IMAGE PROCESSING TECHNIQUES USING OPENCV SOFTWARE

REVIEW-2

Group ID-6

COURSE CODE : CSE4019

COURSE TITLE : IMAGE PROCESSING

SLOT : G2

Under the Guidance of

SANTHI V

GROUP MEMBERS

Eashan Kaushik 17BCE2031

Tanay Nileshkumar Nagarsheth 17BCE2230

Samarth Chauhan 12BCE0283

Prakhar 17BCE2251

ABSTRACT:

In this project we will perform various image processing techniques on certain input images using OPEN CV software the given processes are given below:

1. Smoothing Images

Smoothing an Image. Smoothing is often used to reduce noise within an image or to produce a less pixelated image.

2. Image Thresholding

Here, the matter is straight forward. If pixel value is greater than a threshold value, it is assigned one value (white), else it is assigned another value (black). First argument is the source image, which should be a grayscale image. Second argument is the threshold value which is used to classify the pixel values.

3. Transformations

OpenCV provides two transformation functions, cv2.warpAffine and cv2.warpPerspective, with which you can have all kinds of transformations. cv2.warpAffine takes a 2x3 transformation matrix while cv2.warpPerspective takes a 3x3 transformation matrix as input.

4. Morphological Transformations

Morphological transformations are some simple operations based on the image shape. It is normally performed on binary images. It needs two inputs, one is our original image, second one is called structuring element or kernel which decides the nature of operation. Two basic morphological operators are Erosion and Dilation. Then its variant forms like Opening, Closing, Gradient etc also comes into play.

5. Image Gradients

OpenCV provides three types of gradient filters or High-pass filters, Sobel, Scharr and Laplacian. We will see each one of them.

6. Fourier transform (FT)

The Fourier transform (FT) decomposes a function of time (a *signal*) into the frequencies that make it up, in a way similar to how a musical chord can be expressed as the frequencies (or pitches) of its constituent notes. The Fourier transform of a function of time is itself a complex-valued function of frequency, whose absolute value represents the amount of that frequency present in the original function, and whose complex argument is the phase offset of the basic sinusoid in that frequency. The Fourier transform is called the *frequency domain representation* of the original signal.

INTRODUCTION:

Smoothing Images

Smoothing an Image. Smoothing is often used to reduce noise within an image or to produce a less pixelated image. The following methods given below are used for smoothening images:

- **2D Convolution (Image Filtering)**

As for one-dimensional signals, images also can be filtered with various low-pass filters, high-pass filters, etc. A low pass filter helps in removing noise, or blurring the image. A high pass filter helps in finding edges in an image.

As an example, a 5x5 averaging filter kernel is defined as follows:

$$K = \frac{1}{25}\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Filtering with the above kernel results in the following being performed: for each pixel, a 5x5 window is centered on this pixel, all pixels falling within this window are summed up, and the result is then divided by 25. This equates to computing the average of the pixel values inside that window. This operation is performed for all the pixels in the image to produce the output filtered image.

- **Image Blurring (Image Smoothing)**

Image blurring is achieved by convolving the image with a low-pass filter kernel. It is useful for removing noise. It actually removes high frequency content from the image resulting in edges being blurred when this is filter is applied. Some blurring technique are:

1. **Averaging**

This is done by convolving the image with a normalized box filter. It simply takes the average of all the pixels under kernel area and replaces the central element with this average. A 3x3 normalized box filter would look like this:

$$K = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

2. Gaussian Filtering

In this approach, instead of a box filter consisting of equal filter coefficients, a Gaussian kernel is used. In order to apply a Gaussian filter, we need to specify the width and height of the kernel which should be positive and odd. We also should specify the standard deviation in the X and Y directions, sigmaX and sigmaY respectively. If only sigmaX is specified, sigmaY is taken as equal to sigmaX. If both are given as zeros, they are calculated from the kernel size. Gaussian filtering is highly effective in removing Gaussian noise from the image.

3. Median Filtering

This is highly effective in removing salt-and-pepper noise. One interesting thing to note is that, in the Gaussian and box filters, the filtered value for the central element can be a value which may not exist in the original image. However, this is not the case in median filtering, since the central element is always replaced by some pixel value in the image. This reduces the noise effectively. The kernel size must be a positive odd integer.

4. Bilateral Filtering

As we noted, the filters we presented earlier tend to blur edges. This is not the case for the bilateral filter i.e. it is highly effective at noise removal while preserving edges. But the operation is slower compared to other filters. We already saw that a Gaussian filter takes the neighbourhood around the pixel and finds its Gaussian weighted average. This Gaussian filter is a function of space alone, that is, nearby pixels are considered while filtering. It does not consider whether pixels have almost the same intensity value and does not consider whether the pixel lies on an edge or not. The resulting effect is that Gaussian filters tend to blur edges, which is undesirable.

The bilateral filter also uses a Gaussian filter in the space domain, but it also uses one more multiplicative Gaussian filter component which is a function of pixel intensity differences. The Gaussian function of space makes sure that only pixels are ‘spatial neighbours’ are considered for filtering, while the Gaussian component applied in the intensity domain ensures that only those pixels with intensities similar to that of the central pixel are included to compute the blurred intensity value. As a result, this method preserves edges, since for pixels lying near edges, neighbouring pixels placed on the other side of the edge, and therefore exhibiting large intensity variations when compared to the central pixel, will not be included for blurring.

Image Thresholding

1. **Simple Thresholding**

Here, the matter is straight forward. If pixel value is greater than a threshold value, it is assigned one value (white), else it is assigned another value (black). First argument is the source image, which should be a grayscale image. Second argument is the threshold value which is used to classify the pixel values. Third argument is the maxVal which represents the value to be given if pixel value is more than (sometimes less than) the threshold value

2. **Adaptive Thresholding**

In simple thresholding technique, we used a global value as threshold value. But it may not be good in all the conditions where image has different lighting conditions in different areas. In that case, we go for adaptive thresholding. In this, the algorithm calculates the threshold for a small regions of the image. So we get different thresholds for different regions of the same image and it gives us better results for images with varying illumination.

3. **Otsu's Binarization**

In global thresholding, we used an arbitrary value for threshold value. How can we know a value we selected is good or not to solve this ambiguity we use the concept of bimodal image. Bimodal image *is an image whose histogram has two peaks*. For that image, we can approximately take a value in the middle of those peaks as threshold value. This is what Otsu binarization does. So in simple words, it automatically calculates a threshold value from image histogram for a bimodal image but for images which are not bimodal, binarization won't be accurate.

Transformations

OpenCV provides two transformation functions, cv2.warpAffine and cv2.warpPerspective, with which you can have all kinds of transformations. cv2.warpAffine takes a 2x3 transformation matrix while cv2.warpPerspective takes a 3x3 transformation matrix as input.

1. **Translation**

Translation is the shifting of object's location. If you know the shift in (x,y) direction, let it be (t_x, t_y), you can create the transformation matrix M as follows:

$$M = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \end{bmatrix}$$

You can take make it into a Numpy array of type `np.float32` and pass it into cv2.warpAffine()function. See below example for a shift of (100,50):

2. **Rotation**

Rotation of an image for an angle θ is achieved by the transformation matrix of the form

$$M = \begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix}$$

But OpenCV provides scaled rotation with adjustable center of rotation so that you can rotate at any location you prefer. Modified transformation matrix is given by

$$\begin{bmatrix} \alpha & \beta & (1-\alpha)\cdot center.x - \beta \cdot center.y \\ -\beta & \alpha & \beta \cdot center.x + (1-\alpha)\cdot center.y \end{bmatrix}$$

where:

$$\alpha = scale \cdot \cos\theta,$$
$$\beta = scale \cdot \sin\theta$$

To find this transformation matrix, OpenCV provides a function**,** cv2.getRotationMatrix2D. Check below example which rotates the image by 90 degree with respect to center without any scaling

3. **Affine Transformation**

In affine transformation, all parallel lines in the original image will still be parallel in the output image. To find the transformation matrix, we need three points from input image and their corresponding locations in output image. Then cv2.getAffineTransform will create a 2x3 matrix which is to be passed to cv2.warpAffine**.**

Check below example, and also look at the points I selected (which are marked in Green color):

4. **Perspective Transformation**

For perspective transformation, you need a 3x3 transformation matrix. Straight lines will remain straight even after the transformation. To find this transformation matrix, you need 4 points on the input image and corresponding points on the output image. Among these 4 points, 3 of them should not be collinear. Then transformation matrix can be found by the function cv2.getPerspectiveTransform. Then apply cv2.warpPerspective with this 3x3 transformation matrix.

Morphological Transformations

Morphological transformations are some simple operations based on the image shape. It is normally performed on binary images. It needs two inputs, one is our original image, second one is called structuring element or kernel which decides the nature of operation. Two basic morphological operators are Erosion and Dilation. Then its variant forms like Opening, Closing, Gradient etc also comes into play.

1. Erosion

The basic idea of erosion is just like soil erosion only, it erodes away the boundaries of foreground object (Always try to keep foreground in white). So what does it do? The kernel slides through the image (as in 2D convolution). A pixel in the original image (either 1 or 0) will be considered 1 only if all the pixels under the kernel is 1, otherwise it is eroded (made to zero).

So what happends is that, all the pixels near boundary will be discarded depending upon the size of kernel. So the thickness or size of the foreground object decreases or simply white region decreases in the image. It is useful for removing small white noises (as we have seen in colorspace chapter), detach two connected objects etc.

2. Dilation

It is just opposite of erosion. Here, a pixel element is ‘1’ if atleast one pixel under the kernel is ‘1’. So it increases the white region in the image or size of foreground object increases. Normally, in cases like noise removal, erosion is followed by dilation. Because, erosion removes white noises, but it also shrinks our object. So we dilate it. Since noise is gone, they won’t come back, but our object area increases. It is also useful in joining broken parts of an object.

3. Opening

Opening is just another name of erosion followed by dilation. It is useful in removing noise, as we explained above. Here we use the function, cv2.morphologyEx()

4. Closing

Closing is reverse of Opening, Dilation followed by Erosion. It is useful in closing small holes inside the foreground objects, or small black points on the object.

5. Morphological Gradient

It is the difference between dilation and erosion of an image.

The result will look like the outline of the object.

6. Top Hat
It is the difference between input image and Opening of the image. Below example is done for a 9x9 kernel.

7. Black Hat
It is the difference between the closing of the input image and input image.

Image Gradients

OpenCV provides three types of gradient filters or High-pass filters, Sobel, Scharr and Laplacian. We will see each one of them.

1. Sobel and Scharr Derivatives

Sobel operators is a joint Gausssian smoothing plus differentiation operation, so it is more resistant to noise. You can specify the direction of derivatives to be taken, vertical or horizontal (by the arguments, `yorder` and `xorder` respectively). You can also specify the size of kernel by the argument `ksize`. If ksize = -1, a 3x3 Scharr filter is used which gives better results than 3x3 Sobel filter. Please see the docs for kernels used.

2. Laplacian Derivatives

It calculates the Laplacian of the image given by the relation, $\Delta src = \frac{\partial^2 src}{\partial x^2} + \frac{\partial^2 src}{\partial y^2}$ where each derivative is found using Sobel derivatives. If `ksize = 1`, then following kernel is used for filtering:

$$kernel = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

LITERATURE SURVEY:

Many of the techniques of digital image processing, or digital picture processing as it often was called, were developed in the 1960s at the Jet Propulsion Laboratory, Massachusetts Institute of Technology, Bell Laboratories, University of Maryland. A few researches such as application to satellite images, wire-photo standards conversion, medical imaging, videophone, character recognition, and photograph enhancement were also carried out.

SuezouNakadateet al discussed the use of digital image processing techniques for electronic speckle pattern interferometry. A digital TV-image processing system with a large frame memory allows them to perform precise and flexible operations such as subtraction, summation, and level slicing. Digital image processing techniques made it easy compared with analog techniques to generate high contrast fringes.

Satoshi Kawataet al discussed the characteristics of the iterative image-restoratio 61 William H highlighted the progress in the image processing and analysis of digital images during the past ten years. The topics included digitization and coding, filtering, enhancement, and restoration, reconstruction from projections, hardware and software, feature detection, matching, segmentation, texture and shape analysis, and pattern recognition and scene analysis.

David W. Robinson presented the application of a general-purpose image-processing computer system to automatic fringe analysis. Three areas of application were examined where the use of a system based on a random access frame store has enabled a processing algorithm to be developed to suit a specific problem. Furthermore, it enabled automatic analysis to be performed with complex and noisy data. The applications considered were strain measurement by speckle interferometry, position location in three axes, and fault detection in holographic nondestructive testing. A brief description of each problem is presented, followed by a description of the processing algorithm, results, and timings.

S V Ahmed discussed the work prepared by concentrating upon the simulation and image processing aspects in the transmission of data over the subscriber lines for the development of an image processing system for eye statistics from eye .

P K Sahooet al presented a survey of thresholding techniques and updated the earlier survey work. An attempt was made to evaluate the performance of some automatic global thresholding methods using the criterion functions such as uniformity and shape measures. The evaluation was based on some real world images.

Marc Antoniniet al proposed a new scheme for image compression taking psycho- visual features in to account both in the space and frequency domains. This new method involved two steps. First, a wavelet transform in order to obtain a set of bi orthogonal subclasses of images; the original image is decomposed at different scales using a yramidal algorithm architecture. Second, according to Shannon's rate distortion theory, the wavelet coefficients are vector quantized using a multi resolution codebook. Furthermore, to encode the wavelet coefficients, a noise shaping bit allocation procedure was proposed which assumes that details at high resolution are less visible to the human eye. Finally, in order to allow the receiver to recognize a picture as quickly as possible at minimum cost, a progressive transmission scheme was presented. It is showed that the wavelet transform is particularly well adapted to progressive transmission.

62 Harpen MD presented a wavelet theory geared specifically for the radiological physicist. As a result, the radiological physicist can expect to be confronted with elementsof wavelet theory as diagnostic radiology advances into teleradiology, PACS, and computer aided feature extraction and diagnosis.

Salem Saleh Al-amriet al attempted to undertake the study of segmentation image techniques by using five threshold methods as Mean method, P-tile method, Histogram Dependent Technique (HDT), Edge Maximization Technique (EMT) and visual Technique and they are compared with one another so as to choose the best technique for threshold segmentation techniques image. These techniques are applied on three satellite images to choose base guesses for threshold segmentation image.

Wiecek B.et al proposed a new image processing tools for conversion thermal and visual images, mainly for application in medicine and biology. A novel method for area and distance evaluation based on statistical differencing was discussed. In order to increase the measurements accuracy, the interpolation and sub pixel bitmap processing were chosen.

