PES University, Bangalore

(Established under Karnataka Act No. 16 of 2013)

MAY 2020: IN SEMESTER ASSESSMENT (ISA) B.TECH. IV SEMESTER

UE18MA251- LINEAR ALGEBRA

MINI PROJECT REPORT

ON

Implementation of Dynamic Image Filters

Submitted by

1.	Rajath J	PES1201801707
2.	Rithvik Kumar	PES1201801794
3.	Chetan S R	PES1201801911

Branch & Section : ECE ,B

PROJECT EVALUATION

(For Official Use Only)

Sl.No.	Parameter	Max Marks	Marks Awarded
1	Background & Framing of the problem	4	
2	Approach and Solution	4	
3	References	4	
4	Clarity of the concepts & Creativity	4	
5	Choice of examples and understanding of the topic	4	
6	Presentation of the work	5	
	Total	25	

Name of the Course Instructor : RENNA SULTANA

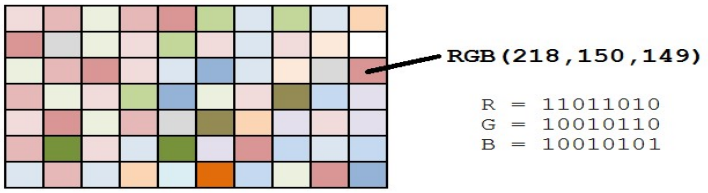
Signature of the Course Instructor :

Introduction

Digital Images and Matrices go hand in hand. A Digital Image in a computer is presented by a Pixel Matrix. The motive of our project is to perceive the nature of such Image matrices. Their change to the application of different filters. Such filters which change an Image are known as Kernels. Kernels have a major importance in Image processing. Every effect/filter for Images nowadays, Uses at least a one Kernel. Hence, they are of great importance in image processing. Generally realizing and visualizing the usage of such Kernels is difficult. As the mathematical computation involved is tedious. We wanted to simplify such tedious tasks. Our end goal was to create an application which helps us in realizing these. We also wanted to include the option for the user to give his own Kernel, see how it behaves and more importantly help visualize how an Image is transformed. Our application and all the below implementation are done using **MATLAB**. This report contains the case study we did for making our program.

Bitmaps :

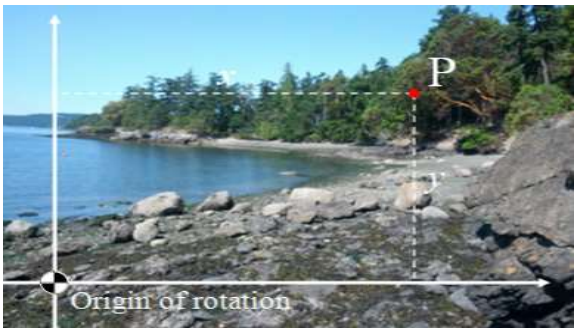
A typical computer image these days uses 24 bits to represent the color of each pixel. Eight bits are used to store the intensity of the **red** part of a pixel (**00000000** through **11111111**), giving 256 distinct values. Eight bits are used to store the **green** component, and eight bits are used to store the **blue** component.



Fundamental Image operations :

Rotation :

Each pixel has a coordinate pair *(x,y)* describing its position on two orthogonal axes from defined origin *O*. It is around this origin we are going to rotate our image.



To complete the rotation, we need to write the RGB value for each pixel in the new location. The *(x,y)* coordinates relative to the axes are the same; what we have done is rotate the

coordinate frame of reference by an angle θ. Using geometry, we can find out the relationship between the source coordinates *(x,y)* and the destination coordinates. The rotation matrix we thus get is given by $\mathbf{R_\Theta}$.

x*=x cos(theta) - y sin(theta)

y*= x sin(theta) + y cos(theta)

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

Original Image

Rotated Image

Here is our source image. By translating the coordinate frame, we can place the origin at the centroid, and it is around this we will rotate it. The matrix expands out as above. We can loop over the image for each *(x,y)* coordinate and find its new destination. The result with a rotation of $\theta = 15°$ is shown. But there's white spots visible in the rotated image due to Aliasing.

It is the same reason that we see jagged staircases on lines drawn on low resolution screens at angles other than horizontal and vertical. Raster screens are digital, and pixel boundaries are at quantized locations. Aliasing can be solved by using Area mapping.

Area Mapping :

For this, you invert the problem, and for each *destination* pixel, you find which four partial source pixels that it was created from. This algorithm not only ensures that there are no gaps in the destination, but also appropriately averages the colors (ensuring both a smoother image and also keeping the average brightness of the rotated image constant). The heart of this method is the expansion of the single 2D rotation matrix into three different matrices.

Scaling :

Bilinear Interpolation :

Bilinear interpolation, also known as first-order interpolation, linearly interpolates pixels along each row of the source image, then interpolates along the columns. Bilinear interpolation assigns to Point *D* in the destination a value that is a bilinear function of the four pixels nearest *S* in the source image.

