ESCUELA SUPERIOR POLITECNICA DEL

LITORAL

Facultad de Ingeniería en Electricidad y Computación

“Análisis matemático e implementación de algoritmos iterativos para degradar y restaurar imágenes utilizando el criterio de minimización de rizado”

Informe de Materia de Graduación

Previa a la obtención del Título de:

INGENIERO EN ELECTRÓNICA Y TELECOMUNICACIONES

Presentada por:

José Saquinaula Brito

Angelo Miranda Muñoz

GUAYAQUIL – ECUADOR

AÑO
2009

A G R A D E C I M I E N T O

Agradecemos:

Primeramente a Dios por la terminación de este trabajo.

A la Ing. Patricia Chávez, por su colaboración y ayuda para la culminación de nuestra carrera.

DEDICATORIA

José Luis Saquinaula Brito:

Agradezco primero a Dios por darme las fuerzas de seguir hasta el final de mi carrera de ingeniería. A mis padres Luis Saquinaula y Eulalia Brito por todo el apoyo recibido. Su esfuerzo, paciencia, dedicación y sacrificio hicieron de este sueño, un logro alcanzado. A mis amigos muy en especial a mi amigo del alma Freddy De La Rosa, aquellos que me acompañaron en momentos buenos y malos, las historias vividas que tendré siempre presentes. Un reconocimiento a mis profesores que me supieron impartir sus conocimientos para llegar a ser un gran profesional. Gracias a todos.

Ángelo Francisco Miranda Muñoz:

A Dios, a mi familia especialmente a mis padres y a todas las personas que con su apoyo y motivación han hecho posible la culminación de la etapa más importante de mi vida.

TRIBUNAL DE SUSTENTACIÒN

Ing. Patricia Chávez
DIRECTOR DEL PROYECTO

MSc. Rebeca Estrada P.
PROFESOR DELEGADO

DECLARACIÓN EXPRESA

"La responsabilidad del contenido de este proyecto, nos corresponde exclusivamente; y el patrimonio intelectual de la misma a la ESCUELA SUPERIOR POLITECNICA DEL LITORAL"

(Reglamento de Graduación de la ESPOL)

_____

José L. Saquinaula Brito

_____

Angelo F. Miranda Muñoz

RESUMEN

Este documento trata acerca del procesamiento digital de imágenes en el que simulamos un tipo real de desenfoque causado por el movimiento de la cámara o el objeto al tomar la foto. Mediante una convoluciòn y el concepto de matrices circulantes de bloques definimos una matriz de respuesta al impulso para la degradación de la imagen.

Para la restauración utilizamos el criterio de minimización de rizado, y describimos algunos métodos como son la de similitud y la de frecuencia, éste ultimo llamado de la pseudo-inversa.

Al final del documento detallamos como manejar la interfaz grafica y el código correspondiente para que el usuario pueda hacerle alguna mejora que crea conveniente.

INDICE GENERAL

RESUMEN……..VI

ÍNDICE GENERAL……VII

INDICE DE FIGURAS……VIII

INDICE DE TABLAS……IX

INTRODUCCIÓN……...1

1. DEGRADACION DE LA IMAGEN…….....3

1.1 Degradación lineal uniforme…….....3

1.2 Convolución…….....4

1.3 Análisis matemático para el código de degradación……..6

2. RESTAURACION DE LA IMAGEN…….....12

2.1 Criterio de minimización de rizado …….....12

2.2 Criterio de similitud…….....17

2.3 Criterio de frecuencia……...18

2.4 Análisis matemático para el código de restauración……....21

3. ANALISIS DE RESULTADOS……...26

3.1 Resultado de imagen degradada…….....26

3.2 Resultado de imagen restaurada…….....27

3.3 Desempeño de la restauración..……...29

CONCLUSION..……32

BIBLIOGRAFIA…….....34

ANEXOS…….....35

ANEXO A: Manejo del software……..35

ANEXO B: Código Fuente……...41

INDICE DE FIGURAS

Figura 1.1. imagen original, b) imagen degrada horizontalmente) imagen degradada verticalmente……...4
Figura 1.2. Diagrama de bloque de un sistema lineal…….5
Figura 1.3. Bloque de degradación de una fila de la imagen…….6
Figura 2.1. Modelo de solución……12
Figura 2.2. Restauración de una fila: a) fila original, b) fila degrada con n = 2, c), d), e) y f) son cuatro posibles soluciones que satisfacen HF^ = G……...14
Figura 2.3 Matriz R……..…….....22
Figura 2.4 Matriz Inversa de R……22
Figura 2.5 Matriz [$T^T.T$]……23
Figura 2.6 Matriz $\left(T^T.T\right)^{-1}$ ……23
Figura 2.7 Matriz T $\left(T^T.T\right)^{-1}$ T^{T}……....24
Figura 2.8 Matriz I - T $\left(T^T.T\right)^{-1}$ T^{T}……...25
Figura 2.9 Resultado de la Matriz I - T $\left(T^T.T\right)^{-1}$ T^{T}……25
Figura 3.1. Degradaciòn horizontal en imágenes en blanco y negro……26
Figura 3.2. Degradación horizontal en imágenes a color……27
Figura 3.3. Imágenes en blanco y negro restauradas……..27
Figura 3.4. Imágenes a color restauradas……..28
Figura 3.5. Restauración de imágenes a) imagen original, b) imagen restaurada con n = 30 de degradación y c) imagen restaurada con n = 40……..28
Figura 3.6. Desempeño de la restauración……30

INDICE DE TABLAS

Tabla 3.1 Valores del ISNR para una Imagen en blanco y negro……31
Tabla 3.2 Valores del ISNR para una Imagen a color……31

INTRODUCCION

Cualquier persona que haya tomado una fotografía entenderá que capturar una imagen exactamente como aparece en el mundo real es muy difícil, si no imposible. Está el ruido contenido, que en el caso de una fotografía puede ser causado por un movimiento de la cámara o del objeto; problemas de enfoque y una calidad imperfecta empeoran hasta el mejor sistema de lentes. El resultado de todas estas degradaciones hace que la imagen (fotografía) sea tan sólo una aproximación de la escena.

Muy frecuentemente la imagen es suficientemente buena para el propósito para el cual fue realizada. Por otra parte, hay algunos casos donde la corrección de la imagen por ordenador es la única manera de obtener imágenes útiles. Los recientes problemas con el Telescopio Espacial Hubble son un ejemplo: la óptica de las imágenes producidas no se aproximaba al potencial del telescopio, y una misión de reparación inmediata no era posible. Los ordenadores fueron utilizados para reparar algunas distorsiones causadas por la óptica y el resultado dio imágenes que eran de alta calidad.

En este proyecto degradamos la imagen a través de un movimiento lineal, sea éste horizontal o vertical, lo cual es seleccionado por el

usuario. Primero se capturarán las imágenes, luego se la manipula utilizando un algoritmo de degradación en el que se aplica convolución. Cada fila de la imagen de entrada se representa como f y cada fila de imagen degradada como g. Se aplica convolución entre f y una matriz h, con lo cual se obtiene g.

Para la restauración utilizamos el criterio de minimización de rizado el cual lo podemos realizar con algunos métodos. El que se utiliza para este proyecto es el de criterio de la frecuencia.

Se emplearon varias herramientas de software que interactúan para dar un mejor resultado. Estas herramientas son:

- JAVA.- Es un lenguaje ampliamente utilizado, independiente de la plataforma, así, podemos crear un software bajo Linux y correrlo en un sistema Windows o Mac sin necesidad de hacer cambios al código. Se empleó la Versión: 1.6.10.
- PROCESSING.- Es un software para procesamiento de imágenes compatible con JAVA. Se utilizó la Versión: 1.0.
- MATHEMATICA.- Es un software para realizar cálculos matemáticos. Se empleó la Versión 5.2

CAPÌTULO 1

1. DEGRADACION DE LA IMAGEN

1.1. Degradación lineal uniforme

En esta parte del proyecto vamos a simular un caso real de degradación o daño al tomar una foto. Si existe un movimiento relativo, sea este horizontal o vertical entre la cámara y el objeto al que se le tomará la foto, la cámara va a capturar información de una escena más ancha que la que hubiese tomado sin movimiento, entonces la imagen que luego será restaurada, tendrá más columnas que la imagen degradada.

La figura 1.1 muestra como se verá la degradación cuando el movimiento es horizontal y vertical.

Que la degradación sea uniforme significa que la matriz respuesta al impulso tiene como elementos una constante (lo detallaremos en la sección 2.3).

Además no solo se limita a imágenes en blanco y negro sino también a imágenes con color y como se trata de una simulación necesitaremos una buena imagen (que llamaremos luego imagen original) para posteriormente degradarla.