Patnaiket al presented an image compression method using auto-associative neural network and embeddedzero-treecoding. The role of the neural network (NN) is to decompose the image stage by stage, which enabled analysis similar to wavelet decomposition. This works on the principle of principal component extraction (PCE). Network training is achieved through a recursive least squares (RLS) algorithm. The coefficients are arranged in a four-quadrant sub-band structure. The zero-treecoding algorithm is employed to quantize the coefficients. The system outperformed the embeddedzero-tree wavelet scheme in a rate-distortion sense, with best perceptual quality for a given compression ratio.

PAPER 1

Abstract: The author addresses the problem of automatic speech recognition in the presence of interfering noise. The novel approach described decomposes the contaminated speech signal using a generalization of standard hidden Markov modeling, while utilizing a compact and effective parametrization of the speech signal. The technique is compared to some existing noise compensation techniques, using data recorded in noise, and is found to have improved performance compared to existing model decomposition techniques. Performance is comparable to existing noise subtraction techniques, but the technique is applicable to a wider range of noise environments and is not dependent on an accurate end pointing of the speech.

There are many types of Noise Models, as Noise is undesirable information in digital images. We need to get rid of it, here are some noise models :

1. Gaussian Noise Model

2. White Noise

3. Brownian Noise (Fractal Noise)

4. Impulse Valued Noise (Salt and Pepper Noise)

5. Periodic Noise

6. Quantization Noise

7. Speckle Noise

8. Photon Noise (Poisson Noise)

9. Poisson-Gaussian Noise

10. Structured Noise

11. Gamma Noise

12. Rayleigh Noise

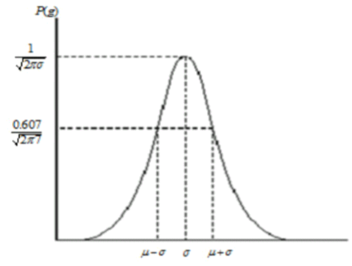
Let's discuss each Noise,

1. Gaussian Noise Model:It is also called as electronic noise because it arises in amplifiers or detectors. Gaussian noise caused by natural sources such as thermal vibration of atoms and discrete nature of radiation of warm objects.

Gaussian noise generally disturbs the gray values in digital images. That is why Gaussian noise model essentially designed and characteristics by its PDF or normalizes histogram with respect to gray value. This is given as

$$P(g) = \sqrt{\frac{1}{2\pi\sigma^2}}e^{-\frac{(g-\mu)}{2\sigma^2}}$$

Where g = gray value, σ = standard deviation and µ = mean. Generally Gaussian noise mathematical model represents the correct approximation of real-world scenarios. In this noise model, the mean value is zero, variance is 0.1 and 256 gray levels in terms of its PDF, which is shown



PDF of Gaussian Noise

Due to this equal randomness the normalized Gaussian noise curve look like in bell shaped. The PDF of this noise model shows that 70% to 90% noisy pixel values of degraded image in between µ σ µ σ − + and The shape of normalized histogram is almost same in spectral domain.

2. White Noise :Noise is essentially identified by the noise power. Noise power spectrum is constant in white noise. This noise power is equivalent to power spectral density function. The statement "Gaussian noise is often white noise" is incorrect [4]. However neither Gaussian property implies the white sense. The range of total noise power is -∞ to +∞ available in white noise in frequency domain. That means ideally noise power is infinite in white noise. This fact is fully true because the light emits from the sun has all the frequency components. In white noise, correlation is not possible because of every pixel values are different from their neighbours. That is why autocorrelation is zero. So that image pixel values are normally disturb positively due to white noise.

3. Brownian Noise (Fractal Noise): Colored noise has many names such as Brownian noise or pink noise or flicker noise or 1/f noise. In Brownian noise, power spectral density is proportional to square of frequency over an octave i.e., its power falls on ¼ th part (6 dB per octave). Brownian noise caused by Brownian motion. Brownian motion seen due to the random movement of suspended particles in fluid. Brownian noise can also be generated from white noise, which is shown in Fig. 2. However this noise follows non stationary stochastic process. This process follows normal distribution. Statistically fractional Brownian noise is referred to as fractal noise. Fractal noise is caused by natural process. It is different from Gaussian process

Although power spectrum of fractal noise, decaying continuously due to increase in frequency. Fractal noise is almost singular everywhere. A fractional Brownian motion is mathematically representing as a zero mean Gaussian process (BH) which is showing in equation (2) and equation (3), respectively

BH(0)=0..(2)

And expected value of fractional Brownian motion is

E{|BH(t)-BH(t-Δ)|^2}= σ^2|Δ|^2H..(3)

4. Impulse Valued Noise (Salt and Pepper Noise) :This is also called data drop noise because statistically its drop the original data values. This noise is also referred as salt and pepper noise. However the image is not fully corrupted by salt and pepper noise instead of some pixel values are changed in the image. Although in noisy image, there is a possibility of some neighbors does not changed [13-14]. This noise is seen in data transmission. Image pixel values are replaced by corrupted pixel values either maximum 'or' minimum pixel value i.e., 255 'or' 0 respectively, if number of bits are 8 for transmission. Let us consider 3x3 image matrices which are shown in the Fig. 3. Suppose the central value of matrices is corrupted by Pepper noise. Therefore, this central value i.e., 212 is given in Fig. 3 is replaced by value zero. In this connection, we can say that, this noise is inserted dead pixels either dark or bright. So in a salt and pepper noise, progressively dark pixel values are present in bright region and vice versa

254	207	210
97	212	32
62	106	20

$\longrightarrow$

254	207	210
97	0	32
62	106	20

The central pixel value is corrupted by Pepper Noise

Inserted dead pixel in the picture is due to errors in analog to digital conversion and errors in bit transmission. The percentagewise estimation of noisy pixels, directly determine from pixel metrics. PDF of noise is given below

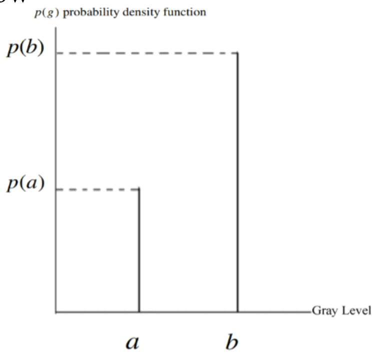


Figure 4. The PDF of Salt and Pepper noise

$$P(g) = \begin{cases} Pa \text{ for } g = a \\ Pb \text{ for } g = b \\ 0 \text{ otherwise} \end{cases} \quad (4$$

This shows the PDF of Salt and Pepper noise, if mean is zero and variance is 0.05. Here we will meet two spike one is for bright region (where gray level is less) called 'region a' and another one is dark region (where gray level is large) called 'region b', we have clearly seen here the PDF values are minimum and maximum in 'region a' and 'region b', respectively [16]. Salt and Pepper noise generally corrupted the digital image by malfunctioning of pixel elements in camera sensors, faluty memory space in storage, errors in digitization process and many more.

5. Periodic Noise: This noise is generated from electronics interferences, especially in power signal during image acquisition. This noise has special characteristics like spatially dependent and sinusoidal in nature at multiples of specific frequency. It's appears in form of conjugate spots in frequency domain. It can be conveniently removed by using a narrow band reject filter or notch filter.

6. Quantization noise : Quantization noise appearance is inherent in amplitude quantization process. It is generally presents due to analog data converted into digital data. In this noise model, the signal to noise ratio (SNR) is limited by minimum and maximum pixel value, Pmin and Pmax respectively. The SNR is given as

$$SNR_{dB} = 20\log_{10}(P_{\max} - P_{\min}) / \sigma_n \qquad (5)$$

Where σ_n = Standard deviation of noise, when input is full amplitude sine wave SNR becomes

$$SNR = 6n + 1.76 \text{ dB} \qquad (6)$$

Where n is number of bits. Quantization noise obeys the uniform distribution. That is why it is referred as uniform noise. Its PDF is shown in Fig. 5.

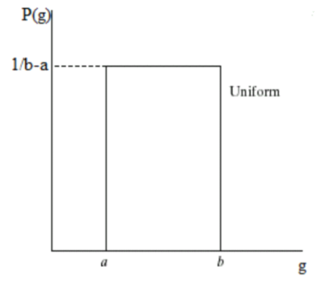


Figure 5 Uniform noise

$$P(g) = \begin{cases} \dfrac{1}{b-a} & \text{if } a \le g \le b \\ 0 & \text{otherwise} \end{cases} \qquad (7)$$

and their mean $\mu = \dfrac{a+b}{2}$ and variance $\sigma^2 = \dfrac{(b-a)^2}{12}$

7. Speckle Noise :This noise is multiplicative noise. Their appearance is seen in coherent imaging system such as laser, radar and acoustics etc,. Speckle noise can exist similar in an image as Gaussian noise. Its probability density function follows gamma distribution, which is shown in Fig. 6 and given as in equation (8) [17-19].

$$F(g) = \frac{g^{\alpha-1} e^{\frac{-g}{a}}}{\alpha - 1! a^{\alpha}} \qquad (8)$$

Speckle Noise with Variance 0.04

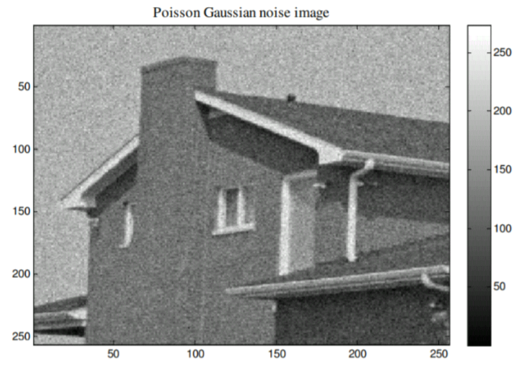
8. Photon Noise (Poisson Noise): The appearance of this noise is seen due to the statistical nature of electromagnetic waves such as x-rays, visible lights and gamma rays. The x-ray and gamma ray sources emitted number of photons per unit time. These rays are injected in patient's body from its source, in medical x rays and gamma rays imaging systems. These sources are having random fluctuation of photons. Result gathered image has spatial and temporal randomness. This noise is also called as quantum (photon) noise or shot noise. This noise obeys the Poisson distribution and is given as

$$P(f_{(pi)}) = k) = \frac{\lambda^{k}{}_{i}e^{-\lambda}}{k!} \tag{9}$$

9. Poisson-Gaussian Noise: In this section the proposed model work for removing of Poisson-Gaussian noise that is arose in Magnetic Resonance Imaging (MRI). Poisson-Gaussian noise is shown in Fig. 7. The paper introduces the two most fantastic noise models, jointly called as Poisson-Gaussian noise model. These two noise models are specified the quality of MRI recipient signal in terms of visual appearances and strength [21]. Despite from the highest quality MRI processing, above model describes the set of parameters of the Poisson-Gaussian noise corrupted test image. The Poisson Gaussian noise model can be illustrated in the following manner.

$$Z(j,k) = \alpha * P_{\alpha}(j,k) + N_{\alpha}(j,k) \tag{10}$$

Where, the model has carried Poisson-distribution (p_{α}) follows with Gaussian-distribution (n_{α}). Poisson-distribution estimating using mean $(\mu_{\alpha 1})$ at given level $\alpha > 0$ and Gaussian-distribution counted at given level $\alpha > 0$ using zero mean $(\mu_{\alpha 2})$ and variance $\sigma^{2}_{\alpha 2}$. To evaluate $\mu_{\alpha 1}$, noisy Poisson image was quantitatively added to underlying original image. After all to get a noisy image $Z(j,k)$ from the Poisson-Gaussian model is based on [22-23].



Poisson-Gaussian Noise House Image

10. Structured Noise: Structured noise are periodic, stationary or non-stationary and aperiodic in nature. If this noise is stationary, it has fixed amplitude, frequency and phase. Structured noise caused by interferences among electronic components [24]. Noise presents in communication channel are in two parts, unstructured noise (u) and structured noise (s). structured noise is also called low rank noise. In a signal processing, it is more advantagable (more realistic) to considering noise model in a lower dimensionality space. Further, this model is mapped into full rank measurement space in physical system. So we can conclude that in the measurement space, resulting noise has low rank and exhibits structure dependent on physical system. Structured noise model is showing in equation (11) and equation (12), respectively [25].

$$y_{(n)} = x_{(n,m)} + v_{(n)} \tag{11}$$

$$y_{(n)} = H_{(n,m)} * \theta_{(m)} + S_{(n,t)} * \phi_{(t)} + v_{(n)} \tag{12}$$

Where, $n =$ rows, $m =$ columns, $y =$ received image, $H =$ Transfer function of linear system, $S =$ Subspace, $t =$ rank in subspace, $\phi =$ underlying process exciting the linear system (S), $\theta =$ signal parameter sets initial conditions or excites, linear system H is used to produce original signal x in terms of n vector random noise $\left(v_{(n)}\right)$.

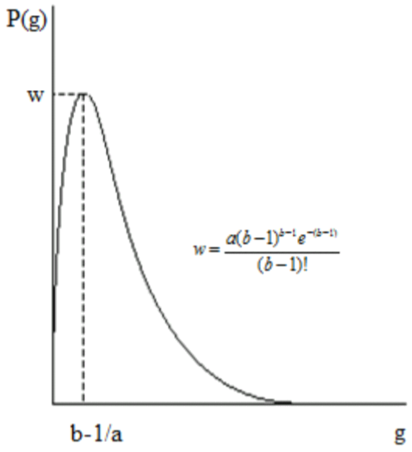
Structured Noise (when noise is periodic and non-stationary)

Structured Noise

11. Gamma Noise: Gamma noise is generally seen in the laser based images. It obeys the Gamma distribution. Which is shown below

$$P(g) = \begin{cases} \dfrac{a^b g^{b-1} e^{-ag}}{(b-1)!} & \text{for g} \geq 0 \\ 0 & \text{for g} < 0 \end{cases}$$

Where mean $\mu = \frac{b}{a}$ and variance $\sigma^2 = \frac{b}{a^2}$ are given as, respectively.