Bilinear interpolation results in an improvement in image quality over nearest-neighbor interpolation but may still result in less-than-desirable smoothing effects.

Bilinear interpolation requires a neighborhood extending one pixel to the right and below the central sample. If the subsample position is given by (u, v), the resampled pixel value will be:

$$(1-v) * [(1-u) * p00 + u * p01] + v * [(1-u) * p10 + u * p11]$$

Bilinear Interpolation:

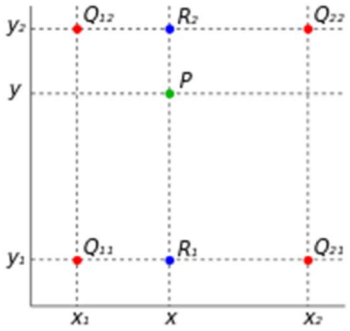
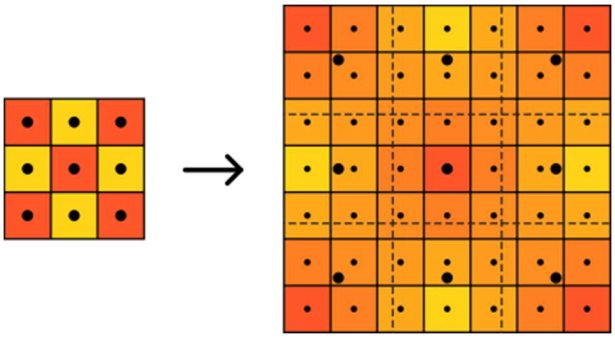


Image scaled using Bilinear Interpolation



Filters :

From the point of view of linear algebra, filters are applied to each pixel of the matrix using a filter function. The filter function simply performs 2D convolution over our Image using the Kernel matrix. The input of this function can be just a pixel like the adjustment of brightness, or a submatrix of pixels like the blur, where the order of the submatrix will depend on the blur ratio.

Let us consider the matrix M, as the matrix associated to a full colour image:

$$M = \begin{bmatrix} p11 & p12 & \cdots & p1n \\ p21 & p22 & \cdots & p2n \\ \vdots & \vdots & \ddots & \vdots \\ pm1 & pm2 & \cdots & pmn \end{bmatrix}$$

Here, p_{ij} is the pixel in the position (i, j), which is represented as the vector: $\begin{bmatrix} r \\ g \\ b \end{bmatrix}$

In the simplest case (the filter needs only a pixel as input), the function can be a linear transformation, that transforms a three-dimensional vector (pixel) into another three-dimensional vector, or not.

When it is a linear transformation, the transformation can be represented as a 3x3 matrix T, where:

$$\begin{bmatrix} r' \\ g' \\ b' \end{bmatrix} = T \bullet \begin{bmatrix} r \\ g \\ b \end{bmatrix}$$

As a simple example, consider the case of isolated pixels on a zero background:

```
>> a = zeros(9);
>> idx = [1 5 9];
>> a(idx,idx) = 1
a =
     1     0     0     0     1     0     0     0     1
     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0
     1     0     0     0     1     0     0     0     1
     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0
     1     0     0     0     1     0     0     0     1
```

Let T be a 2x2 max filter

```
>> maxfilt(a)
ans =
     1     1     0     1     1     1     0     1     1
     1     1     0     1     1     1     0     1     1
     0     0     0     0     0     0     0     0     0
     1     1     0     1     1     1     0     1     1
     1     1     0     1     1     1     0     1     1
     1     1     0     1     1     1     0     1     1
     0     0     0     0     0     0     0     0     0
     1     1     0     1     1     1     0     1     1
     1     1     0     1     1     1     0     1     1
```

We see that by simply find the max terms we were able to significantly brighten the image. This is what our filter Kernel do. Just by applying a simple transformation, they can significantly change our Image.
We dwell into some come filters below and the reason why we used those specific operations while making our application.

Grayscale Image :

The need to convert an image to its grayscale counter part frequently appears in Image processing. Gray images simplify computations and sometimes help prevent redundancy. The normal RGB to Gray uses the simple formula :

Grayscale image = ((0.3 * R) + (0.59 * G) + (0.11 * B))

Which is nothing but a simple linear operation.

Note : Simply taking the average of RGB is not a good idea as leads to loss in Image information.

The above formula comes from the fact that the green quotient of the image must have high weight in the end gray image. As almost all contrast is observed from the G matrix. Whereas R and B contribute more to the darkness of the Image.

Here is an example :

Original Image

Average Method

Formula Method

Edge detection:

Edge Detection is simply a case of trying to find the regions in an image where we have a sharp change in intensity or a sharp change in color, a high value indicates a steep change and a low value indicates a shallow change. For Edge detection we will be using the **Sobel Operator**.