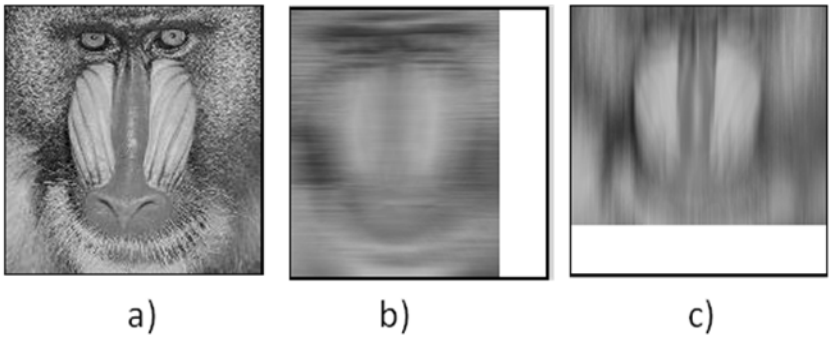


Figura 1.1 Degradación Horizontal y Vertical
a) imagen original, b) imagen degradada horizontalmente
c) imagen degrada verticalmente.

1.2. Convolución

La herramienta matemática principal en el proceso de degradación es la convoluciòn. El tratamiento de imágenes más empleado y conocido, es el tratamiento espacial también conocido como convolución. Las convoluciones discretas son muy usadas en el procesado de imagen para el suavizado de imágenes, el afilado de imágenes, detección de bordes, y otros efectos. Mediante este proceso se calcula el valor de un determinado punto en función de su valor y del valor de los puntos que le rodean, aplicando una simple

operación matemática en función de la cual se obtendrá un valor resultante para el punto en cuestión.

Una manera formal de definir la Convolución sería decir que:

Es una operación matemática que nos ayuda a obtener la salida de un sistema de manera relativamente sencilla analizado en el dominio del tiempo o la frecuencia, esta salida depende de la señal de entrada al sistema y la respuesta al impulso del mismo.

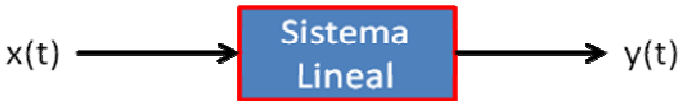


Figura 1.2 Diagrama de bloque de un sistema lineal

$$y(t) = x(t) * h(t)$$

Para señales en una dimensión (1D) la convolución se define como:

$$(f*g)(t)=\int_{-\infty}^{\infty} f(\tau)\cdot g(t-\tau)\partial\tau=\int_{-\infty}^{\infty} g(\tau)\cdot f(t-\tau)\partial\tau$$

Convolución en el dominio del tiempo

$$(f*g)[n]=\sum_{m=-\infty}^{\infty} f[m]\cdot g[n-m]=\sum_{m=-\infty}^{\infty} g[m]\cdot f[n-m]$$

Convolución en el dominio de la frecuencia

Para señales en dos dimensiones (2D) la convolución se define como:

$$(f*g)(t_1,t_2)=\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(\tau_1,\tau_2)\cdot g(t_1-\tau_1,t_2-\tau_2)\partial\tau_1\partial\tau_2$$

Convolución en el dominio del tiempo

$$(f*g)[n_1,n_2]=\sum_{k_1=-\infty}^{\infty}\sum_{k_2=-\infty}^{\infty} f[k_1,k_2]\cdot g[n_1-k_1,n_2-k_2]$$

Convolución en el dominio de la frecuencia

1.3. Análisis matemático para el código de degradación.

La degradación de una imagen lo realizamos utilizando la teoría de sistemas lineales y operación de matrices. La figura 1.3 es un diagrama de bloque para representar la degradación perteneciente a una fila de la imagen.

Figura 1.3: Bloque de degradación de una fila de la imagen

Primero definamos las variables de entrada y salida de este sub sistema:

- F: representa una fila de la imagen original
- H: representa un vector de n entradas que se define como la respuesta al impulso de la degradación
- G: representa una fila de la imagen degradada

La degradación que vamos a modelar matemáticamente para obtener el código es el de movimiento lineal tanto horizontal como vertical. Adicionalmente, empleando operaciones básicas de matrices y el concepto de convolución.

A continuación se hará el análisis para degradar una fila de la imagen de entrada o también llamada imagen original y obtener la expresión matemática de cada elemento de la fila de la imagen degradada. De esta manera notaremos la secuencia de estos números y se obtiene el código correspondiente a la degradación de la imagen.
Simplemente para recorrer toda la fila utilizaremos lazos for en JAVA para hacer el análisis de todas las filas de la imagen original.

Dada la imagen que tiene M filas, la entrada al sistema de degradación es una fila de la imagen original representada por F agrupada en un vector $F = (f_1 \ f_2 \ f_3 \ ..... f_M)^T$ de M elementos correspondientes a esa fila. Mientras que la salida del sistema es una

fila de la imagen degradada representada por G agrupada en un vector $G = \left(g_1 \;\; g_2 \;\; g_3 \;\; ..... \; g_N \right)^T$ de N elementos.

La degradación lineal se la realiza mediante una convolución entre f y h en donde “la matriz h representa un vector de n entradas que se define como la respuesta al impulso del proceso de degradación o bien conocido como PSF” [1]. La matriz H es:

$$H = \begin{pmatrix} h_1 \;\; .... \; h_n \;\; 0 \;\; 0 \;\; 0 \;\; 0 \\ 0 \;\; h_1 .... \; h_n \;\; 0 \;\; 0 \;\; 0 \\ \vdots \qquad\qquad \vdots \\ 0 \;\; 0 \;\; .... \; 0 \;\; h_1 ....h_n \end{pmatrix}$$

Todo lo expresado anteriormente matemáticamente se expresa como:

$$HF = G$$

$$g = f * h = \begin{pmatrix} h_1 \;\; .... \; h_n \;\; 0 \;\; 0 \;\; 0 \;\; 0 \\ 0 \;\; h_1 .... \; h_n \;\; 0 \;\; 0 \;\; 0 \\ \vdots \qquad\qquad \vdots \\ 0 \;\; 0 \;\; .... \; 0 \;\; h_1 ....h_n \end{pmatrix} \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_M \end{pmatrix} = \begin{pmatrix} g_1 \\ g_2 \\ \vdots \\ g_N \end{pmatrix} \quad \textbf{(1)}$$

Como la degradación correspondiente a un movimiento lineal uniforme de n píxeles, los elementos de la matriz H son constantes:

$H = (h_1 \;\; h_2 \;\; h_3 \;\; ..... \; h_n)^T$ donde $h_i = 1/n$. Es importante que los elementos de h deban cumplir con la siguiente condición $\sum h_i = 1$ para que la imagen degradada no sea ni más oscura ni más clara que la imagen original.

En esta ecuación se cumple:

$$M = N + n - 1 \qquad \textbf{(2)}$$

Las dimensiones de la fila degradada (G) son de Nx1 mientras que las dimensiones de la fila original (F) son de Mx1, para que se pueda realizar la multiplicación la matriz H debe tener dimensión NxM.

Para entender este proceso y realizar el código correspondiente supondremos que una fila de la imagen original tiene 4 elementos, por lo tanto M = 4. $F = (f_1 \;\; f_2 \;\; f_3 \;\; f_4)^T$. Vamos a desplazar 2 pixeles para degradarla, por lo tanto:

n = 2. $h = (h_1 \;\; h_2)^T$.

Con los valores de M y n obtenemos el valor de N de la ecuación (2)

N = M – n +1

N = 4 -2 +1

N = 3

Por lo tanto las dimensiones de las matrices son:

- Matriz F: 4x1
- Matriz H: 3x4
- Matriz G: 3x1

La ecuación (1) aplicado a este ejemplo será:

$$g = \begin{pmatrix} 1/n & 1/n & 0 & 0 \\ 0 & 1/n & 1/n & 0 \\ 0 & 0 & 1/n & 1/n \end{pmatrix} \begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \end{pmatrix} = \begin{pmatrix} g_1 \\ g_2 \\ g_3 \end{pmatrix}$$

Realicemos la multiplicación para obtener cada elemento de la fila degradada.

$$g_1 = \frac{1}{n}(f_1 + f_2)$$
$$g_2 = \frac{1}{n}(f_2 + f_3)$$
$$g_3 = \frac{1}{n}(f_3 + f_4)$$

Para una imagen con M = 5 y n = 3 los elementos de la fila

$$g_1 = \frac{1}{n}(f_1 + f_2 + f_3)$$
$$g_2 = \frac{1}{n}(f_2 + f_3 + f_4)$$
$$g_3 = \frac{1}{n}(f_3 + f_4 + f_5)$$

La secuencia de estos elementos de la matriz (G) da como el resultado el código desarrollado en processing-1-0.1 que se puede encontrar en el apéndice “Códigos Fuente”

CAPÌTULO 2

2. RESTAURACION DE LA IMAGEN

.2.1 Criterio de minimización de rizado

La restauración consiste en recuperar de la mejor manera posible la fila de la imagen original (vector F). Con el concepto de invertibilidad de sistemas lineales invariantes en el tiempo que dice: “Un sistema invariante en el tiempo TI tiene la propiedad de que cierta entrada siempre dará la misma salida, sin consideración alguna a cuando la entrada fue aplicada al sistema” [5].