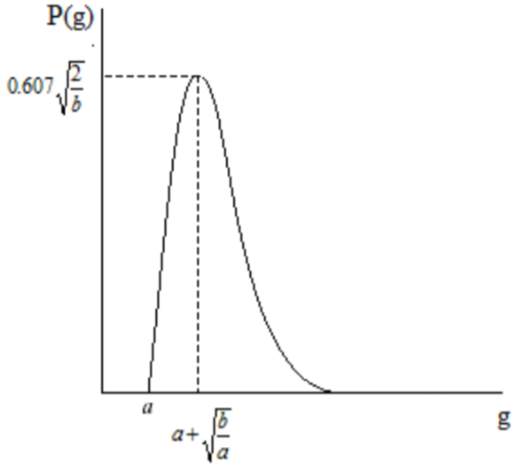


Gamma Distribution

12. Rayleigh noise Rayleigh noise presents in radar range images. In Rayleigh noise, probability density function is given as

$$P(g) = \begin{cases} \frac{2}{b}(g-a)e^{\frac{-(g-a)^2}{b}} & \text{for } g \geq a \\ 0 & \text{for } g < a \end{cases} \qquad (14)$$

Where mean $\mu = a + \sqrt{\frac{\pi b}{4}}$ and variance $\sigma^2 = \frac{b(4-\pi)}{4}$ are given as, respectively.



Rayleigh Distribution

CONCLUSION:

During image acquisition and transmission, noise is seen in images. This is characterised by noise model. So study of noise model is very important part in image processing. On the other hand, Image denoising is necessary action in image processing operation. Without the prior knowledge of noise model we cannot elaborate and perform denoising actions. Hence, here we have reviewed and presented various noise models available in digital images. We addressed that noise models can be identified with the help of their origin. Noise models also designed by probability density function using mean, variance and mainly gray levels in digital images. We hope this work will provide as a susceptible material for researchers and of course for freshers in the image processing field.

PAPER 2:

FILTERS AND THEIR CLASSIFICATION

Filter

A filter is a device that discriminates according to one or more attributes at its input, what passes through it.

One example is the color filter which absorbs light at certain wavelengths.

By filter design we can create filters that pass signals with frequency components in some bands, and attenuate signals with content in other frequency bands.

Required Classification as per Requirement

1. Low Pass Filter

A low-pass filter is a filter that passes low frequencies but attenuates higher than the cutoff frequency.

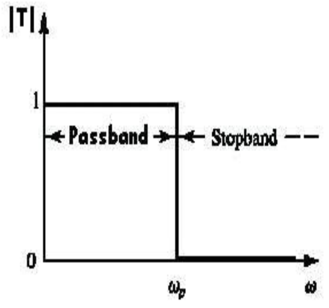


Figure 2

2. High Pass Filter

A high-pass filter is a filter that passes high frequencies well, but attenuates frequencies lower than the cut-off frequency.

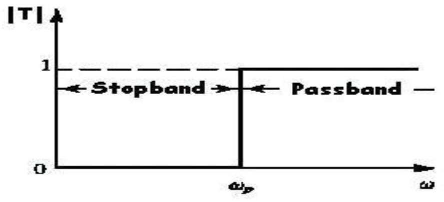


Figure 3

If we combine the above two together, we can design a filter that starts as a low-pass filter and slowly allows higher frequency components also and finally all frequencies can pass through that filter and we get the whole image.

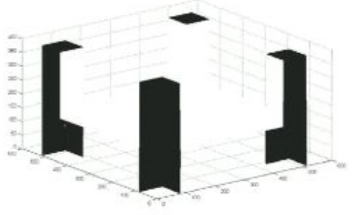


Figure 4

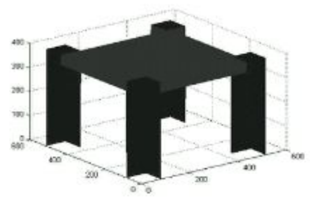
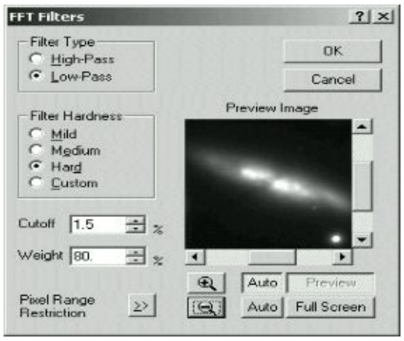


Figure 5

3. FFT Filter

FFT Filters provide precisely controlled low- and high-pass filtering (smoothing and sharpening, respectively) using a Butterworth characteristic. The image is converted into spatial frequencies using a Fast Fourier Transform, the appropriate filter is applied, and the image is converted back using an inverse FFT.



FFT Filters

Filter Type
High-Pass [✓ Low-Pass]

Filter Hardness
Mild Medium [✓ Hard] Custom

Cutoff 1.5 %
Weight 80. %

Pixel Range Restriction

<u>*Image Selection*</u>

Many grey-level images are available for the purpose of showing the function and working of a filter but for standard conventions I decided to choose Lenna.

Figure 6

<u>*Matrix Representation*</u>
The image chosen is now scaled to a fixed size of (512x512) and represented as a matrix. One important thing that has to be kept in mind is that image must have its both dimensions of at least 512 or else there will be a run-time error.

<u>*Area Division*</u>
The (512x512) matrix is divided into two major portions. First of all we separate a fixed square size area (say 60x60) from all corners of the image and we assume that in any of the functions applied this area will nor be taken into account. This area will simply act as pillars of the filter. The rest of the area left comprises of the second major portion of the image matrix.

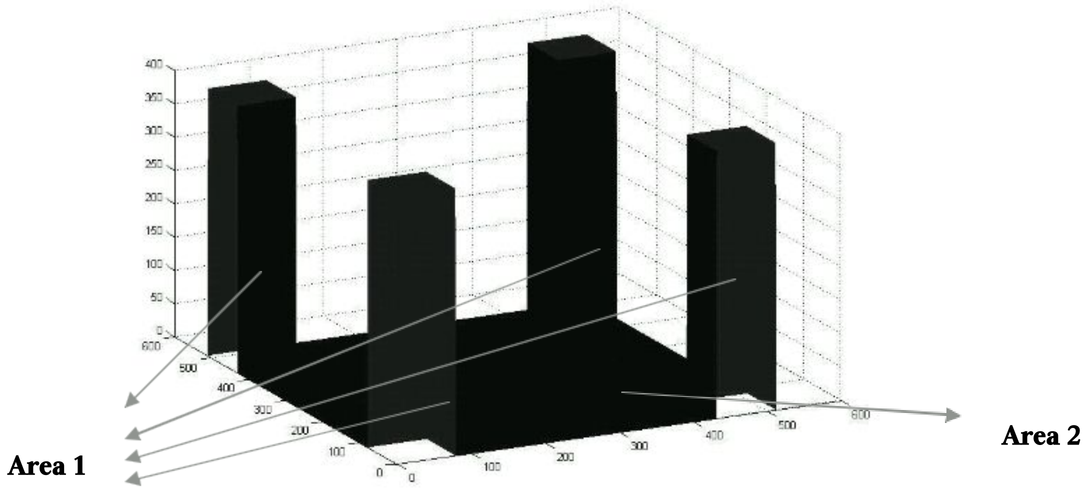


Figure 7

The Working

Phase - 1

Since MATLAB was very new to me, I started off by calculating the FFT of an image and observed some important things. When the FFT was calculated the entire image seemed to show noise all over. Since no boundary had been set for the frequencies, hence noise intrusion was very large and hence NOISE.

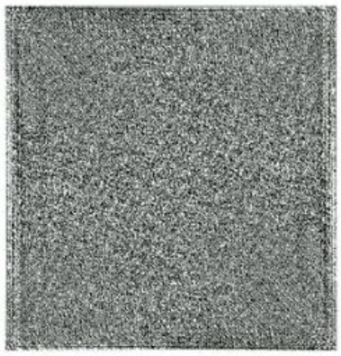


Figure 8

When 2-D FFT was calculated, proceedings made much more sense. Unless I processed a completely black image, a 2D Fourier transform of an image file (where all pixels have positive values) will always have a bright pixel in the center. That center pixel is called the DC term and represents the average brightness across the entire image.

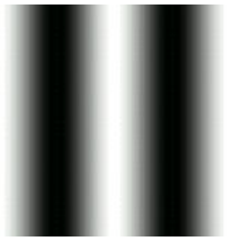


Figure 9

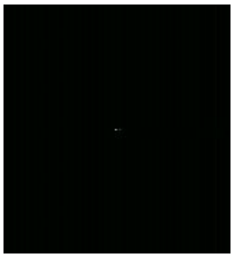


Figure 10

On the left side is an image of a sine wave where black pixels represent the bottom of the sine wave, white pixels the top and the gray pixels in between represent the sloping areas of the curve. On the right is the FFT of that image. **2D Fourier transforms are always symmetrical**. The upper left quadrant is identical to the lower right quadrant and the upper right quadrant is identical to the lower left quadrant. This is a natural consequence of how Fourier transforms work.

Phase-2

When the IFFT of the image was calculated, a white screen appeared showing nothing. The reason was; since the initial 2-D FFT was nothing but noise so nothing appeared, but when the mesh/surface plot of the image was viewed, I saw the following interesting result.

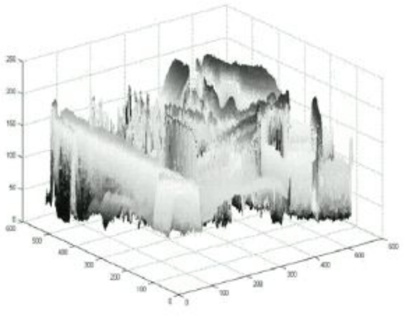


Figure11

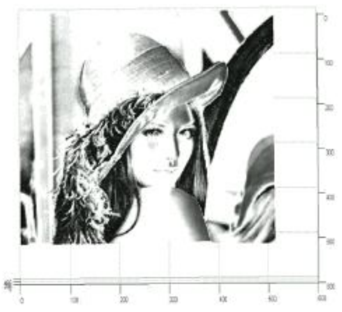


Figure 12

Figure 11 clearly depicts that when the MESH function was used, the colored parametric mesh defined by the matrix arguments was visible.

On rotating it in a 3-D manner, Lenna was clearly visible as in **Figure 12**.

The reason why Lenna could not be seen was that all the values after calculating the FFT were out of the specified range i.e. they were not NORMALIZED. The moment IFFT was calculated the values were back in range but there were some differences in the original values of the image matrix and the ones obtained after IFFT.

The values were not same because in the process of NORMALIZATION round-off functions are used which might change the values at some point of time.

A few important functions that had to be used were:

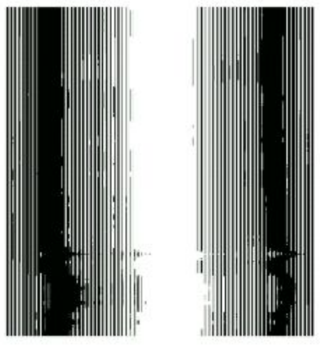
RGB2GRAY : Converts RGB image to grayscale by eliminating the hue and saturation information while retaining the luminance.

IM2DOUBLE : Convert image to double precision. IM2DOUBLE takes an image as input, and returns an image of class double. If the input image is of class double, the output image is identical to it. If the input image is not double, IM2DOUBLE returns the equivalent image of class double, rescaling or offsetting the data as necessary.

Once I understood the basic concepts, I moved ahead with the filter thing. After making the side pillars, now the sheet area that was in between those pillars had to be moved up slowly so that it acts as the frequency limit till where the frequencies could enter.

Phase – 3

To move the inner sheet up so that it acts as a frequency cut-off limiter, a code was written that limited only the inner area to move and not the entire block along with the sheet. Once this was achieved, I had to multiply the FFT of the image with this filter function that had been designed.
After multiplication the IFFT of the entire result was calculated and the filtered result was viewed for various values. The same result kept on repeating.



.

Figure 13

NORMALIZATION once again comes into picture. Since the final result had not been subjected to the specified range the result could not be viewed.

After all corrections results for various tests are displayed below:

Result-1

Filter window size 10 (Only very few low-range frequencies are allowed to pass) Image:

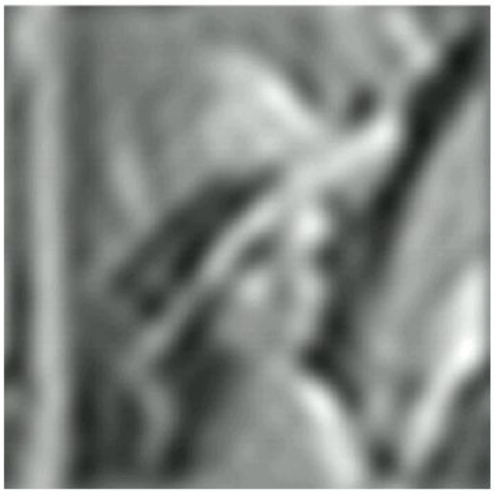


Figure 14

Result-2

Filter window size 50 (A few more frequencies are allowed to pass) Image:

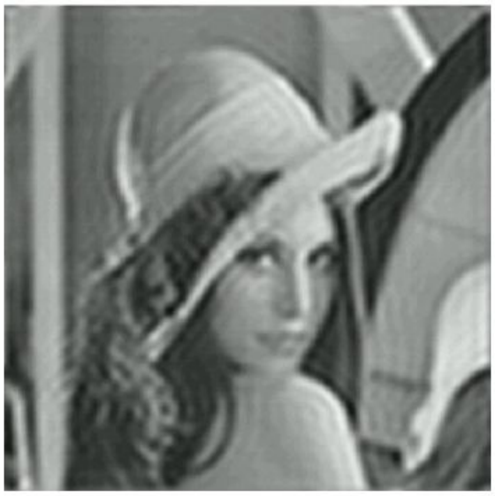


Figure 15

Result-3

Filter window size 180 (A lot more frequencies are allowed to pass) Image:

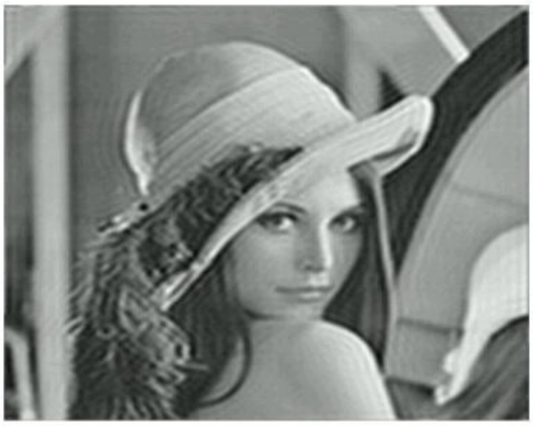


Figure 16

As you can see, by allowing the window size to increase and bringing the resultant frequencies within the range, the resulting filtered image becomes more and more clear.

Phase-4

Calculation of MSQE

Mean square quantization error (MSQE) is a figure of merit for the process of analog to digital conversion.
As the input is varied, the input's value is recorded when the digital output changes. For each digital output, the input's difference from ideal is normalized to the value of the least significant bit, then squared, summed, and normalized to the number of samples.