Sobel Operator:

It approximates a derivative of an image. It is separate in the y and x directions. If we look at the x-direction, the gradient of an image in the x-direction is equal to this operator here. We use a kernel 3 by 3 matrix, one for each x and y direction. The gradient for x-direction has minus numbers on the left-hand side and positive numbers on the right hand side and we are preserving a little bit of the center pixels. Similarly, the gradient for y-direction has minus numbers on the bottom and positive numbers on top and here we are preserving a little bit on the middle row pixels.

-1	0	+1
-2	0	+2
-1	0	+1

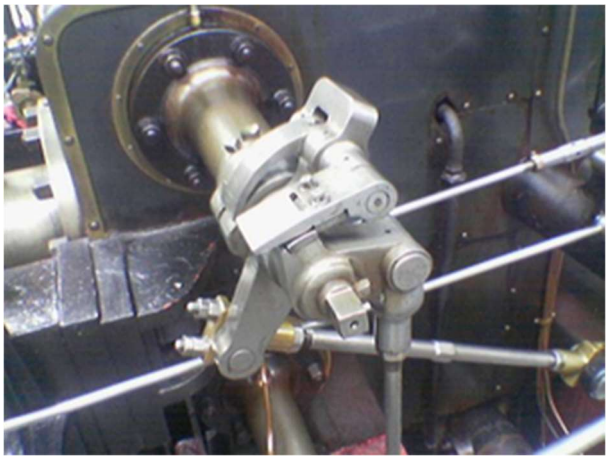
Gx

+1	+2	+1
0	0	0
-1	-2	-1

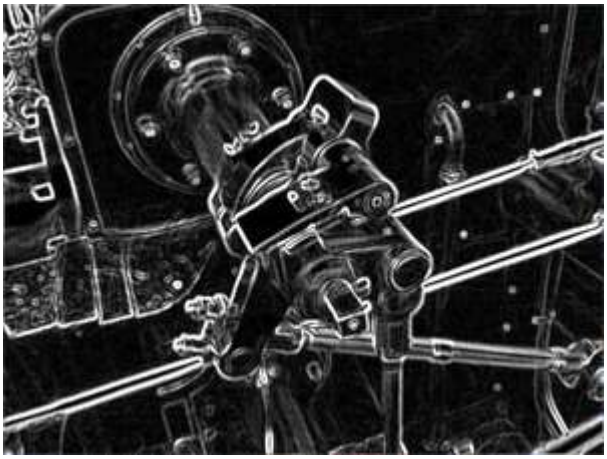
Gy

An Example of how it works :

Original Image



Sobel Operator filtered Image



100	100	200	200
100	100	200	200
100	100	200	200
100	100	200	200

-1	0	1
-2	0	2
-1	0	1

-100
-200
-100
200
400
+200
=400

Kernel Convolution: The bigger the value at the end, the more noticeable the edge will be.

The Sobel Operator method can be successfully used for segmenting an image without any human intervention. The Sobel Operator is very quick to execute as well. Since it produces the same output every time you execute it over an image, makes Sobel Operator a stable edge detection technique for image segmentation.

Blurring as a Linear operation And Removing Noise from a Blurred image :

Blurring, i.e., the operation of going from the sharp image to the blurred image, is *linear.*
This means that if B1 and B2 are the blurred images of the exact images X 1 and X2 ,
then ,

$$\mathbf{B} = a1\ B1 + a2\ B2$$

is the image of $\mathbf{X = a1\ X1 + a2\ X2.}$ When this is the case, then there exists a large matrix **A** such that **b = vec(B)** and **x = vec(X)** are related by the equation

A x = b.

The matrix A represents the blurring that is taking place in the process of going from the exact to the blurred image.

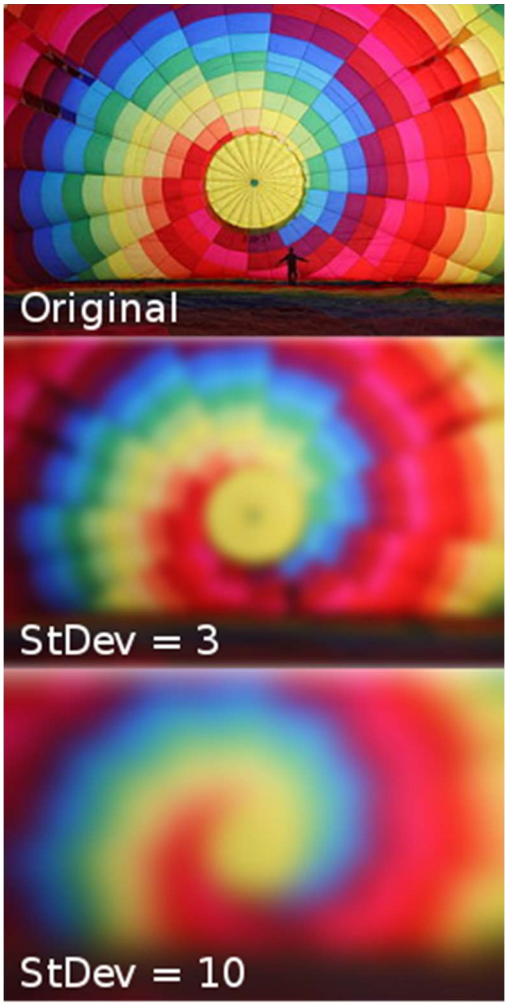
Gaussian , Bokeh and Mean Blur :

These are the three most significant blurs used in Image processing.

