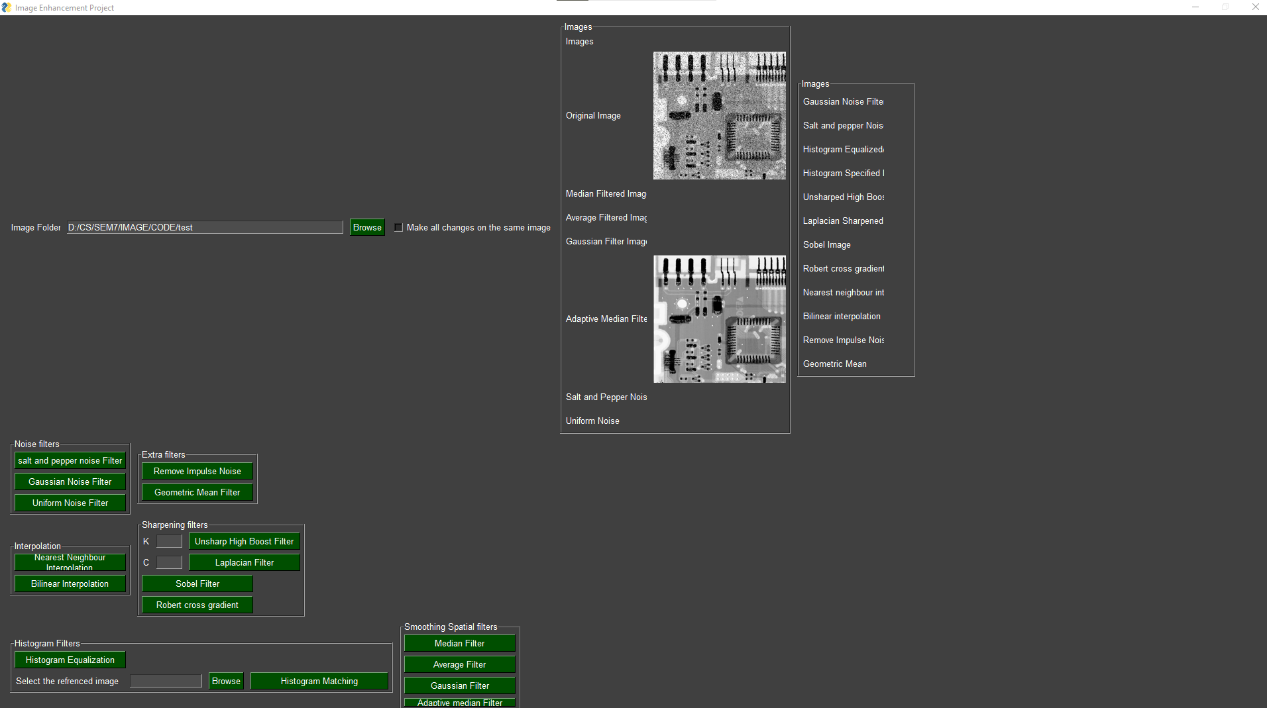
**Gaussian filter**

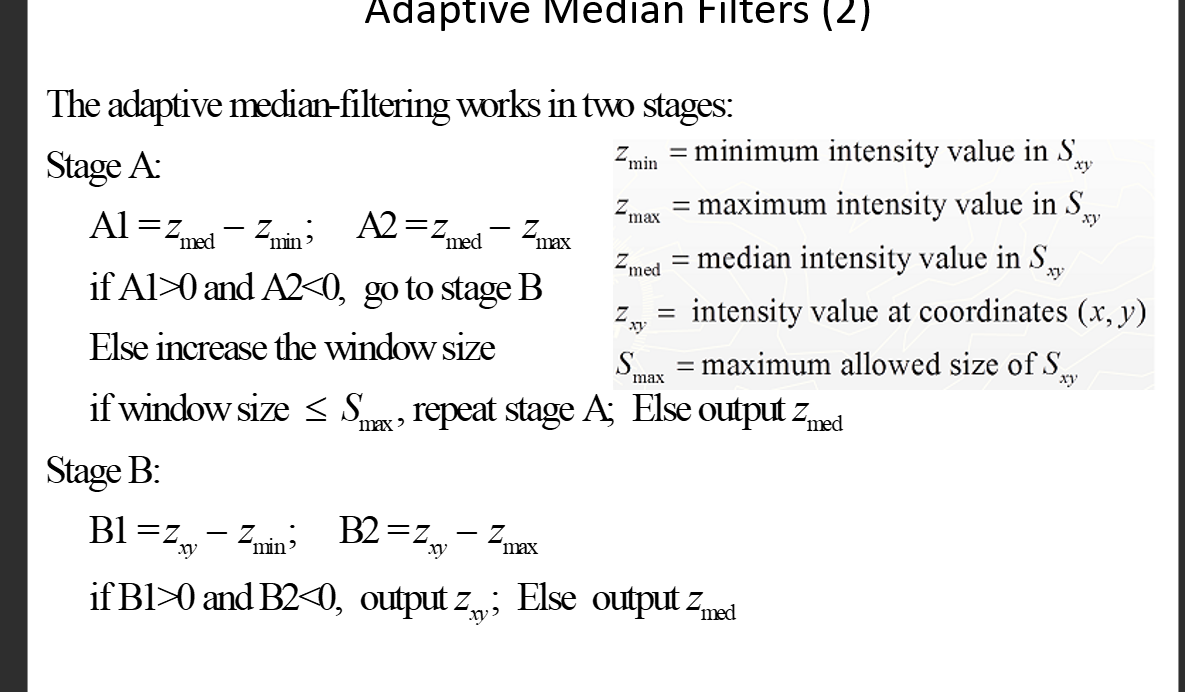
****

A picture containing chart

Description automatically generatedThe gaussian filter uses this equation in order to blur the image, however as this equation is hard to calculate for each pixel we use the approximation matrix for the 3x3 kernel which is and multiply the neighbourhood by it and sum them and update the pixel with the sum

**Adaptive mean filter**

****



This is the summery of how this filter works at every point we calculate the minimum intensity of neighbourhood, max intensity, and median and apply these conditions to it. Note that the maximum allowed size in this application is 7x7 kernel.

**Sharpen filter**

Graphical user interface, application

Description automatically generated

Graphical user interface, application

Description automatically generated

**Laplacian**

Some of the laplacina Kernels that were used in order to be convoluted with the image.

laplacian=np.array([[0,1,0],

          [1,-4,1],

          [0,1,0]])

another\_laplacian=np.array([[-1,-1,-1],

          [-1,8,-1],

          [-1,-1,-1]])

The Laplace formula used.

A picture containing text

Description automatically generated