MSQE calculations were carried out for all the results abtained after filtering and were within permissible range i.e **-3.6% to +3.6%** of the original value.

Phase-5

Size of Outer Squares

The area that comprised of the four outer squares also plays a very important part in the final outlook of the image. Larger the square size, sharper is the final image.
The reason of such an outcome is that when the squares are chosen, the particular area is not being operated by any of the functions. Whenever a function acts on the matrix the image covered under the four squares is untouched and in the final output appears as it is. Hence there is a major change in the final image as we increase or decrease the size of the squares.

Observation : Larger is the size of the outer squares, better is the final image.

For an example, a few of the images have been shown below in increasing order of size of squares.

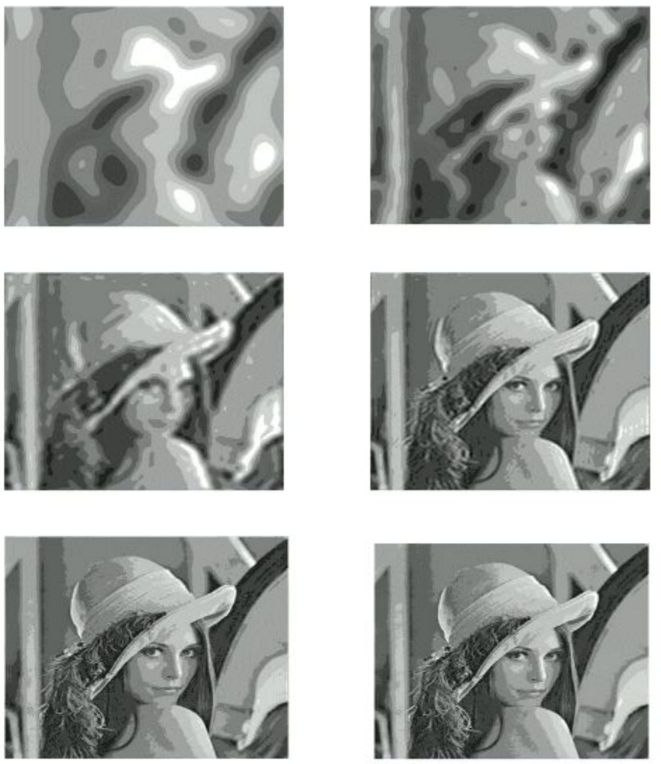


Figure-17

CODE:

```
print("\t\t\t\t\tWELCOME TO IMAGE PROCESSING PROJECT\n")
print("\nProject is made by:\n\nEashan Kaushik 17BCE2031\n\nTanay Nileshkumar Nagarsheth
17BCE2230\n\nSamarth Chauhan 12BCE0283\n\nPrakhar 17BCE2251\n")

q=int(input("\nEnter 1 to start\n"))

if q==1:

    print("\n"*50)


while(1):

    print("SELECT YOUR CHOICE\n")

    print("1.Image Thresholding\n")

    print("2.Image Smoothing\n")

    print("3.Geometric Transformations of Images\n")

    print("4.Morphological Transformation\n")

    print("5.Image Gradiet\n")

    print("6.Fourier Transform\n")


    x=int(input(""))

    print("\n"*50)


    if x==1:

        print("Select Choice:\n")

        print("1.Simple Thresholding\n")

        print("2.Adaptive Thresholding\n")

        print("3.Otsu's Binarization\n")

        y=int(input(""))


        if y==1:

            import cv2

            import numpy as np

            from matplotlib import pyplot as plt


            img = cv2.imread('grad.jpg',0)


            ret,thresh1 = cv2.threshold(img,127,255,cv2.THRESH_BINARY)

            ret,thresh2 = cv2.threshold(img,127,255,cv2.THRESH_BINARY_INV)

            ret,thresh3 = cv2.threshold(img,127,255,cv2.THRESH_TRUNC)

            ret,thresh4 = cv2.threshold(img,127,255,cv2.THRESH_TOZERO)

            ret,thresh5 = cv2.threshold(img,127,255,cv2.THRESH_TOZERO_INV)


            titles = ['Original Image','BINARY','BINARY_INV','TRUNC','TOZERO','TOZERO_INV']
            images = [img, thresh1, thresh2, thresh3, thresh4, thresh5]


            for i in xrange(6):
                plt.subplot(2,3,i+1),plt.imshow(images[i],'gray')
                plt.title(titles[i])
                plt.xticks([]),plt.yticks([])


            plt.show()
            print("\n"*50)


        if y==2:
            import cv2
            import numpy as np
            from matplotlib import pyplot as plt


            img = cv2.imread('crossword.jpg',0)
            img = cv2.medianBlur(img,5)


            ret,th1 = cv2.threshold(img,127,255,cv2.THRESH_BINARY)


            th2 = cv2.adaptiveThreshold(img,255,cv2.ADAPTIVE_THRESH_MEAN_C,\
                        cv2.THRESH_BINARY,11,2)


            th3 = cv2.adaptiveThreshold(img,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C,\
                        cv2.THRESH_BINARY,11,2)


            titles = ['Original Image', 'Global Thresholding (v = 127)'
                    ,'Adaptive Mean Thresholding', 'Adaptive Gaussian Thresholding']
            images = [img, th1, th2, th3]


            for i in xrange(4):
                plt.subplot(2,2,i+1),plt.imshow(images[i],'gray')
                plt.title(titles[i])
                plt.xticks([]),plt.yticks([])


            plt.show()
            print("\n"*50)


        if y==3:
                import cv2
                import numpy as np
                from matplotlib import pyplot as plt
                img = cv2.imread('noisy.jpg',0)

                # global thresholding
                ret1,th1 = cv2.threshold(img,127,255,cv2.THRESH_BINARY)

                # Otsu's thresholding
                ret2,th2 = cv2.threshold(img,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)

                # Otsu's thresholding after Gaussian filtering
                blur = cv2.GaussianBlur(img,(5,5),0)
                ret3,th3 = cv2.threshold(blur,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)

                images = [img, 0, th1, img, 0, th2,blur, 0, th3]
                titles = ['Original Noisy Image','Histogram','Global Thresholding (v=127)',
                          'Original Noisy Image','Histogram',"Otsu's Thresholding",
                          'Gaussian filtered Image','Histogram',"Otsu's Thresholding"]

                for i in xrange(3):
                    plt.subplot(3,3,i*3+1),plt.imshow(images[i*3],'gray')
                    plt.title(titles[i*3]), plt.xticks([]), plt.yticks([])
                    plt.subplot(3,3,i*3+2),plt.hist(images[i*3].ravel(),256)
                    plt.title(titles[i*3+1]), plt.xticks([]), plt.yticks([])
                    plt.subplot(3,3,i*3+3),plt.imshow(images[i*3+2],'gray')
                    plt.title(titles[i*3+2]), plt.xticks([]), plt.yticks([])

                plt.show()
                print("\n"*50)

            if x==2:
                print("Select Your Choice:\n")
                print("1.2-D Convulation(Image Filtering)\n")
                print("2.Averaging\n")
                print("3.Gaussian Filtering\n")
                print("4.Median Filtering\n")
                print("5.Bilateral Filtering\n")
```

```
y=int(input(""))

if(y==1):
    import cv2
    import numpy as np
    from matplotlib import pyplot as plt

    img = cv2.imread('OpenCV.jpg')

    kernel = np.ones((5,5),np.float32)/25
    dst = cv2.filter2D(img,-1,kernel)

    plt.subplot(121),plt.imshow(img),plt.title('Original')
    plt.xticks([]), plt.yticks([])
    plt.subplot(122),plt.imshow(dst),plt.title('Averaging')
    plt.xticks([]), plt.yticks([])

    plt.show()
    print("\n"*50)

if(y==2):
    import cv2
    import numpy as np
    from matplotlib import pyplot as plt

    img = cv2.imread('OpenCV.jpg')

    blur = cv2.blur(img,(5,5))

    plt.subplot(121),plt.imshow(img),plt.title('Original')
    plt.xticks([]), plt.yticks([])
    plt.subplot(122),plt.imshow(blur),plt.title('Blurred')
    plt.xticks([]), plt.yticks([])

    plt.show()
    print("\n"*50)

if(y==3):
    import cv2
    import numpy as np
    from matplotlib import pyplot as plt

    img = cv2.imread('OpenCV.jpg')

    blur = cv2.GaussianBlur(img,(5,5),0)

    plt.subplot(121),plt.imshow(img),plt.title('Original')
    plt.xticks([]), plt.yticks([])
    plt.subplot(122),plt.imshow(blur),plt.title('Blurred')
    plt.xticks([]), plt.yticks([])

    plt.show()
    print("\n"*50)

if(y==4):
    import cv2
    import numpy as np
    from matplotlib import pyplot as plt

    img = cv2.imread('OpenCV2.jpg')

    median = cv2.medianBlur(img,5)

    plt.subplot(121),plt.imshow(img),plt.title('Original')
    plt.xticks([]), plt.yticks([])
    plt.subplot(122),plt.imshow(median),plt.title('Blurred')
    plt.xticks([]), plt.yticks([])

    plt.show()
    print("\n"*50)

if(y==5):
    import cv2
    import numpy as np
    from matplotlib import pyplot as plt

    img = cv2.imread('ori.jpg')
```

```
        blur = cv2.GaussianBlur(img,(5,5),0)

        plt.subplot(121),plt.imshow(img),plt.title('Original')
        plt.xticks([]), plt.yticks([])
        plt.subplot(122),plt.imshow(blur),plt.title('Blurred')
        plt.xticks([]), plt.yticks([])

        plt.show()
        print("\n"*50)
if x==3:
    print('Select Your Choice:\n')
    print('1.Affine Transformation\n')
    print('2.Translation\n')
    print('3.Rotation\n')
    print('4.Perspective Transformation\n')

    y=int(input(""))

    if y==1:
        import cv2
        import numpy as np
        from matplotlib import pyplot as plt

        img = cv2.imread('affine.jpg')
        rows,cols,ch = img.shape

        pts1 = np.float32([[50,50],[200,50],[50,200]])
        pts2 = np.float32([[10,100],[200,50],[100,250]])

        M = cv2.getAffineTransform(pts1,pts2)

        dst = cv2.warpAffine(img,M,(cols,rows))

        plt.subplot(121),plt.imshow(img),plt.title('Input')
        plt.subplot(122),plt.imshow(dst),plt.title('Output')
        plt.show()
        print("\n"*50)
    if y==2:
        import cv2
        import numpy as np


        img = cv2.imread('messi.jpg',0)
        rows,cols = img.shape


        M = np.float32([[1,0,100],[0,1,50]])
        dst = cv2.warpAffine(img,M,(cols,rows))


        cv2.imshow('img',dst)
        cv2.waitKey(0)
        cv2.destroyAllWindows()
        print("\n"*50)
    if y==3:
        import cv2
        import numpy as np


        img = cv2.imread('messi.jpg',0)
        rows,cols = img.shape


        M = cv2.getRotationMatrix2D((cols/2,rows/2),90,1)
        dst = cv2.warpAffine(img,M,(cols,rows))


        cv2.imshow('img',dst)
        cv2.waitKey(0)
        cv2.destroyAllWindows()
        print("\n"*50)
    if y==4:
        import cv2
        import numpy as np
        from matplotlib import pyplot as plt


        img = cv2.imread('cross.jpg')
        rows,cols,ch = img.shape


        pts1 = np.float32([[56,65],[368,52],[28,387],[389,390]])
        pts2 = np.float32([[0,0],[300,0],[0,300],[300,300]])


        M = cv2.getPerspectiveTransform(pts1,pts2)
```