Para restaurar la imagen original (F) es posible realizar un procedimiento inverso aplicado a la imagen degradada (G) como se muestra en la figura 2.1

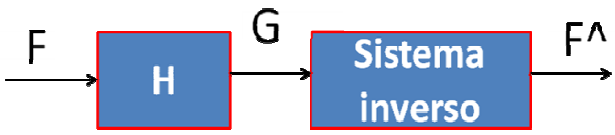


Figura 2.1 Modelo de solución

La señal restaurada la denotamos con f^ y se almacena en un vector F^. De la ecuación (1) se desprende que el sistema de ecuaciones tiene M variables y N ecuaciones. Dado que M>N, el sistema no tiene solución única sino que tiene infinitas soluciones para F^ que satisfacen la ecuación Hf^= g.

Las soluciones en la gran mayoría no darán el resultado esperado para la imagen restaurada debido a que muchas de ellas presentan oscilaciones considerables y debemos obtener una respuesta para F^ donde se disminuya estas oscilaciones o sea lo que se llama minimizar el rizado

La Figura 2.2 muestra un ejemplo en el que se aprecia una señal F de 50 muestras y su correspondiente degradación obtenida con:

$$H = [0.5\ 0.5]^T.$$

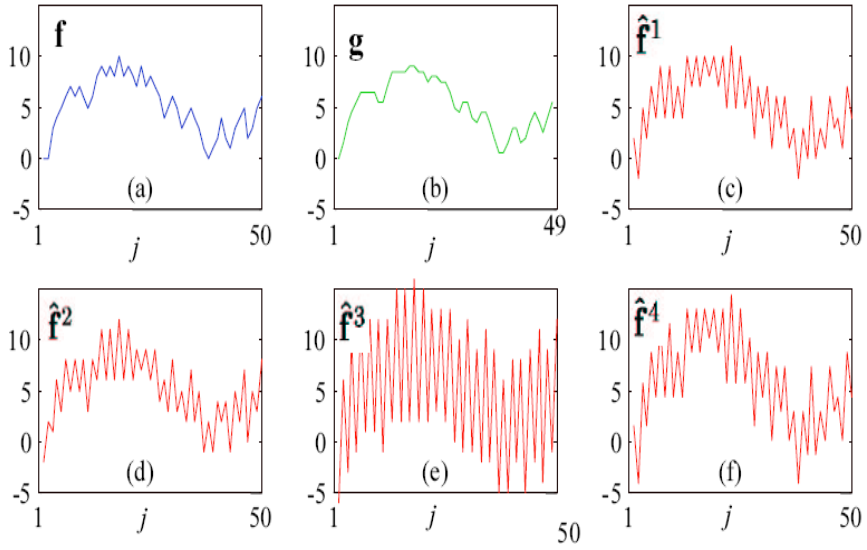


Figura 2.2 Restauración de una fila
a) fila original, b) fila degrada con n = 2, c), d), e) y f) son cuatro posibles soluciones que satisfacen HF^ = G.

En esta figura f^^{1}, f^^{2}, f^^{3} y f^^{4} representan soluciones para la ecuación Hf^^{k} = g pero no son satisfactorias debido a que tienen bastante distorsión.

Para minimizar el rizado o suavizar la señal

Esta condición de suavizado se obtiene forzando una similitud entre las imágenes. El problema es que f^ y g no tienen el mismo número de elementos.

Lo que se hace es lo siguiente: Minimizamos $||f\hat{}_N - g||$, donde $f\hat{}_N$ es definido como los primeros *N* elementos de f_N . Con la restricción $||Hf\hat{} - g|| = 0$. Tratando de hacer cumplir de que se encuentre la mejor solución para eliminar las oscilaciones.

Cuidado $f\hat{}_N$ y g no son iguales ya que esta solución no cumpliría necesariamente la restricción $||Hf\hat{} - g|| = 0$ porque hay más de una solución. Pero utilizando el concepto de derivada para hallar mínimos podemos encontrar la mejor solución que minimice el rizado.

Matemáticamente, este problema de optimización se plantea usando la siguiente función:

$$V(\hat{f}) = \lambda \left\| H\hat{f} - g \right\|^2 + \left\| P\hat{f} - g \right\|^2 \to \min, \qquad \textbf{(3)}$$

Donde λ es un número muy grande conocido como multiplicador de Lagrange que es “Un método para trabajar con funciones de varias variables que nos interesa maximizar o minimizar, y está sujeta a ciertas restricciones” [6]

La matriz P de dimensiones NxM es definida de tal forma que

$Pf\hat{} = f\hat{}_N$, [1] es decir:

$$P = \begin{pmatrix} 1 & 0 \ldots 0 & 0 \ldots 0 \\ 0 & 1 \ldots 0 & 0 \ldots 0 \\ & : & : \ 0 \\ 0 & 0 \ldots 1 & 0 \ldots 0 \end{pmatrix}$$

La solución se obtiene derivando V de la ecuación (3) en función de f^ e igualando a cero, por lo tanto:

$$\hat{f} = \left[\lambda H^T H + P^T P\right]^{-1} + \left[\lambda H + P\right] g \qquad \textbf{(4)}$$

A continuación vamos a presentar otras restricciones o métodos para hacer cumplir la minimización del rizado, las cuales se basan en:

- la similitud entre $f\hat{}_N$ y g.
- la minimización de los armónicos de $f\hat{}$.

Se analizara de manera general la forma de cómo restaurar la imagen mediante estos criterios. Este proyecto resuelve el problema de la restauración mediante un método directo aplicado al criterio de la frecuencia o minimización de armónicos en el cual no depende del multiplicador de Lagrage (λ) ya que encontrar estos valores no es fácil y debemos seguir ciertos criterios matemáticos.

2.2 Criterio de similitud

Se considera que dos señales son similares si están correlacionadas entre sí. Existen varias medidas o puntos de vista que sirven para evaluar la similitud entre dos señales, tres de ellas son la correlación, la covarianza y el coeficiente de correlación. Si desea profundizar en estos conceptos tome en cuenta la referencia [7].

La solución buscada es una señal $\hat{f}_N$ que minimice $||H\hat{f} - g||$. El criterio de similitud puede plantearse de las siguientes maneras:

- Criterio de similitud 1: Correlación

$$V(\hat{f}) = \left[\left\|H\hat{f} - g\right\|\right]\left[\left\|g^T \hat{f}_N - g\right\|^{-1}\right] \rightarrow \min \quad \textbf{(5)}$$

- Criterio de similitud 2: Covarianza

$$V(\hat{f}) = \left[\left\|H\hat{f} - g\right\|\right]\left[\left\|g - u_g\right\|^T\right]\left[\left\|\hat{f}_N - u_{fN}\right\|^{-1}\right] \rightarrow \min \quad \textbf{(6)}$$

- Criterio de similitud 3: Coeficiente de correlación

$$V(\hat{f}) = \left[\left\|H\hat{f} - g\right\|\right]\frac{\left[\left\|g - u_g\right\|^T\right]\left[\left\|\hat{f}_N - u_{fN}\right\|\right]^{-1}}{\left[\sigma_s \sigma_{fN}\right]^{-1}} \rightarrow \min \quad \textbf{(7)}$$

2.3 Criterio de frecuencia

En este procedimiento vamos a minimizar una señal sˆ que se obtiene a partir de un filtro pasa alto definido para fˆ, es decir:

||s^|| = ||fˆ*w|| = ||Wfˆ||

Donde w se define como "el vector que representa la respuesta al impulso del filtro pasa alto, y W es la matriz circulante respectiva definida a partir de w" [1], de la misma manera como H fue definido a partir de h en la ecuación (1).

De esta forma la función a minimizar será:

$$V(\hat{f}) = \left\| W\hat{f} \right\|^2 = v\hat{(}\theta) = \lambda \left\| WSg + WT\hat{\theta} \right\|^2 + \rightarrow \min \quad \textbf{(8)}$$

En el criterio de frecuencia también se utiliza el multiplicador de Lagrage conocido como el parámetro de regularización.

La minimización de V (f^) se obtiene derivando V con respecto a f^ para luego igualar a cero, es decir:

:

$$\hat{f} = \left[\lambda H^T H + W^T W \right]^{-1} + H^T g \qquad \textbf{(9)}$$

Donde se utiliza la condición de que $W^T W = I$. Todas las matrices que cumplen con esta condición son llamadas matrices pseudo inversa. Si desea profundizar en este concepto tome en cuenta la referencia [8].

Por lo tanto la ecuación (9) queda:

$$\hat{f} = \left[\lambda H^T H + I \right]^{-1} + H^T g \qquad \textbf{(10)}$$

Los métodos explicados dependen del parámetro λ y escoger este valor no es trivial y hay que hacer un análisis respectivo. Por este motivo nuestro proyecto se basa en un método directo que no depende de λ.