Where g(x,y) the ouput image f(x,y) input image and c is a constant either 1 or -1 depends on the mask.

**Robert Cross Gradient**

Mx =np.array([[1,0],

          [0,-1]])

My = np.array([[0,1],

          [-1,0]])

2 masks r convoluted with the image then the square root were taken from the sum of each output power 2. Then a minimum threshold were taken to just allow high frequencies to pass then the result were added to the original image to apply sharpening.

for x in range(Robert\_img.shape[0]):

      for y in range(Robert\_img.shape[1]):

        output[x][y]= sqrt((Robert\_img[x][y]\*\*2)+(Robert\_img1[x][y]\*\*2))

        if output[x][y]>threeshold:

          output[x][y]=0

**Sobel**

A kernel was convoluted with the original image to get a mask then then mask was subtracted from the original image to get the sobel sharpened images

kernel\_1=np.array([[-1,0,1],

          [-2,0,2],

          [-1 ,0,1]])

kernel\_2= np.array([[-1,-2,-1],

          [0,0,0],

           [1,2,1]])

Sobel\_img = convolve2d(img, kernel=kernel\_1,c=1,kern\_size=3)

cv2\_imshow(Sobel\_img)

sharpened= img-Sobel\_img

**Unsharp - High boost**

mask = img - blured\_img

Unsharp\_Highboost = img + k\*mask

A smart way to apply un-sharping or high boost filtering depends on the value K. where k=1 unsharp and K>1 high boost filtering.