The **Mean blur** is basically a Matrix of ones which is normalized, this is the simplest type of blurs. But the Mean blur does not have much significance as it does not give a smooth blur as it takes equal weight from all the Adjacent Images.

Gaussian blur (also known as Gaussian smoothing) is the result of blurring an image by a Gaussian function (named after mathematician and scientist Carl Friedrich Gauss). It is a widely used effect in graphics software, typically to reduce image noise and reduce detail. The visual effect of this blurring technique is a smooth blur resembling that of viewing the image through a translucent screen. Gaussian smoothing is also used as a pre-processing stage in computer vision algorithms in order to enhance image structures at different scales.

Mathematically, applying a Gaussian blur to an image is the same as convolving the image with a Gaussian function. Applying a Gaussian blur has the effect of reducing the image's high-frequency components; a Gaussian blur is thus a low pass filter.



Gaussian Blur is based on the Gaussian formula

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

An Implementation of the Gaussian blur is done on the beside Image.

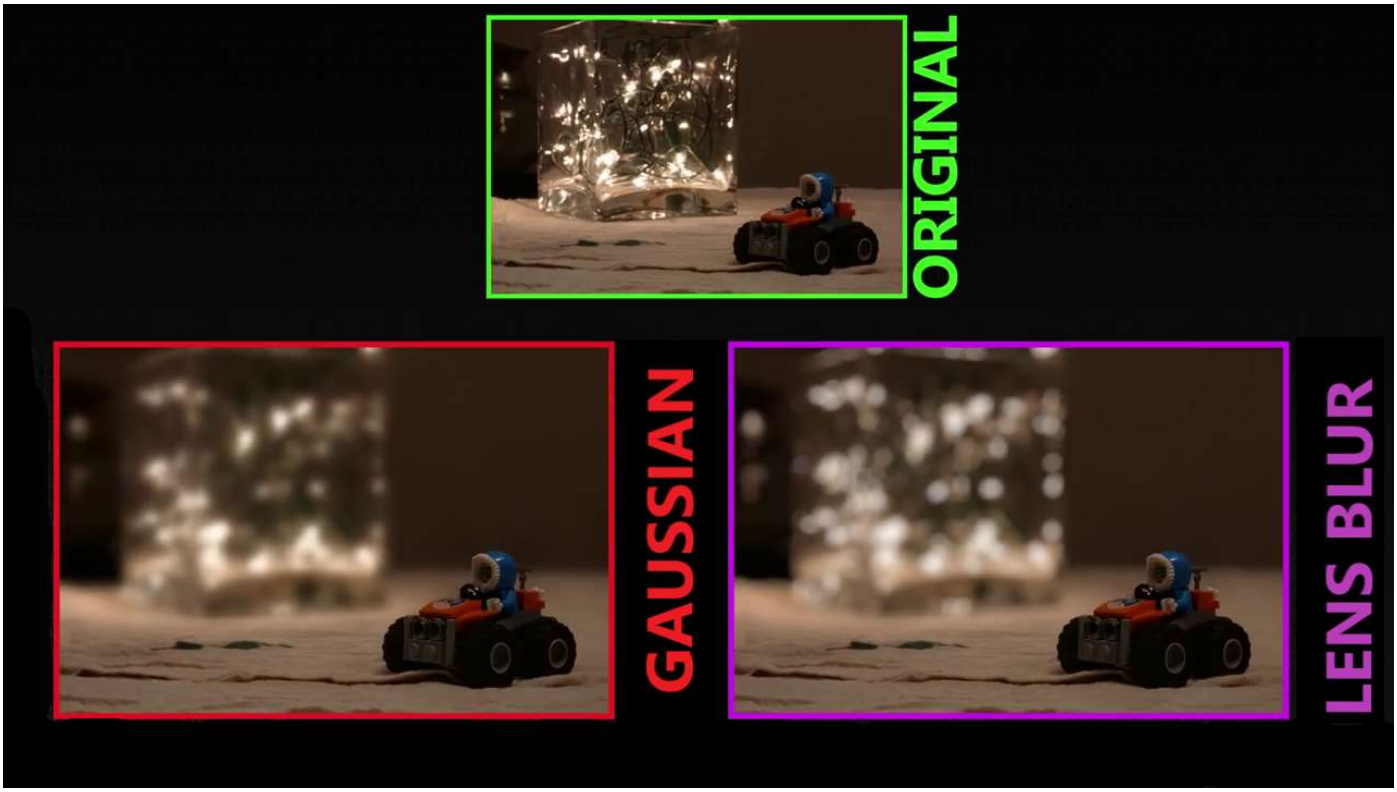
Just by changing the value of Sigma or StDev we are able to achieve different degrees of blurring.