Lo que hacemos es tomar los n – 1 elementos de f e igualarlas a las variables $\theta_1,\ldots, \theta_{n-1}$. De esta manera la ecuación (1) puede ser planteada de la siguiente manera:

$$\begin{bmatrix} h_1 & h_2 & \ldots & h_n & 0 & 0 & 0 & 0 & 0 \\ 0 & h_1 & h_2 & \ldots & h_n & 0 & 0 & 0 & 0 \\ & \therefore & & & \therefore & & & \therefore & \\ 0 & 0 & 0 & \ldots & h_1 & h_2 & \ldots & h_n & 0 \\ 0 & 0 & 0 & \ldots & 0 & h_1 & h_2 & \ldots & h_n \\ \hline 0 & 0 & 0 & \ldots\ldots\ldots & 0 & 1 & 0 & \ldots\ldots\ldots & 0 \\ 0 & 0 & 0 & \ldots\ldots\ldots & 0 & 0 & 1 & \ldots\ldots\ldots & 0 \\ & \therefore & & & \therefore & & & \therefore & \\ 0 & 0 & 0 & \ldots\ldots\ldots & 0 & 0 & 0 & \ldots\ldots\ldots & 1 \end{bmatrix} \begin{bmatrix} f_1 \\ f_2 \\ \therefore \\ f_{N-1} \\ f_N \\ \hline f_{N+1} \\ f_{N+2} \\ \therefore \\ f_M \end{bmatrix} = \begin{bmatrix} g_1 \\ g_2 \\ \therefore \\ g_{N+1} \\ g_N \\ \hline \theta_1 \\ \theta_2 \\ \therefore \\ \theta_{N-1} \end{bmatrix} \qquad \textbf{(11)}$$

Cuya representación matricial es:

$$Rf = [\, g^T \theta^T \,]^T$$

Podemos escribir la ecuación matricial anterior para verlo más fácilmente como:

$$f = \left[\frac{f_N}{\theta} \right] = R^{-1} \left[\frac{g}{\theta} \right] = Sg + T\theta \qquad \textbf{(12)}$$

R^{-1} es la matriz que necesitamos tener para empezar nuestro análisis matemático y posteriormente desarrollar los códigos. La matriz R^{-1} será dividida en dos matrices más pequeñas llamadas S y T definidas como las primeras N columnas de R^{-1} y las últimas (*n* - 1) columnas de R^{-1}. De esta manera redefinimos la función que debemos minimizar quedando de la siguiente manera:

$$V(\hat{\mathbf{f}}) = \| \mathbf{W}\hat{\mathbf{f}} \|^2 = V(\hat{\theta}) = \| \mathbf{WSg} + \mathbf{WT}\hat{\theta} \|^2 \rightarrow \min \qquad \textbf{(13)}$$

El problema de restauración consiste ahora en analizar la variable θˆ para que minimice la expresión:

$$||WSg + WT\theta^{\wedge}||$$

Siendo W, S, T y g conocidos.

Se emplea la técnica de mínimos cuadrados, es decir, se encuentra θˆ tal que $||X\theta - y||^2$ sea mínimo. En nuestro caso la solución quedaría:

$$\hat{\theta} = -\left[T^T W^T W T\right]^{-1} T^T W^T W S g. \qquad \textbf{(14)}$$

Conociendo θˆ se determina fˆ por sustitución directa en (12)

$$\hat{f} = \left(I - T\left[T^T W^T W T\right]^{-1} T^T W^T W\right) S g. \qquad \textbf{(15)}$$

2.4 Análisis matemático para el código de restauración

A continuación detallaremos la minimización de rizado usando el método de la pseudo - inversa. Describiremos con detalle el análisis matricial para obtener S, T y W para posteriormente con la secuencia que siguen obtener los códigos correspondientes, los mismos que se encuentran en el anexo “Código Fuente”

Empleando el software *Mathemática* obtenemos las multiplicaciones e inversas.

Primero definimos lo siguiente $W^T W = I$ de esta manera simplificamos el análisis. Por lo que la ecuación (15) queda:

$$\hat{f} = \left(I - T\left[T^T T\right]^{-1} T^T\right) S g. \qquad \textbf{(16)}$$

De la ecuación (11) obtenemos la matriz R indicada en la figura 2.3 evaluada con n=5 . La figura 2.4 nos muestra la matriz R^{-1}.

$$R=\begin{pmatrix}
1/n & 1/n & 1/n & 1/n & 1/n & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1/n & 1/n & 1/n & 1/n & 1/n & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1/n & 1/n & 1/n & 1/n & 1/n & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1/n & 1/n & 1/n & 1/n & 1/n & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1/n & 1/n & 1/n & 1/n & 1/n & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1/n & 1/n & 1/n & 1/n & 1/n & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1/n & 1/n & 1/n & 1/n & 1/n & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/n & 1/n & 1/n & 1/n & 1/n & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/n & 1/n & 1/n & 1/n & 1/n & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/n & 1/n & 1/n & 1/n & 1/n & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/n & 1/n & 1/n & 1/n & 1/n & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/n & 1/n & 1/n & 1/n & 1/n & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/n & 1/n & 1/n & 1/n & 1/n & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/n & 1/n & 1/n & 1/n & 1/n & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/n & 1/n & 1/n & 1/n & 1/n & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/n & 1/n & 1/n & 1/n & 1/n \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{pmatrix}$$

Figura 2.3: Matriz R

$$R^{-1}=\begin{pmatrix}
n & -n & 0 & 0 & 0 & n & -n & 0 & 0 & 0 & n & -n & 0 & 0 & 0 & n & -1 & -1 & -1 & -1 \\
0 & n & -n & 0 & 0 & 0 & n & -n & 0 & 0 & 0 & n & -n & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & n & -n & 0 & 0 & 0 & n & -n & 0 & 0 & 0 & n & -n & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & n & -n & 0 & 0 & 0 & n & -n & 0 & 0 & 0 & n & -n & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & n & -n & 0 & 0 & 0 & n & -n & 0 & 0 & 0 & n & -n & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & n & -n & 0 & 0 & 0 & n & -n & 0 & 0 & 0 & n & -1 & -1 & -1 & -1 \\
0 & 0 & 0 & 0 & 0 & 0 & n & -n & 0 & 0 & 0 & n & -n & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & n & -n & 0 & 0 & 0 & n & -n & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & n & -n & 0 & 0 & 0 & n & -n & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & n & -n & 0 & 0 & 0 & n & -n & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & n & -n & 0 & 0 & 0 & n & -1 & -1 & -1 & -1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & n & -n & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & n & -n & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & n & -n & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & n & -n & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & n & -1 & -1 & -1 & -1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{pmatrix}$$

Figura 2.4: Matriz Inversa de R

Las primeras dieciséis columnas pertenecen a la matriz S y las tres últimas columnas a la matriz T.

De la ecuación (16) vamos a obtener [T^T T] como se observa en la figura 2.5. Mientras que la figura 2.6 representa la siguiente operación [T^T T]$^{-1}$:

$$T^T.T = \left(\left[\begin{pmatrix} -1 & -1 & -1 & -1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -1 & -1 & -1 & -1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -1 & -1 & -1 & -1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -1 & -1 & -1 & -1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}\right] \cdot \begin{pmatrix} -1 & -1 & -1 & -1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -1 & -1 & -1 & -1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -1 & -1 & -1 & -1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -1 & -1 & -1 & -1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}\right)$$

Figura 2.5: Matriz [$T^T.T$]

$$\left(T^T.T\right)^{-1} = \begin{pmatrix} \frac{1}{5} & -\frac{1}{20} & -\frac{1}{20} & -\frac{1}{20} \\ -\frac{1}{20} & \frac{1}{5} & -\frac{1}{20} & -\frac{1}{20} \\ -\frac{1}{20} & -\frac{1}{20} & \frac{1}{5} & -\frac{1}{20} \\ -\frac{1}{20} & -\frac{1}{20} & -\frac{1}{20} & \frac{1}{5} \end{pmatrix}$$

Figura 2.6: Matriz $\left(T^T.T\right)^{-1}$

Para posteriormente realizar la siguiente multiplicación T $[T^T T]^{-1} T^T$ la cual está representada en la figura 2.7:

$$20\begin{pmatrix} -1 & -1 & -1 & -1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -1 & -1 & -1 & -1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -1 & -1 & -1 & -1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -1 & -1 & -1 & -1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \frac{1}{5} & -\frac{1}{20} & -\frac{1}{20} & -\frac{1}{20} \\ -\frac{1}{20} & \frac{1}{5} & -\frac{1}{20} & -\frac{1}{20} \\ -\frac{1}{20} & -\frac{1}{20} & \frac{1}{5} & -\frac{1}{20} \\ -\frac{1}{20} & -\frac{1}{20} & -\frac{1}{20} & \frac{1}{5} \end{pmatrix} ..Transpose \begin{bmatrix} -1 & -1 & -1 & -1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -1 & -1 & -1 & -1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -1 & -1 & -1 & -1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -1 & -1 & -1 & -1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figura 2.7: Matriz T $\left(T^T.T\right)^{-1}$ T^T

Luego hacemos I - T $[T^T T]^{-1}$ T^T que en el código se llama I_menos_resto , esta operación se muestra a continuación en la figura 2.8 y su resultado en la figura 2.9.