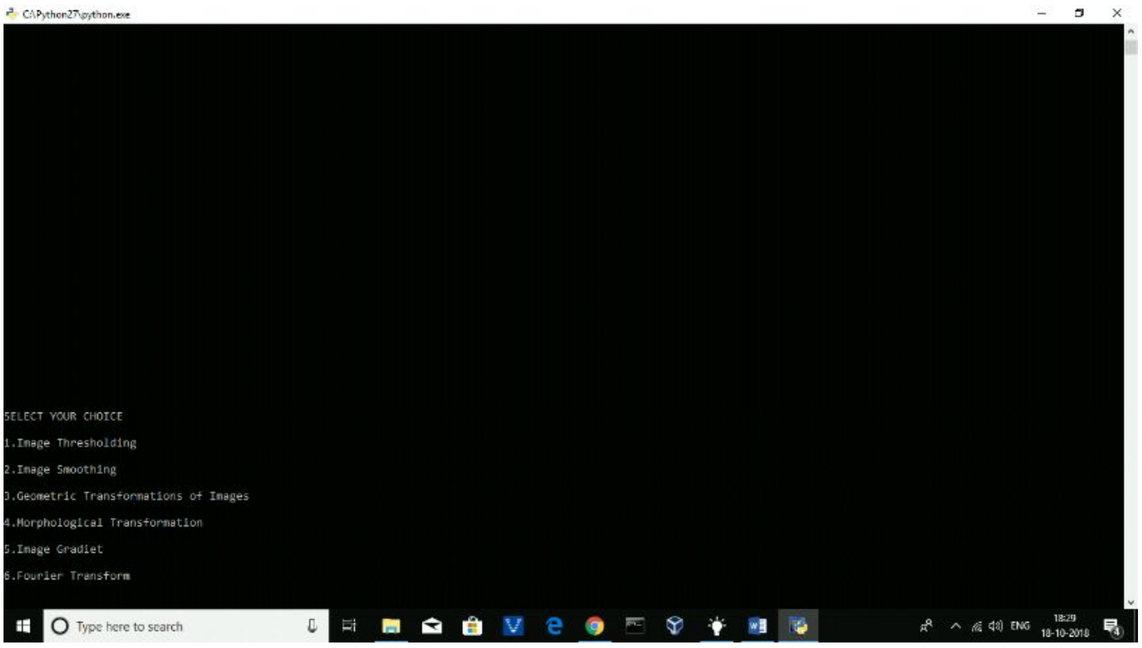
```
        dst = cv2.warpPerspective(img,M,(300,300))


        plt.subplot(121),plt.imshow(img),plt.title('Input')
        plt.subplot(122),plt.imshow(dst),plt.title('Output')
        plt.show()
        print("\n"*50)
if x==4:
    print('Select Your Choice:\n')
    print('1.Erosion\n')
    print('2.Dilation\n')
    print('3.Opening\n')
    print('4.Closing\n')
    print('5.Morphological Gradient\n')
    print('6.Top Hat\n')
    print('7.Black Hat\n')

    y=int(input(""))

    if y==1:
        import cv2
        import numpy as np
        from matplotlib import pyplot as plt

        img = cv2.imread('j.jpg',0)
        kernel = np.ones((5,5),np.uint8)
        erosion = cv2.erode(img,kernel,iterations = 1)
        plt.subplot(122),plt.imshow(erosion),plt.title('Output')
        plt.show()
        print("\n"*50)
    if y==2:
        import cv2
        import numpy as np
        from matplotlib import pyplot as plt

        img = cv2.imread('j.jpg',0)
        kernel = np.ones((5,5),np.uint8)
        dilation = cv2.dilate(img,kernel,iterations = 1)

        plt.subplot(122),plt.imshow(dilation),plt.title('Output')
    plt.show()
    print("\n"*50)
if y==3:
    import cv2
    import numpy as np
    from matplotlib import pyplot as plt

    img = cv2.imread('j2.jpg',0)
    kernel = np.ones((5,5),np.uint8)
    opening = cv2.morphologyEx(img, cv2.MORPH_OPEN, kernel)

    plt.subplot(122),plt.imshow(opening),plt.title('Output')

    plt.show()
    print("\n"*50)
if y==4:
    import cv2
    import numpy as np
    from matplotlib import pyplot as plt

    img = cv2.imread('j3.jpg',0)
    kernel = np.ones((5,5),np.uint8)
    closing = cv2.morphologyEx(img, cv2.MORPH_CLOSE, kernel)

    plt.subplot(122),plt.imshow(closing),plt.title('Output')

    plt.show()
    print("\n"*50)
if y==5:
    import cv2
    import numpy as np
    from matplotlib import pyplot as plt

    img = cv2.imread('j.jpg',0)
    kernel = np.ones((5,5),np.uint8)
    gradient = cv2.morphologyEx(img, cv2.MORPH_GRADIENT, kernel)

    plt.subplot(122),plt.imshow(gradient),plt.title('Output')
```

```
        plt.show()
        print("\n"*50)
    if y==6:
        import cv2
        import numpy as np
        from matplotlib import pyplot as plt

        img = cv2.imread('j.jpg',0)
        kernel = np.ones((5,5),np.uint8)
        tophat = cv2.morphologyEx(img, cv2.MORPH_TOPHAT, kernel)

        plt.subplot(122),plt.imshow(tophat),plt.title('Output')

        plt.show()
        print("\n"*50)
    if y==7:
        import cv2
        import numpy as np
        from matplotlib import pyplot as plt

        img = cv2.imread('j.jpg',0)
        kernel = np.ones((5,5),np.uint8)
        blackhat = cv2.morphologyEx(img, cv2.MORPH_BLACKHAT, kernel)

        plt.subplot(122),plt.imshow(blackhat),plt.title('Output')

        plt.show()
        print("\n"*50)
if x==5:
    print('Select Your Choice:\n')
    print('1.Laplacian Derivative\n')
    print('2.horizontal Sobel filter\n')

    y=int(input(""))

    if(y==1):
        import cv2
        import numpy as np
        from matplotlib import pyplot as plt

        img = cv2.imread('crossword.jpg',0)

        laplacian = cv2.Laplacian(img,cv2.CV_64F)
        sobelx = cv2.Sobel(img,cv2.CV_64F,1,0,ksize=5)
        sobely = cv2.Sobel(img,cv2.CV_64F,0,1,ksize=5)

        plt.subplot(2,2,1),plt.imshow(img,cmap = 'gray')
        plt.title('Original'), plt.xticks([]), plt.yticks([])
        plt.subplot(2,2,2),plt.imshow(laplacian,cmap = 'gray')
        plt.title('Laplacian'), plt.xticks([]), plt.yticks([])
        plt.subplot(2,2,3),plt.imshow(sobelx,cmap = 'gray')
        plt.title('Sobel X'), plt.xticks([]), plt.yticks([])
        plt.subplot(2,2,4),plt.imshow(sobely,cmap = 'gray')
        plt.title('Sobel Y'), plt.xticks([]), plt.yticks([])

        plt.show()
        print("\n"*50)

    if(y==2):
        import cv2
        import numpy as np
        from matplotlib import pyplot as plt

        img = cv2.imread('sobel.jpg',0)

        # Output dtype = cv2.CV_8U
        sobelx8u = cv2.Sobel(img,cv2.CV_8U,1,0,ksize=5)

        # Output dtype = cv2.CV_64F. Then take its absolute and convert to cv2.CV_8U
        sobelx64f = cv2.Sobel(img,cv2.CV_64F,1,0,ksize=5)
        abs_sobel64f = np.absolute(sobelx64f)
        sobel_8u = np.uint8(abs_sobel64f)

        plt.subplot(1,3,1),plt.imshow(img,cmap = 'gray')
        plt.title('Original'), plt.xticks([]), plt.yticks([])
        plt.subplot(1,3,2),plt.imshow(sobelx8u,cmap = 'gray')
        plt.title('Sobel CV_8U'), plt.xticks([]), plt.yticks([])
```

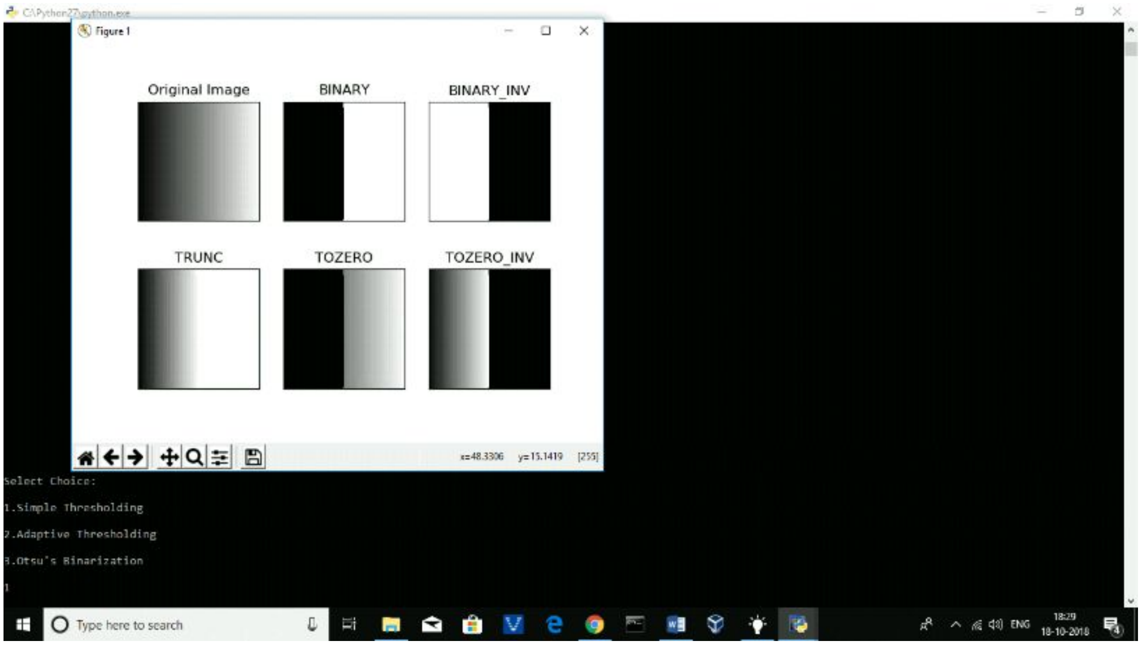
```
        plt.subplot(1,3,3),plt.imshow(sobel_8u,cmap = 'gray')
        plt.title('Sobel abs(CV_64F)'), plt.xticks([]), plt.yticks([])

        plt.show()
        print("\n"*50)
if x==6:
    print("Select Your Choice:\n")
    print("1.Discrete Fourier transform\n")
    print("2.Inverse Discrete Fourier transform\n")

    y=int(input(""))

    if y==1:
        import numpy as np
        import cv2
        from matplotlib import pyplot as plt

        img = cv2.imread('messi.jpg',0)

        dft = cv2.dft(np.float32(img),flags = cv2.DFT_COMPLEX_OUTPUT)
        dft_shift = np.fft.fftshift(dft)
        magnitude_spectrum = 20*np.log(cv2.magnitude(dft_shift[:,:,0],dft_shift[:,:,1])
        plt.subplot(121),plt.imshow(img, cmap = 'gray')
        plt.title('Input Image'), plt.xticks([]), plt.yticks([])
        plt.subplot(122),plt.imshow(magnitude_spectrum, cmap = 'gray')
        plt.title('Magnitude Spectrum'), plt.xticks([]), plt.yticks([])
        plt.show()
        print("\n"*50)
    if y==2:
        import numpy as np
        import cv2
        from matplotlib import pyplot as plt

        img = cv2.imread('messi.jpg',0)
        rows, cols = img.shape
        crow,ccol = rows/2 , cols/2

        # create a mask first, center square is 1, remaining all zeros
        mask = np.zeros((rows,cols,2),np.uint8)
```

```
mask[crow-30:crow+30, ccol-30:ccol+30] = 1
# apply mask and inverse DFT
fshift = dft_shift*mask
f_ishift = np.fft.ifftshift(fshift)
img_back = cv2.idft(f_ishift)
img_back = cv2.magnitude(img_back[:,:,0],img_back[:,:,1])
plt.subplot(121),plt.imshow(img, cmap = 'gray')
plt.title('Input Image'), plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(img_back, cmap = 'gray')
plt.title('Magnitude Spectrum'), plt.xticks([]), plt.yticks([])
plt.show()
print("\n"*50)
```

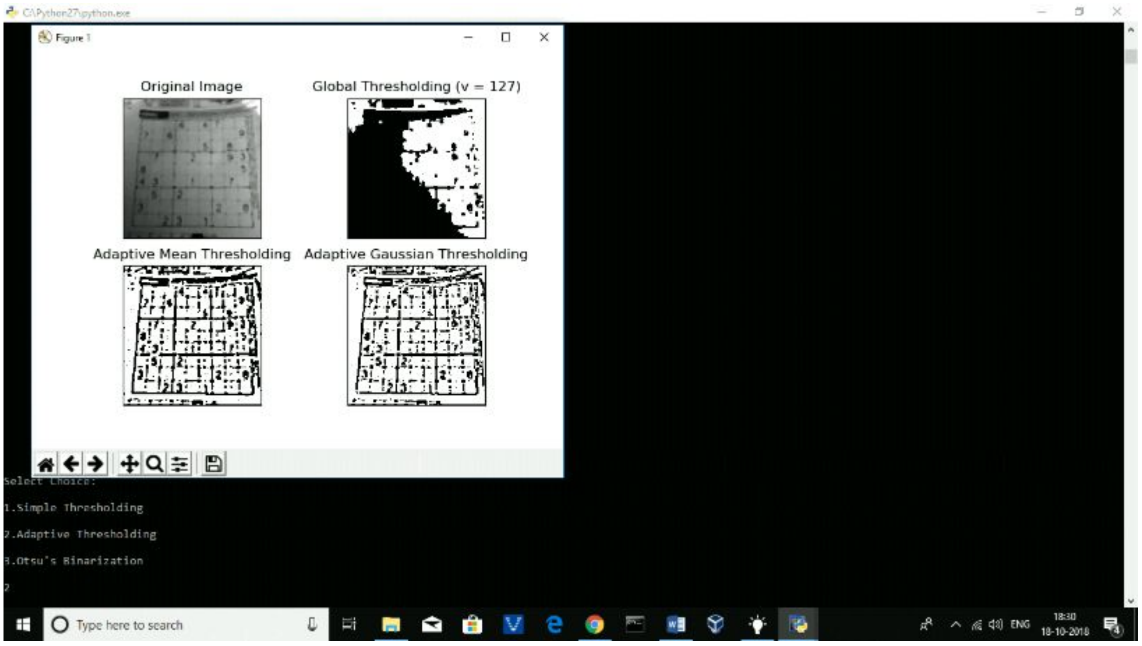
RESULT



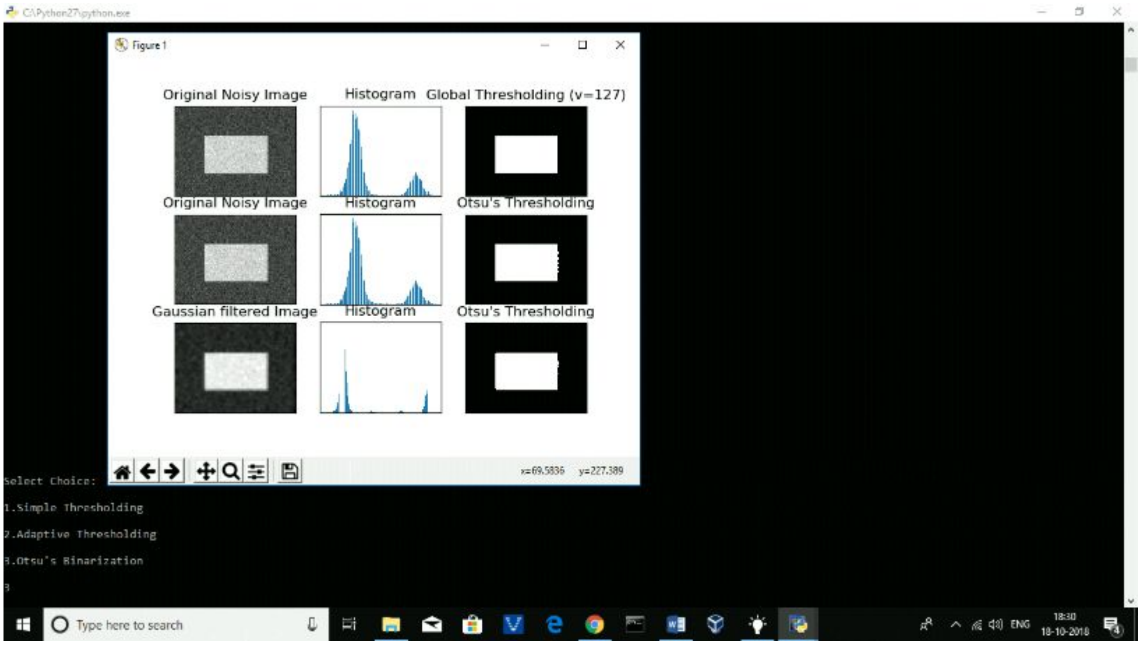
```
SELECT YOUR CHOICE
1.Image Thresholding
2.Image Smoothing
3.Geometric Transformations of Images
4.Morphological Transformation
5.Image Gradiet
6.Fourier Transform
```