The Gaussian Blur is also an important preprocessing filter while edge detection as it smoothens the Image and reduces the noise and false edges found in noisy images.

Bokeh, also known as “Boke” is one of the most popular subjects in photography. The reason why it is so popular, is because **Bokeh** makes photographs visually appealing, forcing us to focus our attention on an area of the image. The word comes from Japanese language, which literally translates as “blur”. Bokeh can be simulated by convolving the image with a kernel that corresponds to the image of an out-of-focus point

source taken with a real camera. Unlike conventional convolution, this convolution has a kernel that depends on the distance of each image point and – at least in principle – has to include image points that are occluded by objects in the foreground. Also, bokeh is not just any blur. To a first approximation, defocus blur is convolution by a uniform disk, a more computationally intensive operation than the "standard" Gaussian blur; the former produces sharp circles around highlights whereas the latter is a much softer effect. Diffraction may alter the effective shape of the blur. Some graphics editors have a filter to do this, usually called "Lens Blur." Bokeh is the quality of out-of-focus or “blurry” parts of the image rendered by a camera lens . The quality and feel of the background/foreground blur and reflected points of light. This blur is of great importance in photography and also in focus and out of focus feature available in the camera’s nowadays.

An implementation and comparison of Bokeh blur with Gaussian blur is given below:



Deblurring:

Visualizing image as vectors:

The key to obtaining the general linear model is to rearrange the elements of the images X and B into column vectors by stacking the columns of these images into two long vectors x and b, both of length $N = mn$. The mathematical notation for this operator is vec, i.e.,

x=[x1 ,x2 ,x3 ,x4 x5, x6,……..]’ € R^n and b=[b1 ,b2 ,b3 ,b4…..]’ € R^n

Since the blurring is assumed to be a linear operation, there must exist a large blurring matrix A € R^(NxN) such that x and b are related by the linear model

A x = b.

For our linear model, the naive approach to image deblurring is simply to solve the linear algebraic system Ax=b. However just solving the equation won't provide right kind of results because in many instances undesirable noise is added to 'b' due to many reasons such as shaky camera , quantization problems etc. Let us call the exact blurred image (i.e. the noise free blurred image) as - **B blurred** and its vector notation as **b-blurred** .

Also, the noise vector as **'e'** i.e $\mathbf{e} = \mathrm{vec}(\mathbf{E})$.

Then the noisy recorded image B is represented by the vector $\mathbf{b} = \mathbf{b}_{\text{exact}} + \mathbf{e},$

and consequently, the naive solution is given by

$$\mathbf{x}_{\text{naïve}} = \mathbf{A}^{-1}\mathbf{b} = \mathbf{A}^{-1}\mathbf{b}_{\text{exact}} + \mathbf{A}^{-1}\mathbf{e} = \mathbf{x} + \mathbf{A}^{-1}\mathbf{e},$$

where the term $\mathbf{A}^{-1}\mathbf{e}$ is called the inverted noise.

The important observation here is that the deblurred image consists of two components: the first component is the exact image, and the second component is the inverted noise. If the deblurred image looks unacceptable, it is because the inverted noise term contaminates the reconstructed image .

Concepts Used:

Important insight about the inverted noise term can be gained using the singular value decomposition (SVD). The SVD of a square matrix **A** is essentially unique and is defined as the decomposition

$$\mathbf{A} = \mathbf{U}\,\mathbf{\Sigma}\,\mathbf{V}^T$$

where **U** and **V** are orthogonal matrices and ∑= **diag((sigma(i))** is a diagonal matrix whose elements **sigma(i)** are nonnegative and appear in decreasing order

i.e sigma(1) > sigma(2) > sigma(3) > ……> sigma(N) > 0

The quantities sigma(i), are called the singular values, and the rank of **A** is equal to the number of positive singular values. The columns $\mathbf{u}_i$ of **U** are called the left singular vectors, while the columns $\mathbf{v}_i$ of **V** are the right singular vectors.

The inverse of A is now given by:

$$\mathbf{A}^{-1} = \mathbf{V}\,\mathbf{\Sigma}^{-1}\mathbf{U}^T$$

Observe that:

$$\mathbf{A} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^T$$

$$= \begin{bmatrix} \mathbf{u}_1 & \cdots & \mathbf{u}_N \end{bmatrix} \begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_N \end{bmatrix} \begin{bmatrix} \mathbf{v}_1^T \\ \vdots \\ \mathbf{v}_N^T \end{bmatrix}$$

$$= \mathbf{u}_1\sigma_1\mathbf{v}_1^T + \cdots + \mathbf{u}_N\sigma_N\mathbf{v}_N^T$$

$$= \sum_{i=1}^{N} \sigma_i \mathbf{u}_i \mathbf{v}_i^T .$$

=>

$$\mathbf{A}^{-1} = \sum_{i=1}^{N} \frac{1}{\sigma_i} \mathbf{v}_i \mathbf{u}_i^T$$

where the small letter notations correspond to the vectors.