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} - \begin{pmatrix} \frac{1}{5} & -\frac{1}{20} & -\frac{1}{20} & -\frac{1}{20} \\ -\frac{1}{20} & \frac{1}{5} & -\frac{1}{20} & -\frac{1}{20} \\ -\frac{1}{20} & -\frac{1}{20} & \frac{1}{5} & -\frac{1}{20} \\ -\frac{1}{20} & -\frac{1}{20} & -\frac{1}{20} & \frac{1}{5} \end{pmatrix}$$

Figura 2.8: Matriz I - T $\left(T^T.T\right)^{-1}$ T^{T}

$$\begin{pmatrix} \frac{4}{5} & \frac{1}{20} & \frac{1}{20} & \frac{1}{20} \\ \frac{1}{20} & \frac{4}{5} & \frac{1}{20} & \frac{1}{20} \\ \frac{1}{20} & \frac{1}{20} & \frac{4}{5} & \frac{1}{20} \\ \frac{1}{20} & \frac{1}{20} & \frac{1}{20} & \frac{4}{5} \end{pmatrix}$$

Figura 2.9: Resultado de la Matriz I - T $\left(T^T.T\right)^{-1}$ T^{T}

Ahora lo que nos queda es multiplicarlo con el resultado de S.g.

En esta parte no tenemos la necesidad de utilizar *Mathematica*, la multiplicación la realizamos directamente en *processing*. La matriz I_menos_resto lo multiplico con S y lo guardamos en IRporS y multiplicamos con cada componente de la matriz g. Esto lo puede revisar en ANEXOS.

CAPÌTULO 3

3 ANALISIS DE RESULTADOS

3.1 Resultado de imagen degradada

A continuación veamos los resultados de cómo se observa la degradación de una imagen en blanco y negro y otra con color para algunos valores de parámetro de degradación (n).

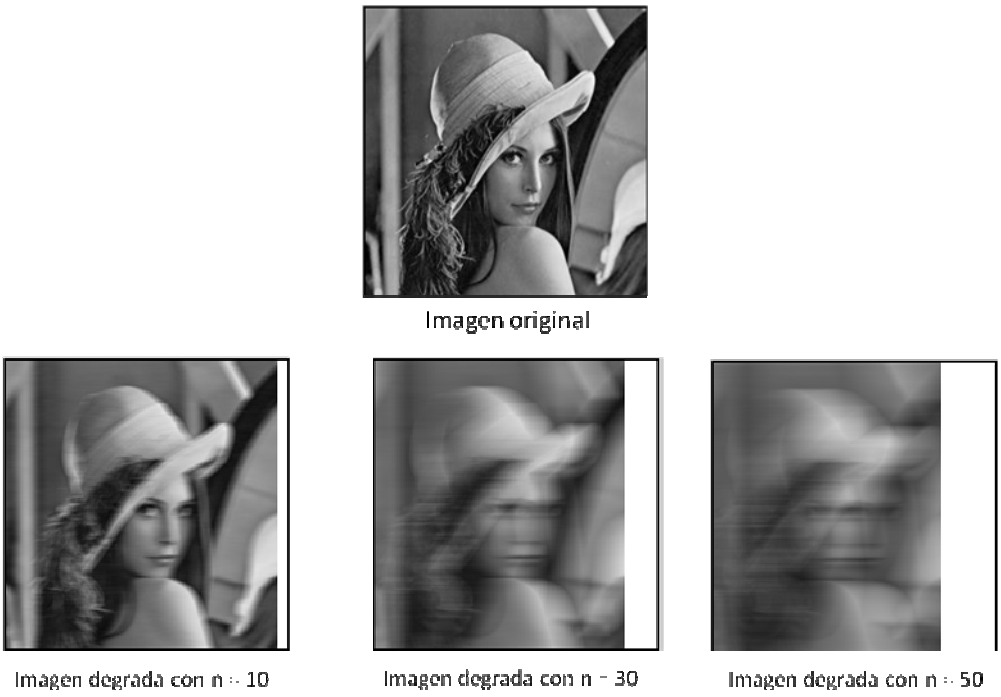


Figura 3.1 Degradación Horizontal en imágenes Blanco y Negro

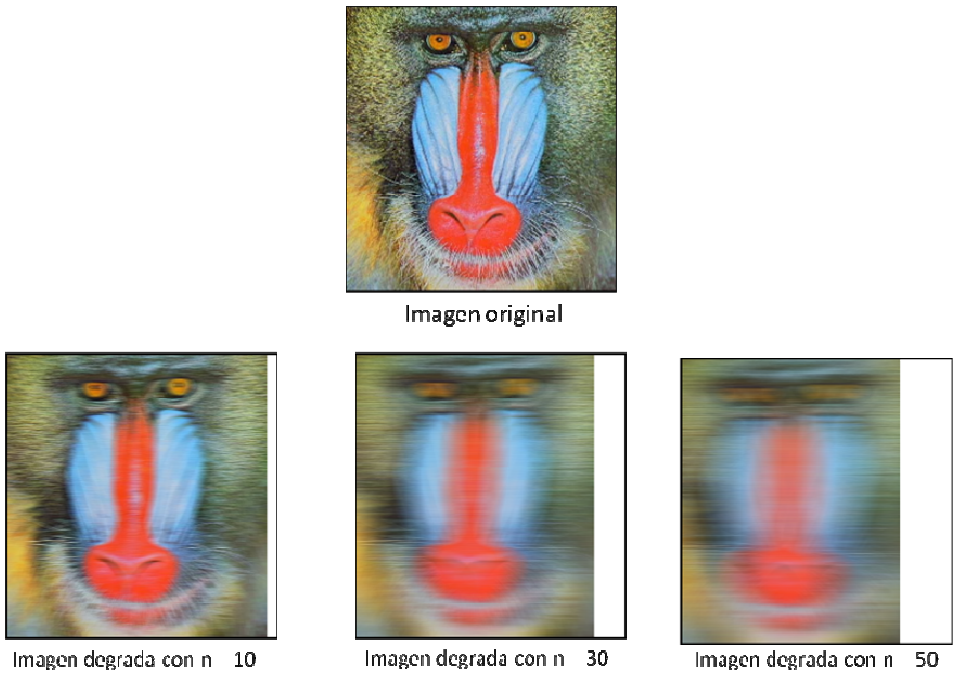


Figura 3.2 Degradación Horizontal en imágenes a Color

3.2 Resultado de imagen restaurada

A continuación veamos los resultados de cómo se observa la restauración de una imagen en blanco y negro y otra con color para los mismos valores de degradación anteriores.

Figura 3.3 Imágenes en blanco y negro restauradas

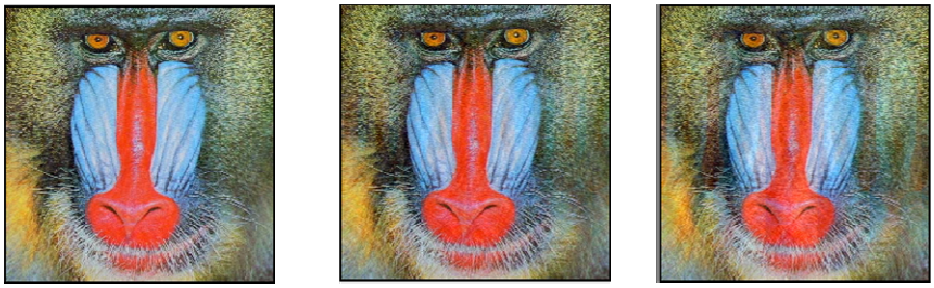


Figura 3.4 Imágenes a color restauradas

Si degradamos una imagen con un valor de n en particular debemos restaurarla con el mismo valor de n para que la restauración de la imagen salga lo mejor posible. No va a quedar idéntica a la imagen original ya que siempre se perderá información cuando se la degrada. Observemos la imagen restaurada cuando la degradación y restauración tienen diferentes valores de n.

Figura 3.5 Restauración de Imágenes
a) Imagen original, b) imagen restaurada con n = 30
c) imagen restaurada con n = 40

3.3 Desempeño de la restauración.

En esta Sección se presentan los resultados obtenidos de la restauración de una manera analítica realizando un grafico de "Relación Señal a Ruido Mejorada" (ISNR) en función de n. Se emplea una imagen original y un proceso de degradación conocido.

La imagen original será de L x M elementos y se almacenará en la matriz F. La imagen degradada tendrá L x N elementos y se almacenaría en la matriz G. La restauración de F se denominará Fˆ y será del mismo tamaño que F.

Para evaluar el desempeño de la restauración se utilizará el valor *ISNR* definido como:

$$ISNR = \left(\frac{\left\| F_N - G \right\|}{\left\| F - \hat{F} \right\|} \right) = \left(\frac{\sum_{i=1}^{L} \sum_{j=1}^{N} \left(F_{ij} - G_{ij} \right)^2}{\sum_{i=1}^{L} \sum_{j=1}^{M} \left(F_{ij} - \hat{F}_{ij} \right)^2} \right) \quad \textbf{(17)}$$

Por lo general el valor de ISNR se lo expresa en decibeles no lo hacemos de esta manera porque estos valores son pequeños y negativos. Igual de la forma expuesta en este documento se puede analizar el desempeño del método de la restauración.