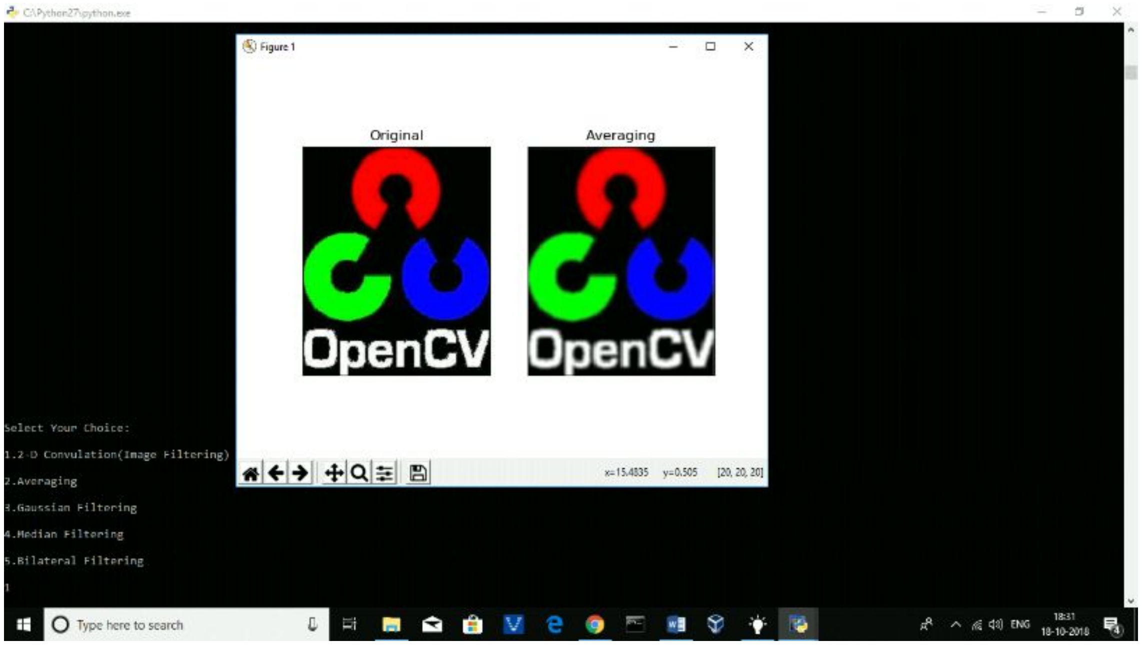
```
Select Choice:
1.Simple Thresholding
2.Adaptive Thresholding
3.Otsu's Binarization
1
```


C:\Python27\python.exe
Figure 1
Original Image
Global Thresholding (v = 127)
Adaptive Mean Thresholding
Adaptive Gaussian Thresholding
Type here to search
18:30
18-10-2018

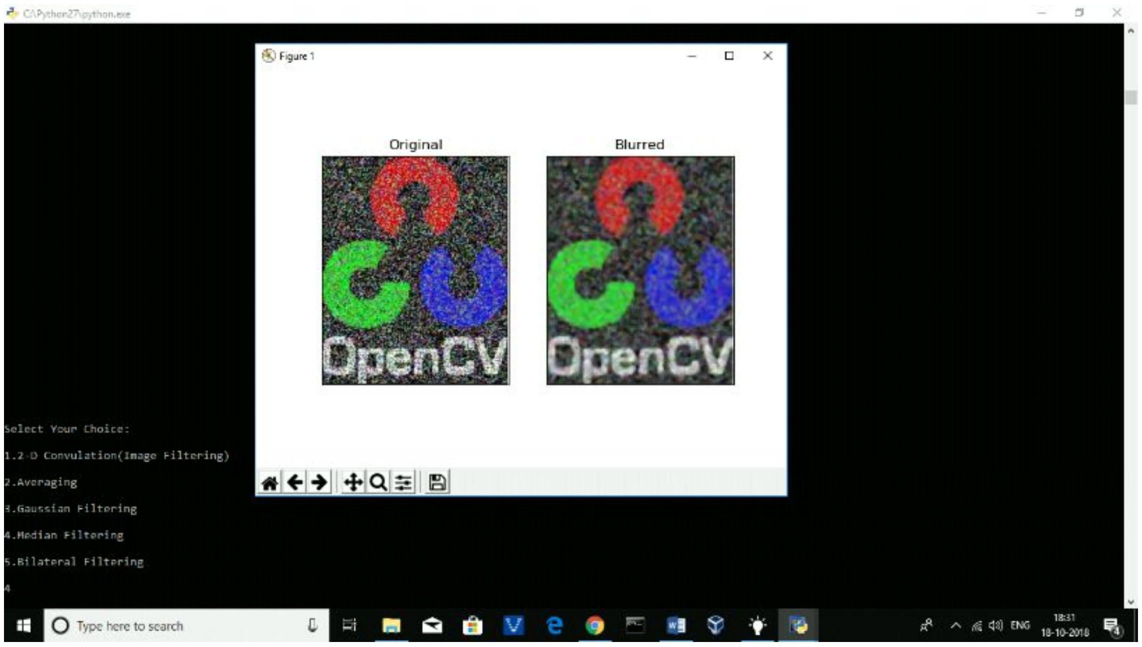
```
Select Choice:
1.Simple Thresholding
2.Adaptive Thresholding
3.Otsu's Binarization
2
```



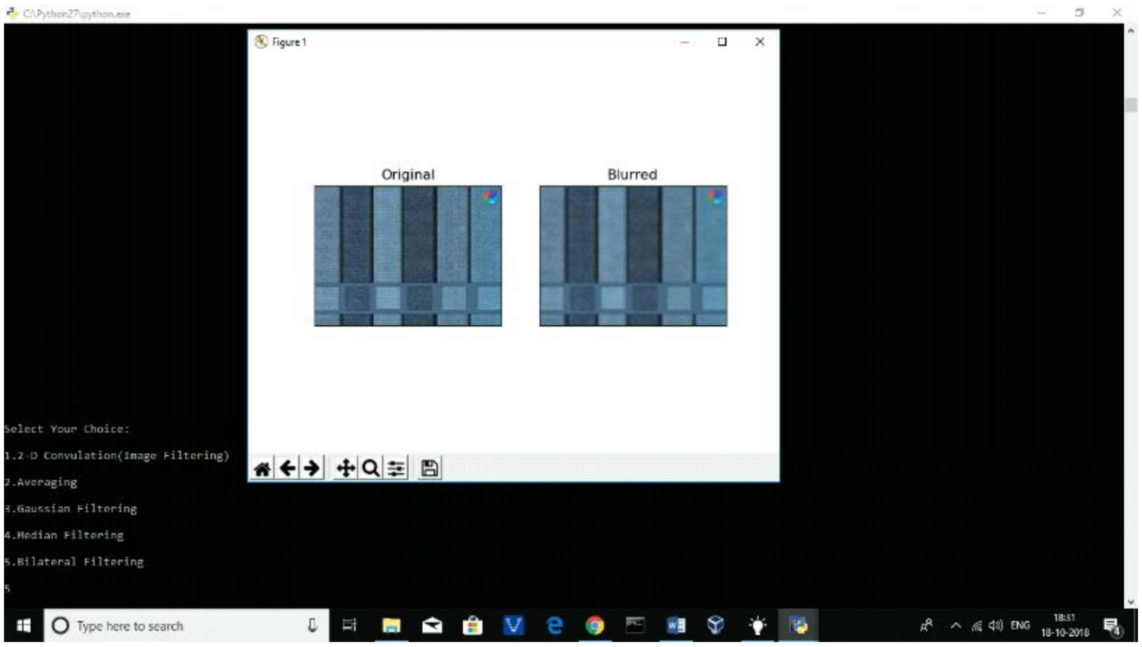
```
Select Choice:
1.Simple Thresholding
2.Adaptive Thresholding
3.Otsu's Binarization
3
```



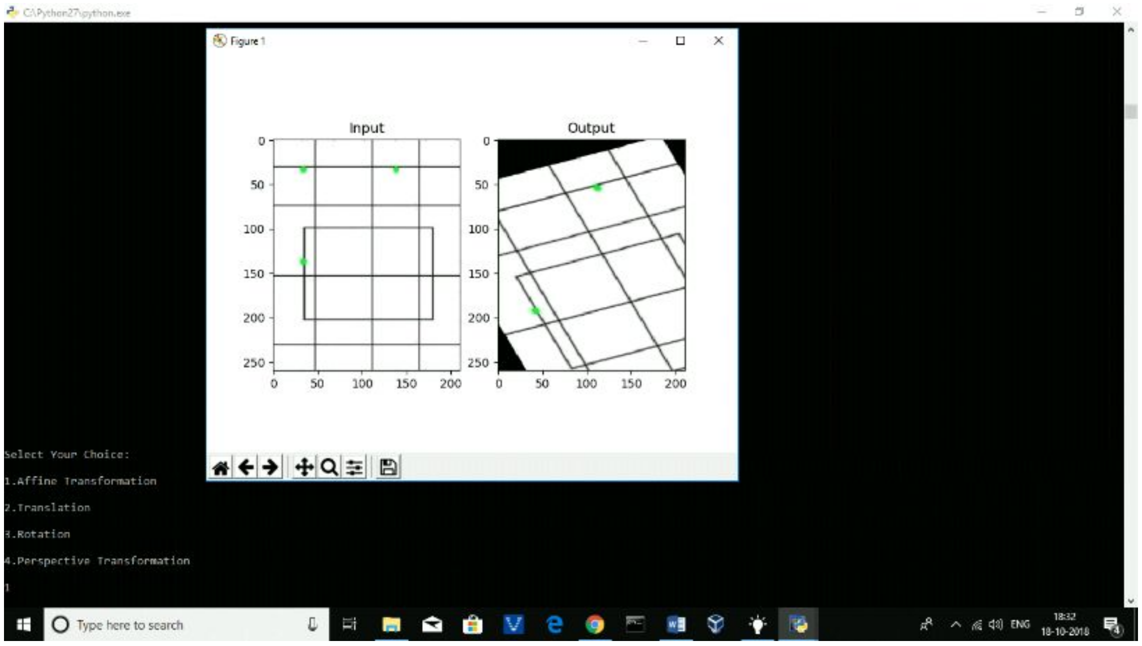
```
Select Your Choice:
1.2-D Convulation(Image Filtering)
2.Averaging
3.Gaussian Filtering
4.Median Filtering
5.Bilateral Filtering
1
```



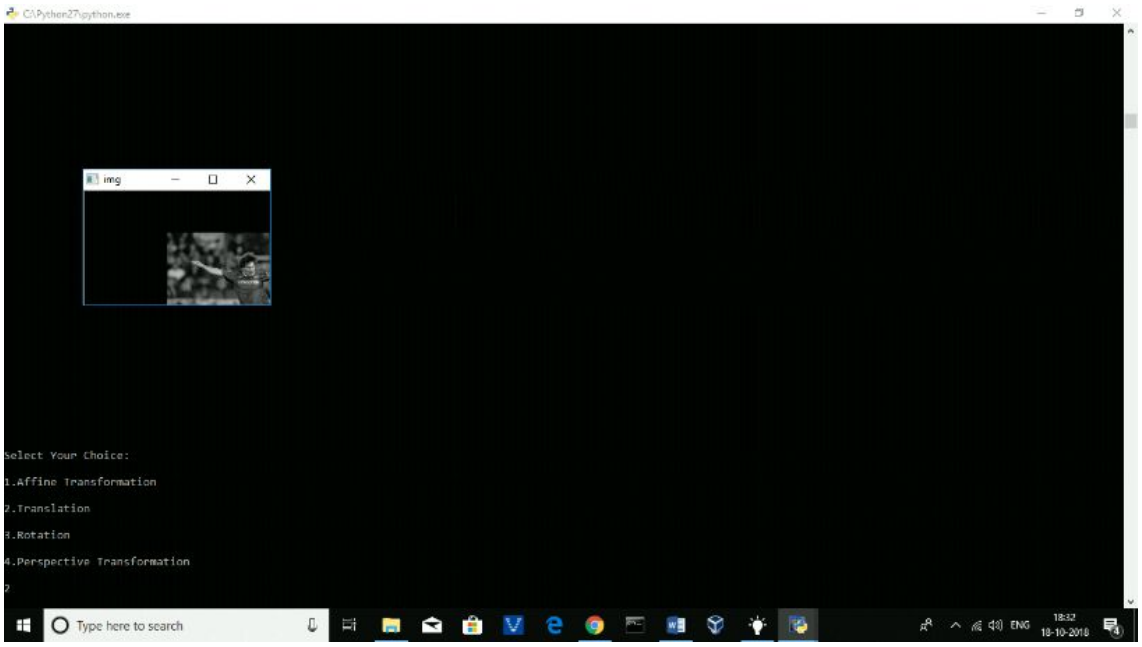
```
Select Your Choice:
1.2-D Convulation(Image Filtering)
2.Averaging
3.Gaussian Filtering
4.Median Filtering
5.Bilateral Filtering
4
```



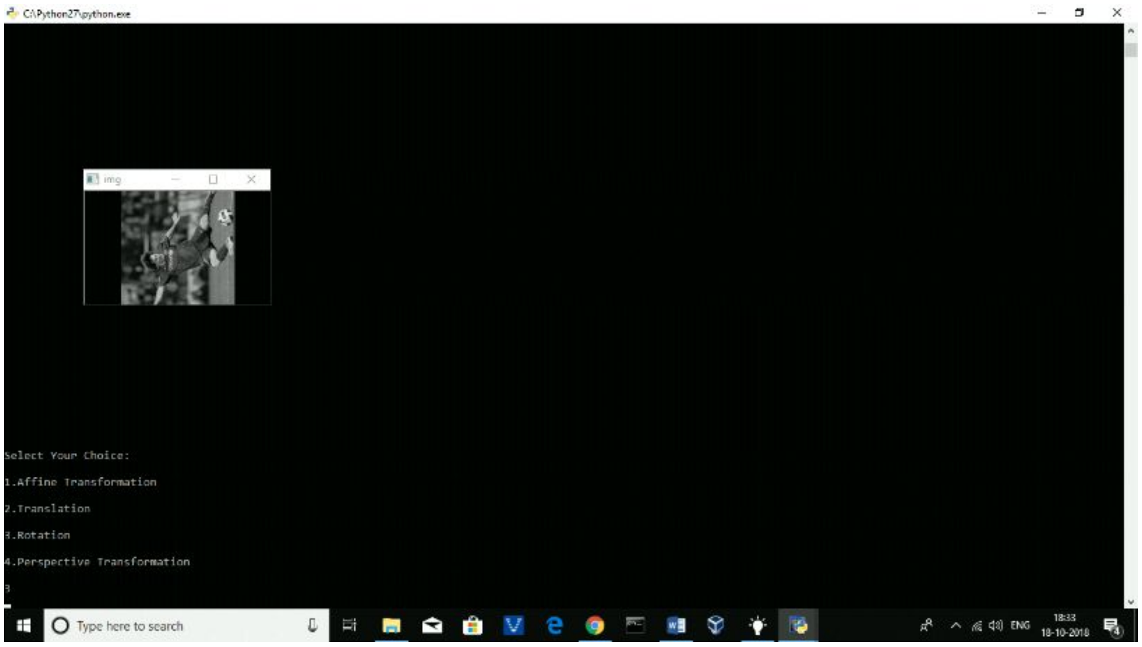
```
Select Your Choice:
1.2-D Convulation(Image Filtering)
2.Averaging
3.Gaussian Filtering
4.Median Filtering
5.Bilateral Filtering
5
```