The naïve solution to the equation i.e the (deblurred image – noise) is given by

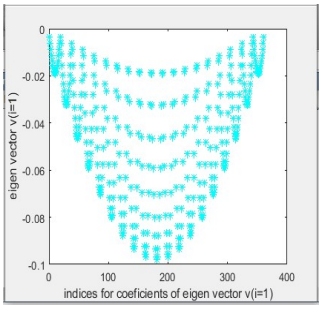
$$\mathbf{x}_{\text{naïve}} = \mathbf{A}^{-1}\mathbf{b} = \mathbf{V}\boldsymbol{\Sigma}^{-1}\mathbf{U}^T\mathbf{b}$$

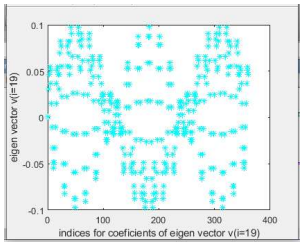
$$= \sum_{i=1}^{N} \frac{\mathbf{u}_i^T\mathbf{b}}{\sigma_i} \mathbf{v}_i \quad + \quad \sum_{i=1}^{N} \frac{\mathbf{u}_i^T\mathbf{e}}{\sigma_i} \mathbf{v}_i$$

Note that in the above equation the second term involves error ‘e’ , but we don’t have the access to error ‘e’ as it is generated from an unknown source and hence the vector ‘e’ is not available to us. So, in such a condition the general properties for image deblurring problems come to use

‘***’The error components*** $|u_i^T e|$ ***are small and typically of roughly the same order of magnitude for all i ’’***

‘’ The spatial frequencies of v_i***, increase with the index i.(Here by frequency we mean the number of times the signs of the eigen vector changes)’’***





First vector has 0 frequency (sign of the vector is same)

19th vector has higher frequency(sign of vector changes lot of time)

When we encounter an expansion of the form $\sum_{i=1}^{N} \varepsilon_i v_i$ then the 'i'th expansion coefficient ε_i measures the contribution of v_i, to the result. And since each vector v_i, can be associated with some "frequency," the 'i'th coefficient measures the amount of information of that frequency in our image.

Looking at the expression for A^{-1} e we see that the quantities $\frac{|u_i^T e|}{\sigma_i}$, are the expansion coefficients for the basis vectors v_i,. When these quantities are small in magnitude , the solution has very little contribution from v_i, but when we divide by a small singular value such as σ_i, we greatly magnify the corresponding error component $|u_i^T e|$, which in turn contributes a large multiple of the high-frequency information contained in the computed solution. This is precisely why a naive reconstruction, such as the one in figure appears as a random image dominated by high frequencies.

Because of this, we might be better off leaving the high-frequency components out altogether, since they are dominated by error. For example, for some choice of $k < N$ we can compute the truncated expansion