Es necesario utilizar F_N, definido como las primeras N columnas de F, para poder compararlo con G que tiene sólo N columnas. Se observa que el ISNR disminuye a medida que n aumenta, y tiende a 1 cuando F y Fˆ son iguales (no se degrada nada). Este criterio sólo puede ser aplicado a problemas de restauración en los que la imagen degradada ha sido simulada, ya que es necesario conocer la imagen original F.

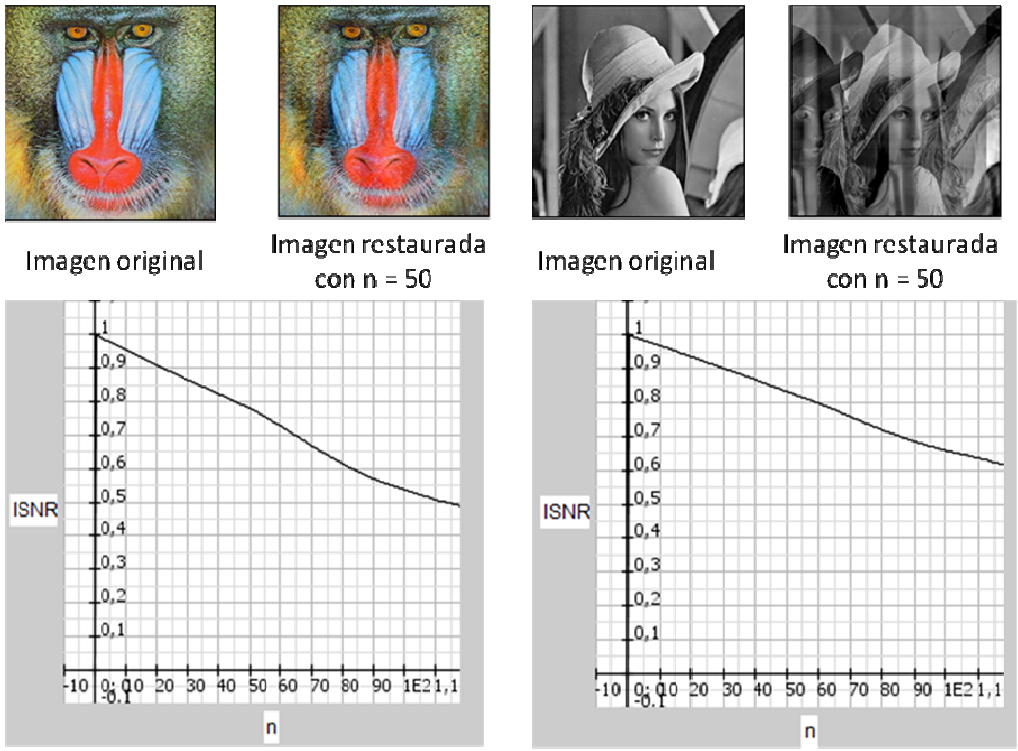


Figura 3.6 Desempeño de la Restauración

A continuación se muestran las tablas del valor del desempeño ISNR en función del parámetro de degradación n, para las imágenes que se están analizando.

Valor de degradación (n)	Desempeño (ISNR)
10	0,97
20	0,93
30	0,9
40	0,87
50	0,83
60	0,8
70	0,76
80	0,72
90	0,68
100	0,66

Tabla 3.1: Valores del ISNR para una Imagen en blanco y negro.

Columna1	Columna2
Valor de degradaciòn (n)	Desempeño (ISNR)
10	0,96
20	0,91
30	0,87
40	0,83
50	0,78
60	0,73
70	0,67
80	0,63
90	0,58
100	0,54

Tabla 3.2: Valores del ISNR para una Imagen a color.

CONCLUSION

Si se realizan todas las operaciones como aparecen en las fórmulas, el costo computacional es muy alto pues se deben realizar muchas multiplicaciones lo que implica hacer tres “lazos for” anidados lo cual consume recursos.

En lugar de eso utilizamos el software *matemática* que nos sirvió para poder programar en base a patrones que se observaban en las distintas pruebas con diferente tamaño de matrices que realizamos. Además con esto reducimos las líneas de código.

La restauración para imágenes en blanco y negro es mejor que para imágenes a color lo cual se indica en el valor de ISNR que es menor para imágenes a colores. Esto es debido a que en imágenes a color tiene que restaurar cada componente de color de la imagen.

La imagen restaurada presentaba unas líneas blancas lo que significaba que en esas filas no se mostraba ningún resultado, estas líneas aparecían cuando *n* es múltiplo de M que es el tamaño de la imagen.

Esto complicaba los valores de ISNR para el desempeño del proyecto. Lo que se hizo para mejorar el ISNR es llenar esas filas con la fila anterior que a simple vista no se nota ninguna diferencia en la imagen restaurada.

La mejor recuperación de la imagen se obtiene poniendo el mismo valor parámetro de degradación *n* tanto para la parte de degradación como la de restauración.

BIBLIOGRAFIA

[1] MERY Domingo ; LÓPEZ Marcela : “Restauración de imágenes usando el criterio de minimización de rizado en la imagen restaurada” Universidad de Santiago de Chile. Departamento de Ingeniería Informática

[2] Tutorial de JAVA (Sun) http://java.sun.com/docs/books/tutorial/

[3] H. Andrews and B.Hunt. “Digital Restoration”. Pretince - Hall

[4] Pita Ruiz. “Algebra Lineal”. Edit Mc Graw Hill. C.

[5] Richard Baraniuk: “Propiedades de los sistemas” www.wikilearning.com/tutorial

[6] http://wikipedia.org/wiki/Multiplicadores_de_Lagrange

[7] Oppenheim, Willsky. Sistemas y Señales, Prentice Hall

[8] Josefa Marín Molina: “Análisis Matemático” Editorial: Universidad Politécnica de Valencia

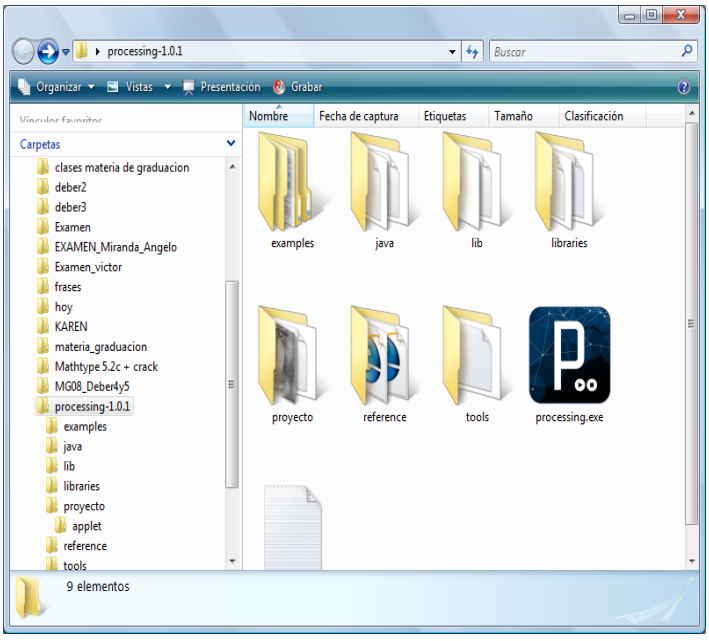
ANEXO

ANEXO A: Manejo del software

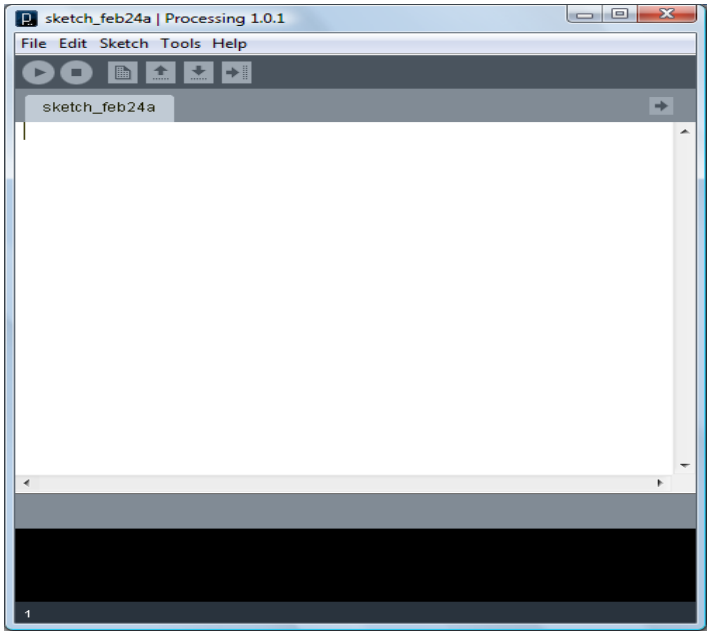
A continuación describiremos como manejar el software.

Primero tiene que instalar *JAVA 1.6.1.0, processing* 1.0 no se instala solo se lo ejecuta directamente y *Mathematica 5.2* no se necesita solo lo utilizamos para simplificar el análisis matemático en la parte de la restauración.