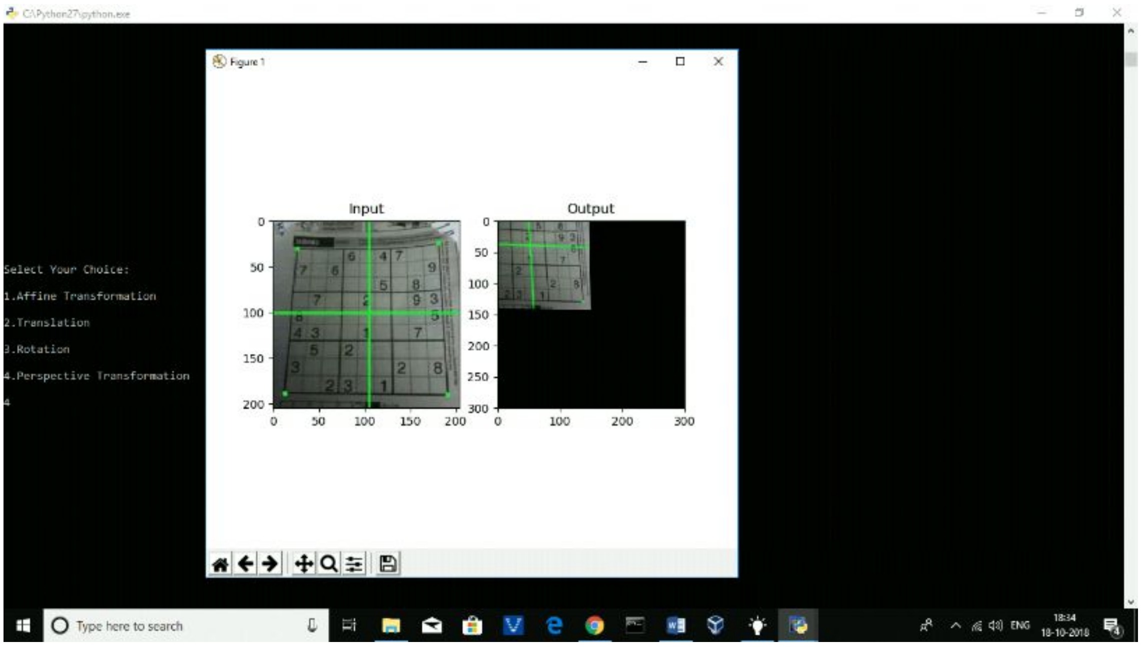
```
Select Your Choice:
1.Affine Transformation
2.Translation
3.Rotation
4.Perspective Transformation
1
```



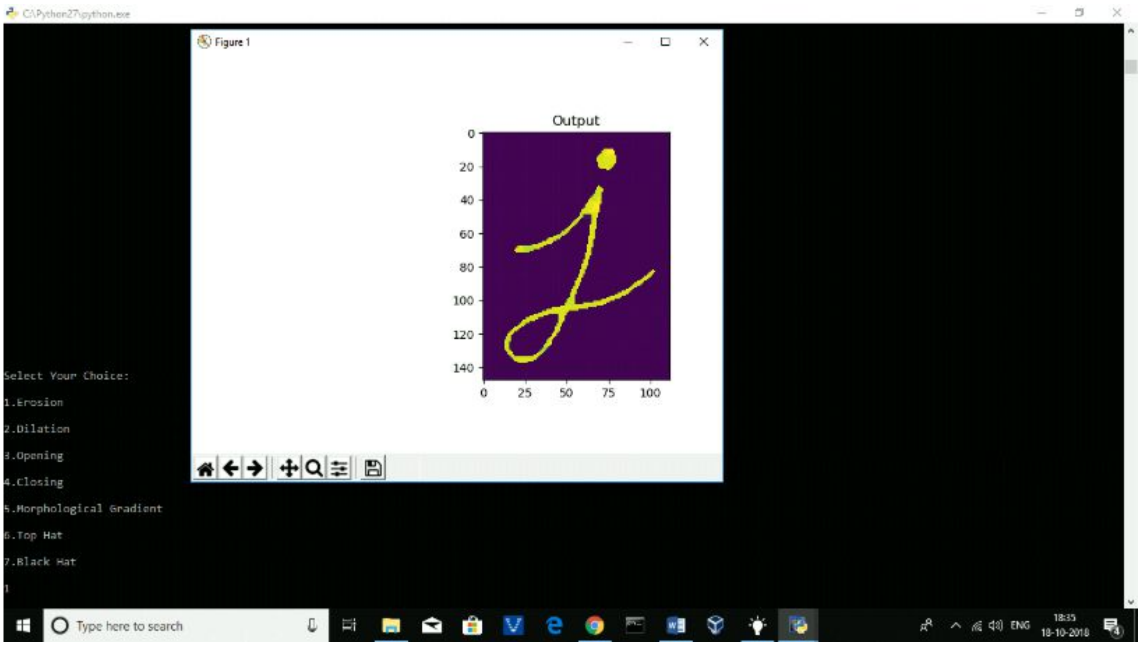
```
Select Your Choice:
1.Affine Transformation
2.Translation
3.Rotation
4.Perspective Transformation
2
```



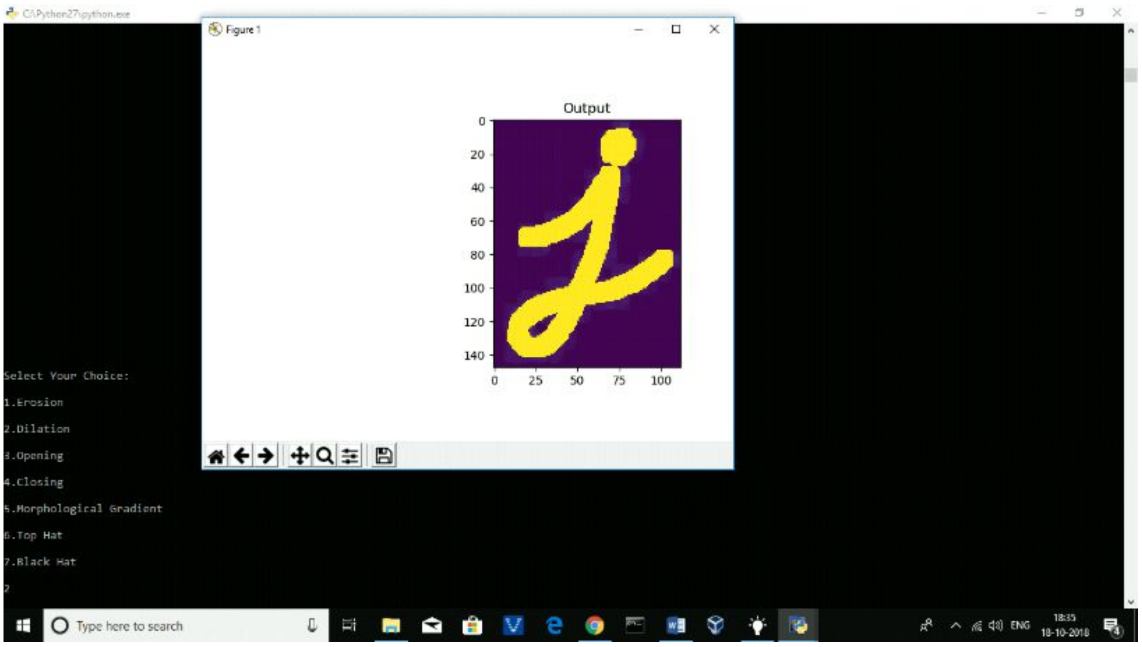
```
Select Your Choice:
1.Affine Transformation
2.Translation
3.Rotation
4.Perspective Transformation
3
```



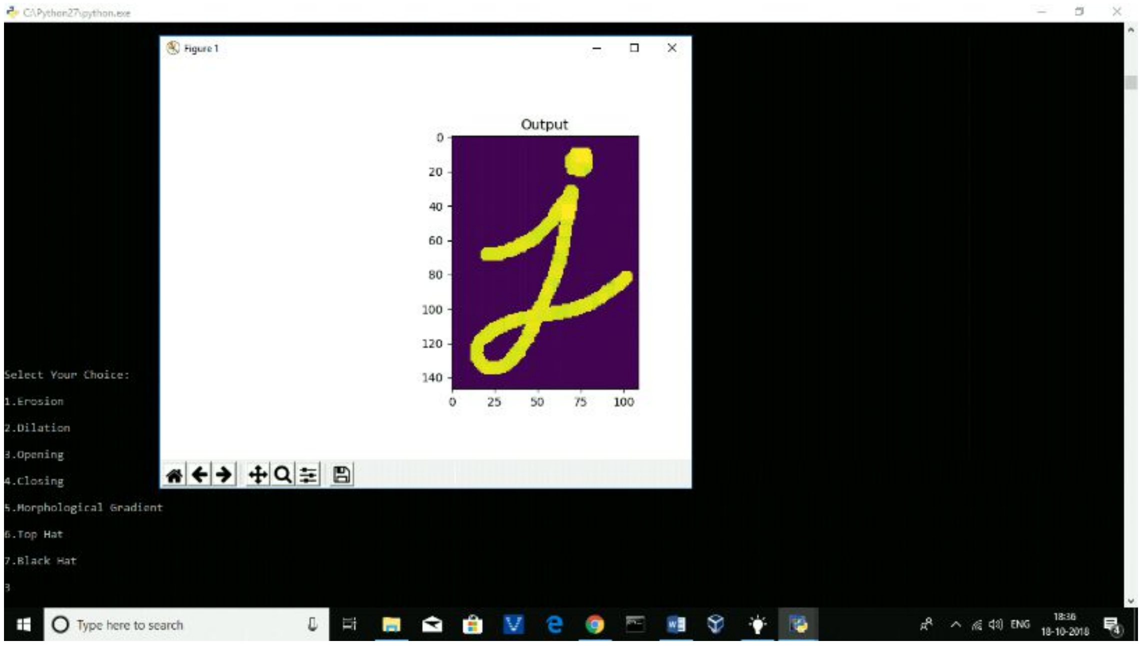
```
Select Your Choice:
1.Affine Transformation
2.Translation
3.Rotation
4.Perspective Transformation
4
```



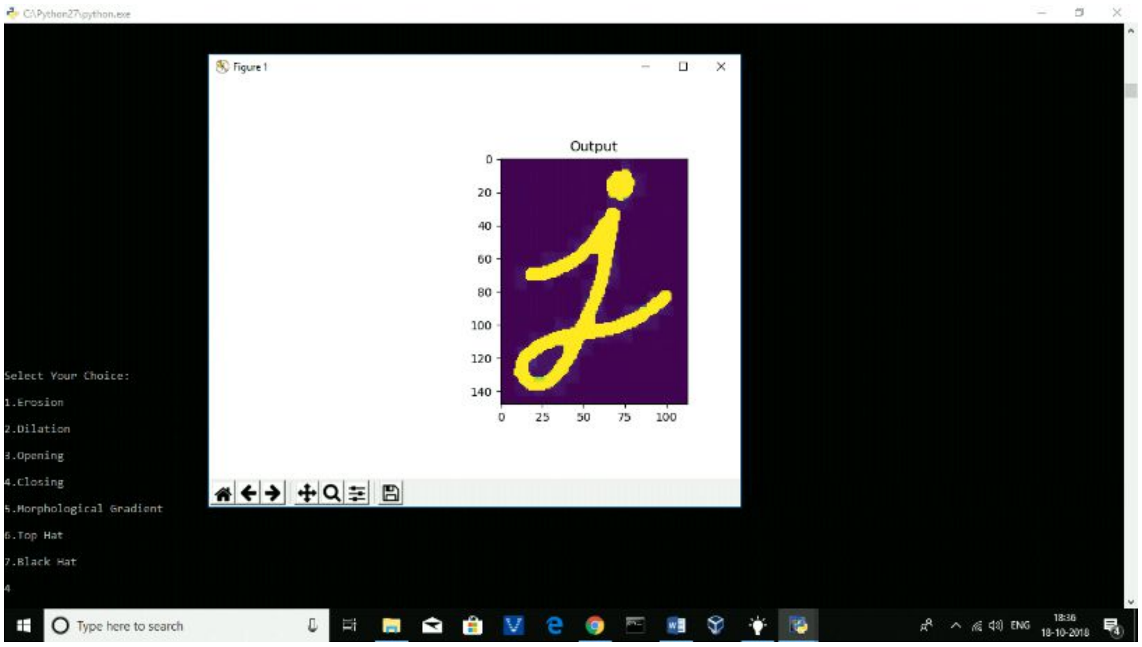
```
Select Your Choice:
1.Erosion
2.Dilation
3.Opening
4.Closing
5.Morphological Gradient
6.Top Hat
7.Black Hat
1
```