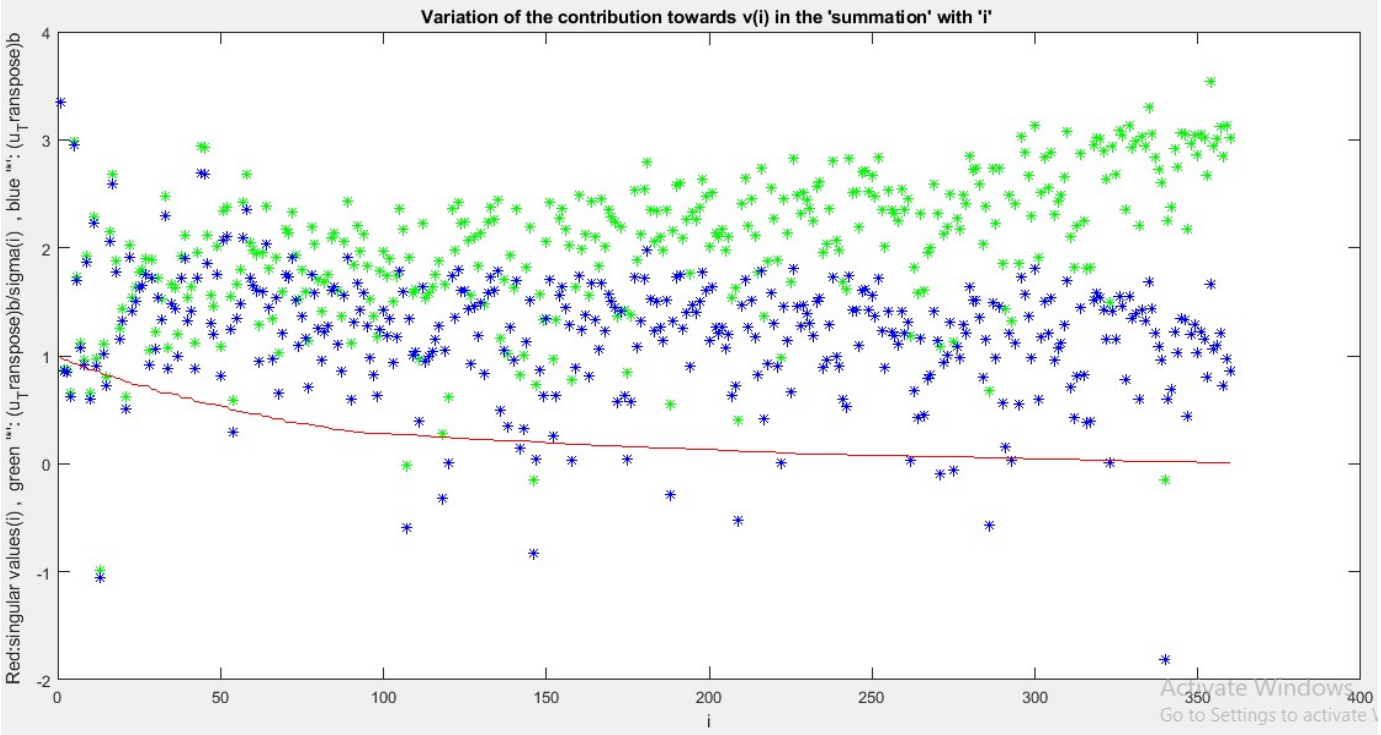
$$\mathbf{x}_k = \sum_{i=1}^{k} \frac{\mathbf{u}_i^T \mathbf{b}}{\sigma_i} \mathbf{v}_i \equiv \mathbf{A}_k^{\dagger} \mathbf{b}$$

in which we have introduced the rank-k matrix as:

$$\mathbf{A}_k^{\dagger} = \begin{bmatrix} \mathbf{v}_1 & \cdots & \mathbf{v}_k \end{bmatrix} \begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_k \end{bmatrix}^{-1} \begin{bmatrix} \mathbf{u}_1^T \\ \vdots \\ \mathbf{u}_k^T \end{bmatrix} = \sum_{i=1}^{k} \frac{1}{\sigma_i} \mathbf{v}_i \mathbf{u}_i^T$$

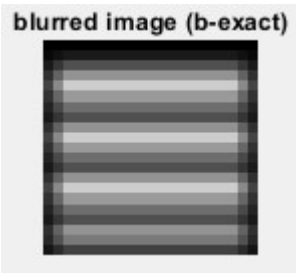
Note that the truncation of the summation at a value 'k' is accompanied with a loss of information in the actual blurred image b_{exact} .So the removal of 'noise' has a trade-off with loss of information in the actual blurred image.Hence the choice of 'k' should be such that it maintains a balance between both 'removal of noise' and 'loss of information'.

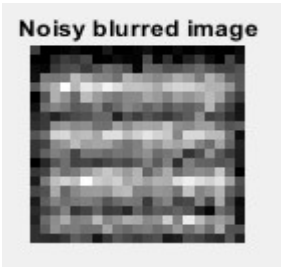
This is also roughly the reason why this technique is not 100% accurate in providing back the exact original image.

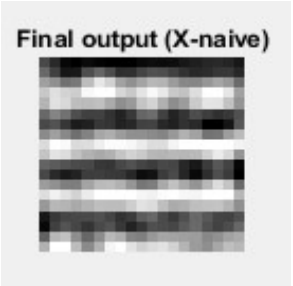


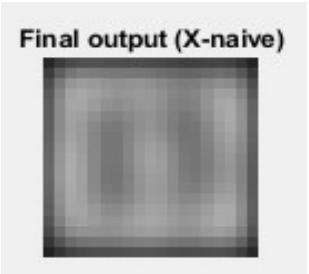
The red line corresponds to the singular values . It is observed that for the contribution $|u_i^T e|$(blue '*') , is amplified to $\frac{|u_i^T e|}{\sigma_i}$ (green '*') and this amplification is larger for lager values of 'i'.

Output:







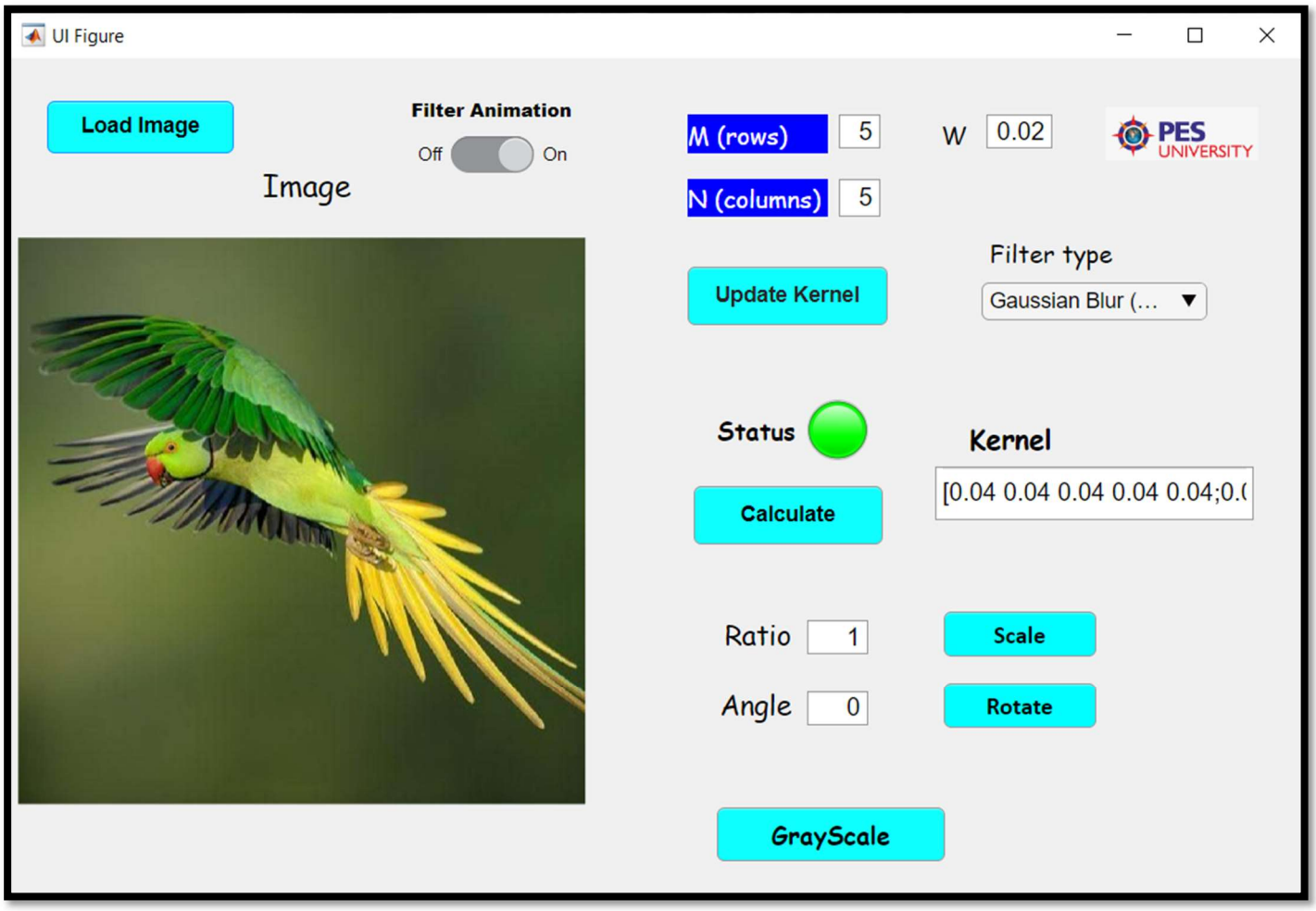


K=361
(No truncation)
(Lot of noise in the image)

K=83
(Optimal reconstruction)

K=20
(Too much of truncation)
(Loss of information to reconstruct the image)

Conclusion:



We were successfully able to build our application and most of the results here is taken from the output of our application. We have implemented all the Kernels and provided the option for the user to give his own kernel and observe different results. This option now allows for experimentation without dwelling in to heavy coding. There is also an animation option which now lets anyone observe how the image gets transformed. Deblurring the image was not implemented as it was a more computation intensive process and thereby was not suitable for our end application. This application can still be improved by including still more filters. Our program and all the implementation code can be found **here**. **Thank You.**

Bibliography:

For Bitmaps,rotation matrix ,scaling:

http://datagenetics.com/blog/august32013/index.html

https://www.nibcode.com/en/blog/1137/linear-algebra-and-digital-image-processing-part-III-affine-transformations

https://colececil.io/blog/2017/scaling-pixel-art-without-destroying-it/

https://en.wikipedia.org/wiki/Bilinear_interpolation

For Filters:

https://www.youtube.com/watch?v=uihBwtPIBxM

https://medium.com/datadriveninvestor/understanding-edge-detection-sobel-operator-2aada303b900

https://www.youtube.com/watch?v=C_zFhWdM4ic

https://www.youtube.com/watch?v=vNG3ZAd8wCc

https://photographylife.com/what-is-bokeh

https://www.nibcode.com/en/blog/1138/linear-algebra-and-digital-image-processing-part-IV-image-editor

For Image deblurring and removal of Noise :

https://www.researchgate.net/post/What_are_the_significance_of_singular_value_decomposition

Deblurring Images: Matrices, Spectra, and Filtering

By Per Christian Hansen, James G. Nagy, Dianne P. O'Leary