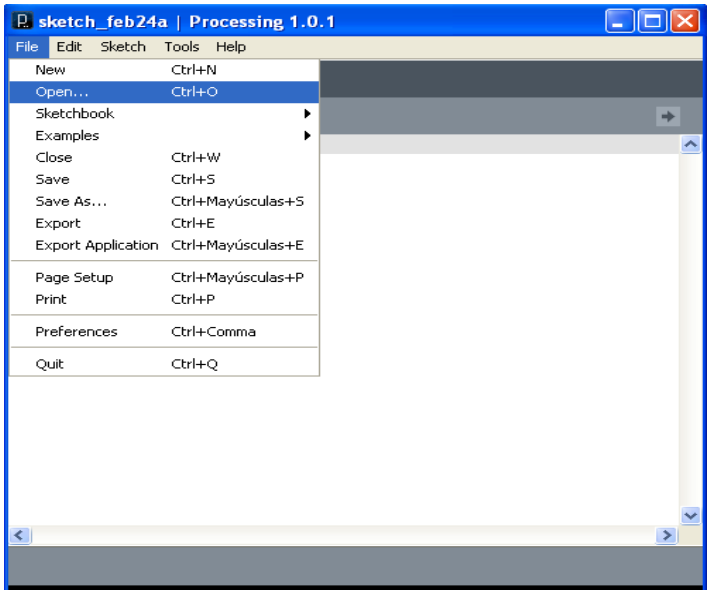
La carpeta llamada procesing 1.0.1 tiene que guardarla en alguna parte de su máquina, el cual contiene processing.exe que es el ejecutable y la carpeta proyecto es donde está nuestro código.



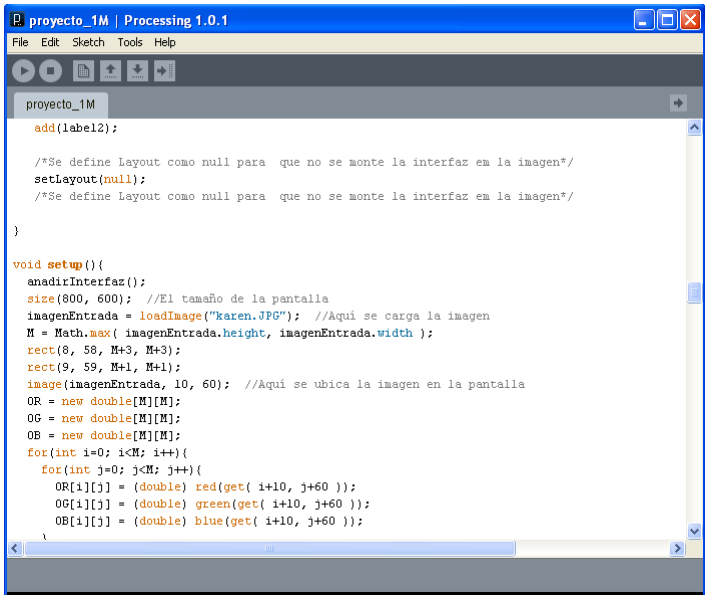
Hacer doble clic en processing.exe, la pantalla que se muestra es la siguiente:



Hacer clic en File, luego en open y buscar la carpeta processing 1.0.1 el cual contiene otra carpeta llamada proyecto que tiene los archivos proyecto.pde y la imagen con que vamos a trabajar.

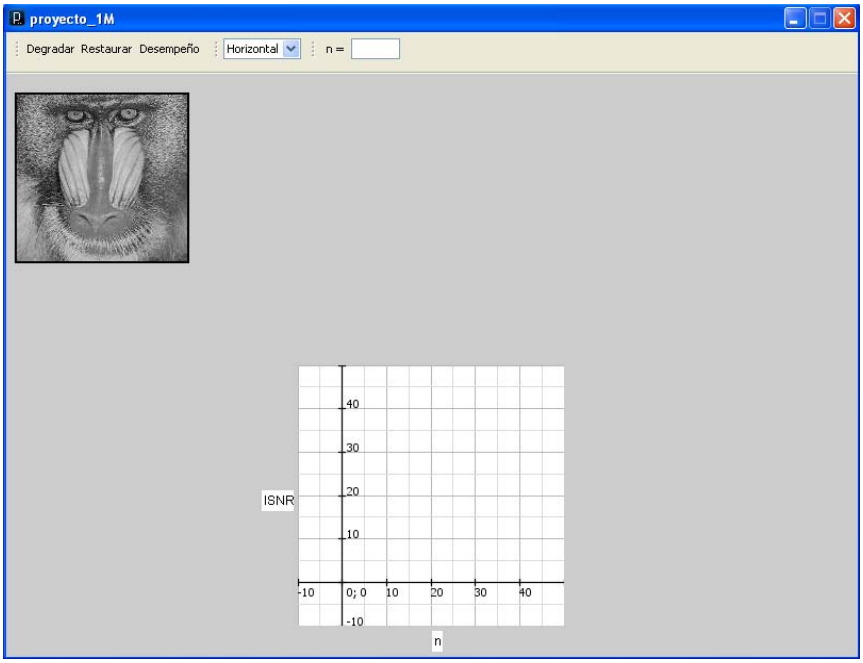


Abrir el archivo proyecto.pde, luego de eso aparece esta pantalla que muestra el código. La pantalla anterior la podemos borrar.



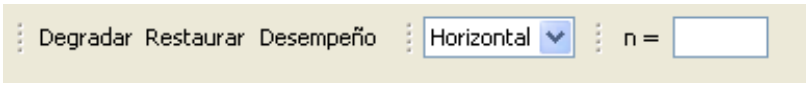
Hacemos clic en RUN que se representa por medio de este símbolo: 

La pantalla que aparece es la interfaz entre el usuario y la pantalla

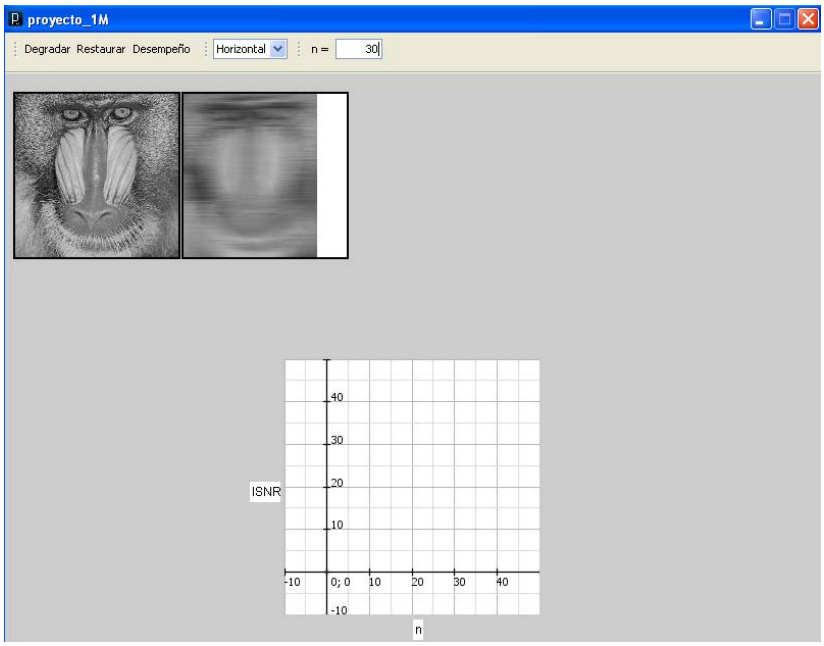


La imagen que se muestra es la original que luego vamos a degradar y posteriormente restaurarla. La cuadricula que se encuentra en la parte baja al final nos muestra el desempeño de la restauración.

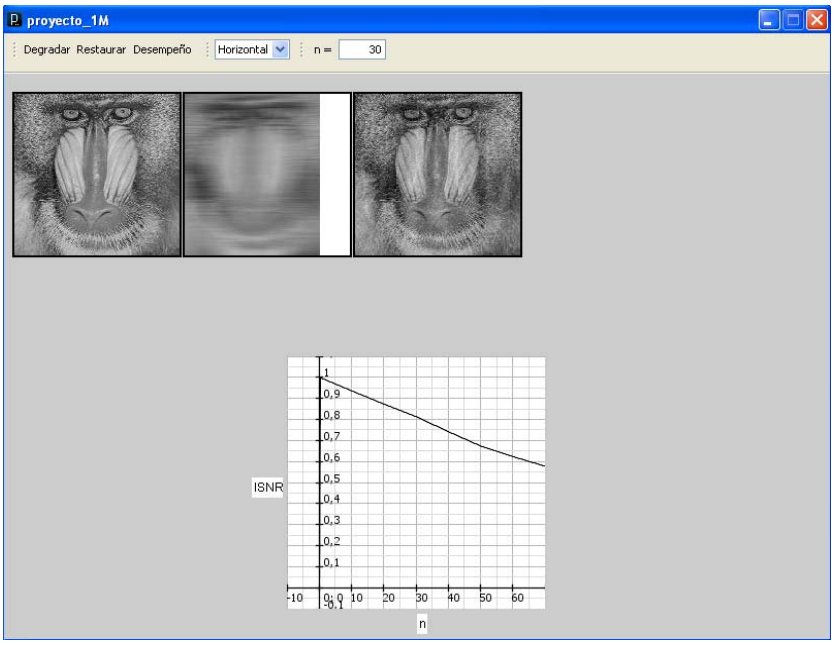
La barra de menú muestra los botones: Degradar, Restaurar, Desempeño. Tenemos una opción que sirve para degradarla mediante un movimiento horizontal o vertical. Al final se encuentra la opción para ingresar el parámetro de degradación.