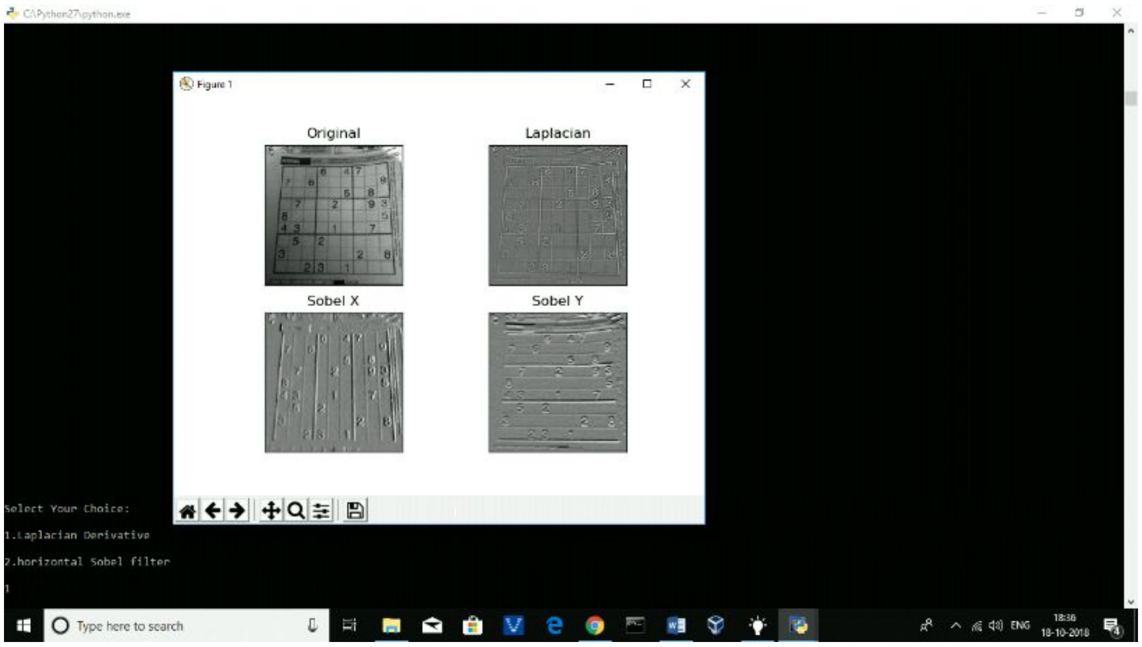
```
Select Your Choice:
1.Erosion
2.Dilation
3.Opening
4.Closing
5.Morphological Gradient
6.Top Hat
7.Black Hat
2
```



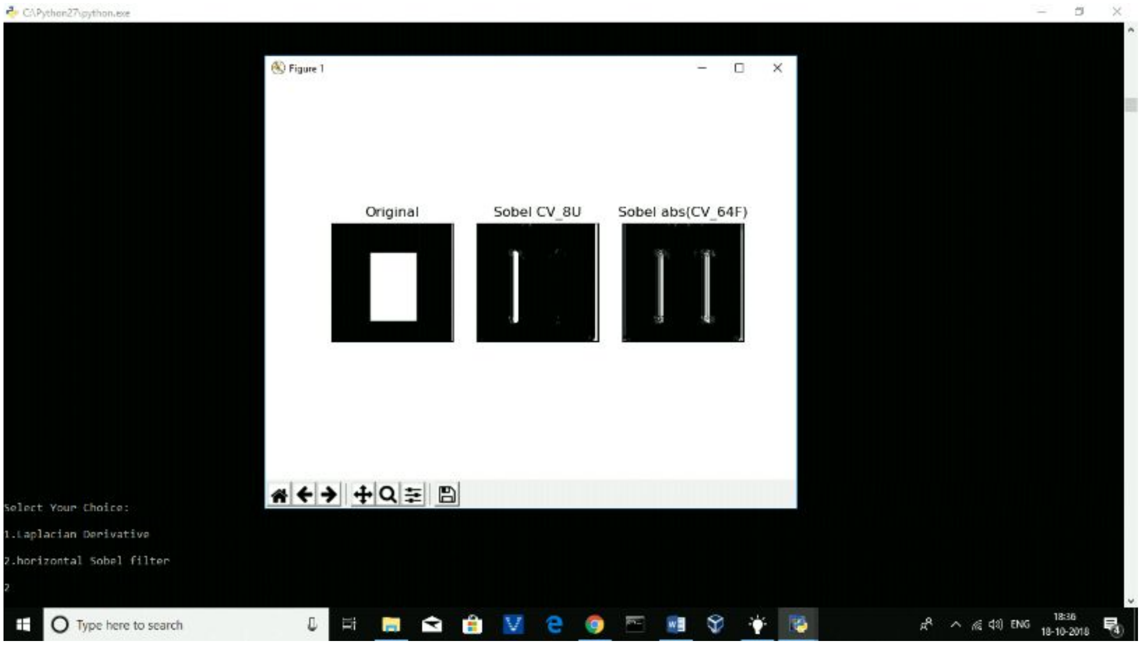
```
Select Your Choice:
1.Erosion
2.Dilation
3.Opening
4.Closing
5.Morphological Gradient
6.Top Hat
7.Black Hat
3
```



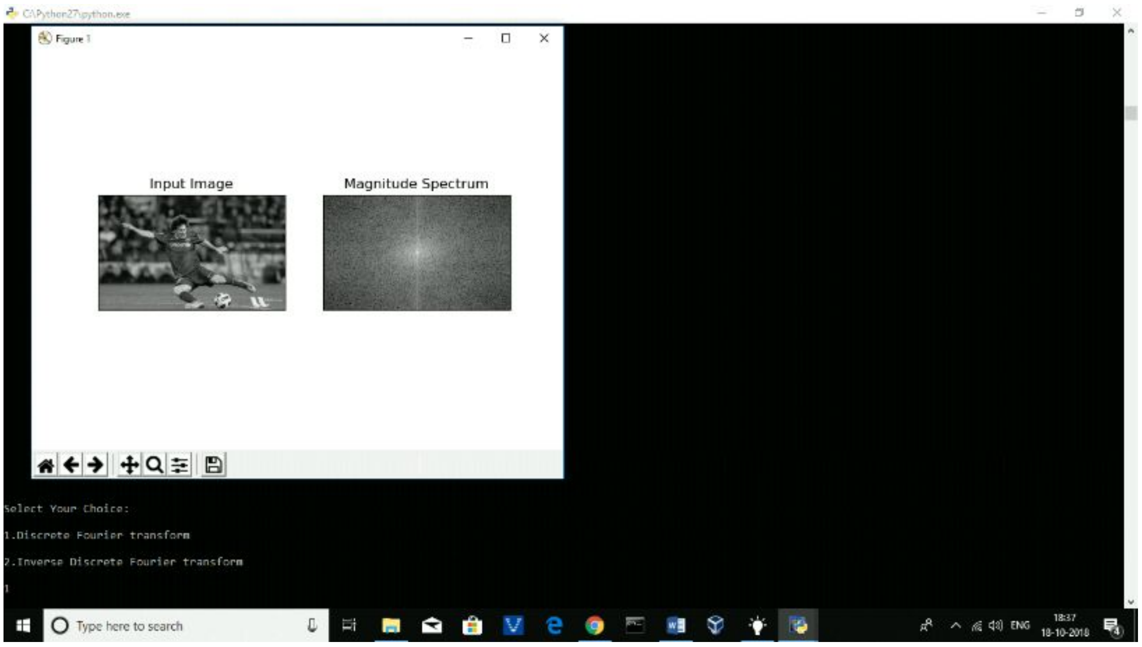
```
Select Your Choice:
1.Erosion
2.Dilation
3.Opening
4.Closing
5.Morphological Gradient
6.Top Hat
7.Black Hat
4
```



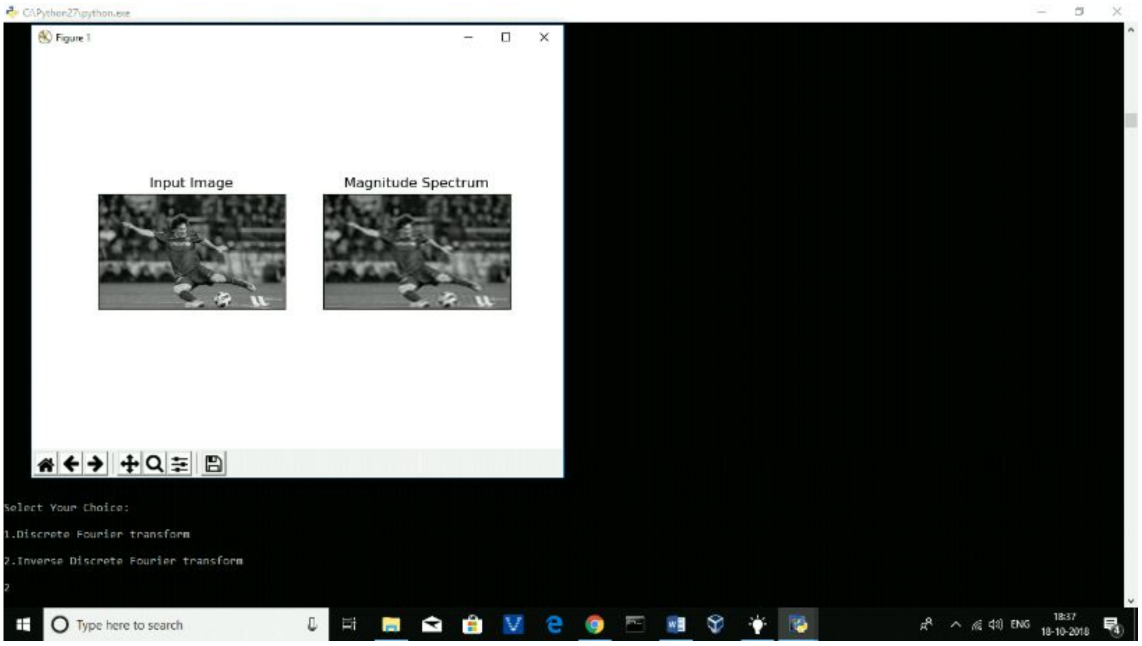
```
Select Your Choice:
1.Laplacian Derivative
2.horizontal Sobel filter
1
```



```
Select Your Choice:
1.Laplacian Derivative
2.horizontal Sobel filter
2
```



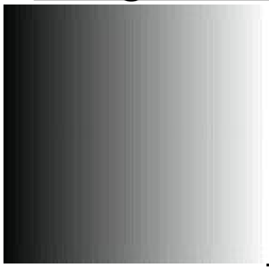
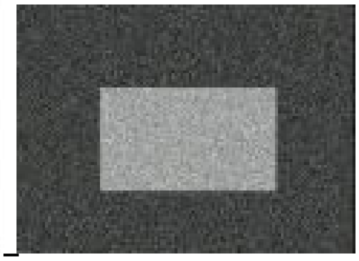
```
Select Your Choice:
1.Discrete Fourier transform
2.Inverse Discrete Fourier transform
1
```



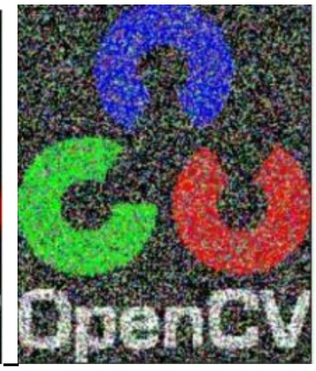
```
Select Your Choice:
1.Discrete Fourier transform
2.Inverse Discrete Fourier transform
2
```

TEST IMAGES:

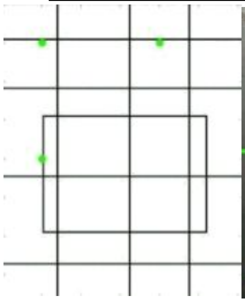
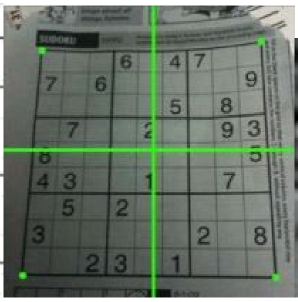
1. Image Thresholding

2. Image Smothening

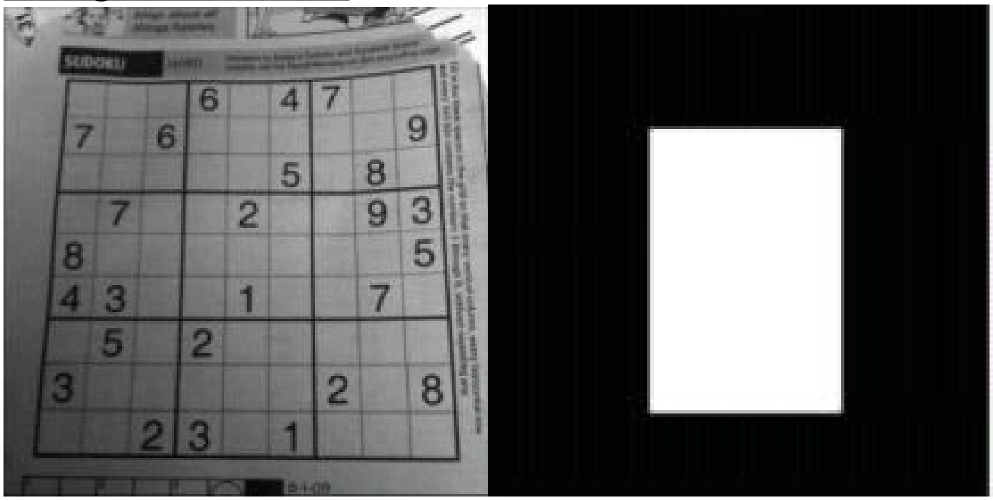


3. Transformation

4. Morphological

5. Image Gradient



6. Fourier transform

CONCLUSION

During image acquisition and transmission, noise is seen in images. This is characterised by noise model. So study of noise model is very important part in image processing. On the other hand, Image denoising is necessary action in image processing operation. Without the prior knowledge of noise model we cannot elaborate and perform denoising actions.

Hence, here we have reviewed and presented various noise models available in digital images. We addressed that noise models can be identified with the help of their origin. Noise models also designed by probability density function using mean, variance and mainly gray levels in digital images. We hope this work will provide as a susceptible material for researchers and of course for freshers in the image processing field

There are numerous other filters for the Image Compression process, just like the one I have designed. So what makes my project different from others ?

The basic concept underlying in my Filter Design is that using simple 2-D FFT and IFFT approach we can restrict the frequencies according to our choice. It depends entirely on the user to mould the filter specifications and allow a particular range of frequencies from the origin to pass through and observe the result obtained as an image.

Also the side squares approach helps us to find a solution where minor (sometimes imperceptible) loss of fidelity is acceptable to achieve a substantial reduction in bit rate and some of the finer details in the image can be sacrificed for the sake of saving a little more bandwidth or storage space.

Currently, design of filters with a very high precision and degree of control are not available. I hope that my effort is going to find applications in near future.

BIBLIOGRAPHY

1. Gonzalez and Woods, Digital Image Processing. Pearson Education Inc., 2002

2. P.Ramesh Babu, Digital Image Processing. Scitech Publications., 2003
3. Rudra Pratap, Getting Started With MATLAB 7. Oxford University Press, 2006
4. Bracewell, R.N. , Two Dimensional Imaging. Prentice Hall, 1995
5. www.ieeexplore.com
6. www.mathwork.com
7. www.wikipedia.com
8. https://arxiv.org/ftp/arxiv/papers/1505/1505.03489.pdf
9.https://ieeexplore.ieee.org/abstract/document/225929/