A picture containing text, spectacles

Description automatically generated

Edges r detected through subtracting original imag from the blured image where high intensity edges r vanished then the high intensities “edges” are multiplied with a constant to either give it a high priority “high boost” or stayed the same to unsharp the image

A picture containing text

Description automatically generated

**Fast Fourier Transform**

A divide and conquer Technique to scan even verse odd elements and implement the formula to achieve BIGO(NLOGN) then It was shifted to collect the high frequencies from the edges to the DC then a log transform was implemented to impower the week parts.

**Text, letter

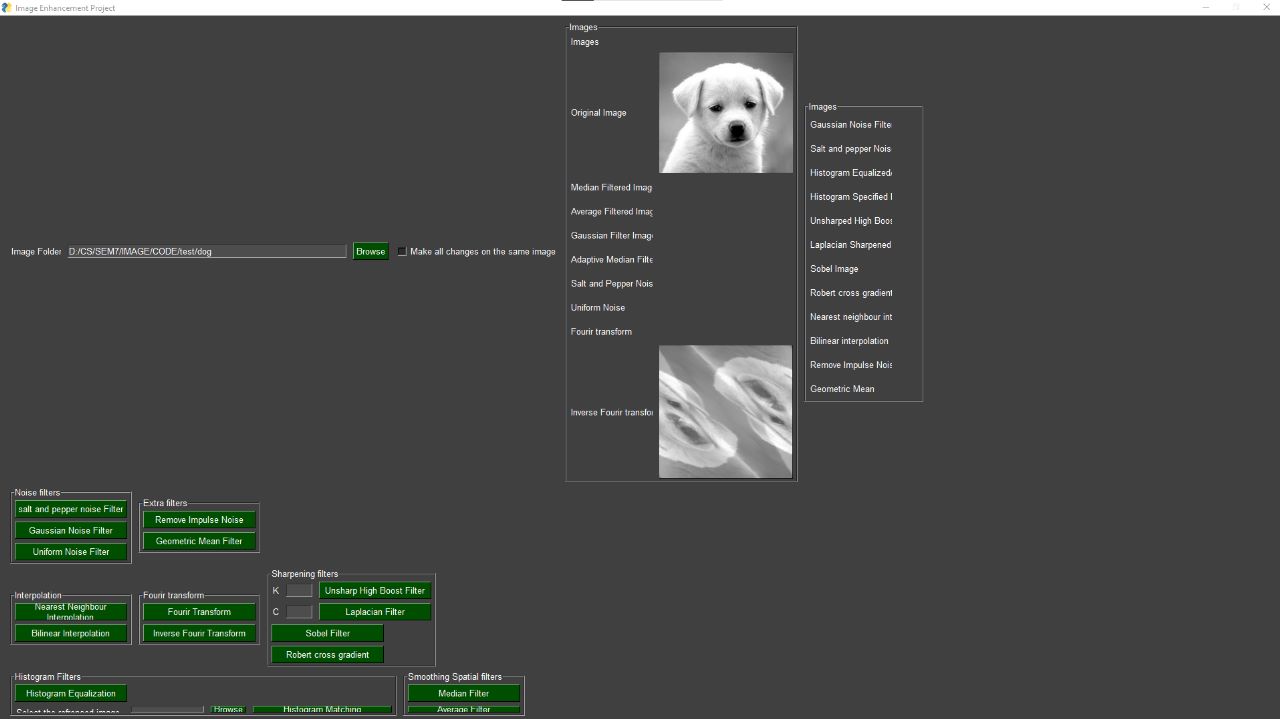
Description automatically generatedText

Description automatically generated**

Graphical user interface

Description automatically generated

**Inverse FFT**

A recursive function to implement inverse Fourier using the formula making advantage of divide and conquer strategy. The user selects the image then it is getting mapped to its original coordinates.

Text

Description automatically generated