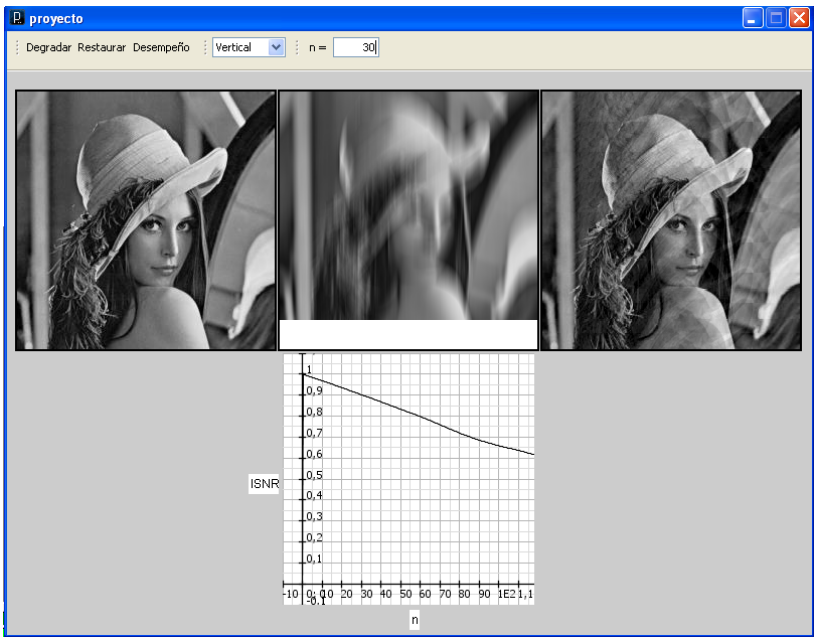
Como ejemplo tomemos el valor de n = 30, valores negativos no están definidos y tampoco valores de n que sean mayores al tamaño de la imagen a degradar. Luego elegimos la opción horizontal y hacer clic en la opción Degradar. El cual muestra la imagen degradada.



Luego hacemos clic en restaurar y finalmente en Desempeño.



Esta imagen es de tamaño 160x160. Podemos trabajar con una imagen de máximo 256x256 por el tamaño de la pantalla. La cual quedará de esta forma luego de hacer los pasos descritos anteriormente.



NOTA: Si queremos trabajar con otra imagen la debemos agregar a la carpeta proyecto y ponerle el nombre imagen.JPG. La extensión debe estar en mayúsculas ya que JAVA si distingue si ponemos la extensión con minúscula y no se abrirá el programa

ANEXO B: Código Fuente

ANEXO B.1: Código para cargar y ubicar la imagen en la interfaz

```
  void setup(){
  anadirInterfaz();
  size(800, 600);  //El tamaño de la pantalla
  imagenEntrada = loadImage("imagen.JPG");  //Aquí se carga la imagen
  M = Math.max( imagenEntrada.height, imagenEntrada.width ); //Se define el valor de M
  rect(8, 58, M+3, M+3); //Es el rectangulo de la imagen inicial
  rect(9, 59, M+1, M+1); //Es el rectangulo de la imagen inicial
  image(imagenEntrada, 10, 60);  //Aquí se ubica la imagen en la pantalla
  OR = new double[M][M]; //Arreglo para guardar el rojo de la imagen original
  OG = new double[M][M]; //Arreglo para guardar el verde de la imagen original
  OB = new double[M][M]; //Arreglo para guardar el azul de la imagen original
//Este for recorre la parte de la pantalla donde esta la imagen original
  //Se guardan las compontes del color de la imagen original en los arreglos OR OG y OB
  for(int i=0; i<M; i++){
    for(int j=0; j<M; j++){
      OR[i][j] = (double) red(get( i+10, j+60 ));
      OG[i][j] = (double) green(get( i+10, j+60 ));
      OB[i][j] = (double) blue(get( i+10, j+60 ));

    }
  }
}
```

ANEXO B.2: Código para degradar la imagen Recibe como entradas la imagen, el parámetro de degradación n y el tipo de orientación.

```
void degradarImagen(PImage imagenEntrada, int n, int orientacion ){
  rect(M+13, 58, M+3, M+3); //El rectangulo de la imagen degradada
  rect(M+14, 59, M+1, M+1);
  N = M - n +1; //El número de filas de la imagen degradada
  float colorDegR, colorDegG, colorDegB; //Aqui se almacenaran las componentes de color
de la imagen degradada
  //Las condiciones para no ingresar valores de n incorrectos
  if( n <= 1 ){
    System.out.println("En valor de n debe ser mayora a dos");  //Mensaje para valores de n
menores o iguales a cero
  }
  else if( n > imagenEntrada.width ){
    System.out.println("En valor de n menor al ancho de la imagen");  //Mensaje para valores
de n mayores al ancho
  }
```

```
  else{
    GR = new double[N][M]; //Los arreglos para almacenar las componentes de color de la
imagen degradada
    GG = new double[N][M];
    GB = new double[N][M];
    for(int j=0; j<M; j++)//El  primer for recorre fila por fila la imagen original. M es el
número de filas
      for(int i=0; i<N; i++){ //Para realizar N multiplicaiones, cada una correspondiente a una
fila de la imagen degradada
        colorDegR=0;
        colorDegG=0;
        colorDegB=0;
        //La convolucion para degradar la imagen
        if( orientacion == HORIZONTAL ){
          for(int k=i; k<n+i; k++){
            colorDegR = colorDegR + (1.0/n)*( red(get( k+10, j+60  )) );
            colorDegG = colorDegG + (1.0/n)*( green(get( k+10, j+60  )) );
            colorDegB = colorDegB + (1.0/n)*( blue(get( k+10, j+60  )) );
          }
        }
        else if( orientacion == VERTICAL ){
          for(int k=i; k<n+i; k++){
            colorDegR = colorDegR + (1.0/n)*( red(get( j+10, k+60 )) );
            colorDegG = colorDegG + (1.0/n)*( green(get( j+10, k+60 )) );
            colorDegB = colorDegB + (1.0/n)*( blue(get( j+10, k+60  )) );
          }

}
        //Se ubica en pantalla la imagen degradada cuyar componentes están almacenada el
        colorDegR/G/B
        if( orientacion == HORIZONTAL ){
          set( i+M+15, j+60, color((int)colorDegR, (int)colorDegG, (int)colorDegB));
        }
        else{
          set( j+M+15, i+60, color((int)colorDegR, (int)colorDegG, (int)colorDegB));
        }
        //Se almacenan las componentes de color de la imagen degradada en los arreglos
        GR[i][j] =   colorDegR;
        GG[i][j] =   colorDegG;
        GB[i][j] =   colorDegB;
      }
    }
}
/* Este método toma como argumento cualquier matriz e inicializa todos sus elementos a
cero*/
public void inicializarMatriz(double[][] matriz){
  for(int i=0; i < matriz.length; i++){
    for(int j=0; j< matriz[0].length; j++){
```

```
      matriz[i][j]=0;
    }
  }
}
/* Este método toma como argumento cualquier matriz e inicializa todos sus elementos a
cero*/
public void inicializarMatriz(double[][] matriz, double valor){
  for(int i=0; i < matriz.length; i++){
    for(int j=0; j< matriz[0].length; j++){
      matriz[i][j]=valor;

    }
  }
}
```

ANEXO B.3: Código para restaurar la imagen Recibe como entradas la imagen degradada, el parámetro de degradación n y el tipo de orientación.

```
void restaurarImagen(PImage imagenDegradada, int n, int orientacion ){
  rect(2*M+18, 58, M+3, M+3); //El rectangulo para la imagen restaurada
  rect(2*M+19, 59, M+1, M+1);
  M = Math.max( imagenEntrada.height, imagenEntrada.width );
  N = M -n + 1; //El número de filas de la imagen degradada
  double S[][] = new double[M][N]; //El arreglo para la matriz
  double T[][] = new double[M][n-1];//El arreglo para la matriz T
  double Tt_por_T_inv[][] = new double[n-1][n-1];//El arreglo para T traspuesta por T
inversa
  double I_menos_resto[][] = new double[M][M];  //El arreglo para I - resto de formula dentro
de los parentesis
  RR = new double[M][M]; //Los arreglos de las componentes de color de la imagen
restaurada
  RG = new double[M][M];
  RB = new double[M][M];
  inicializarMatriz(I_menos_resto, (double)1/M);
  //For anidado para llenar la matriz S
  for(int i=0; i<N; i++){
  for(int j=i; j<N; j=j+n){
     S[i][j] =  n;
     if( j+1<N )
       S[i][j+1] =  -n;
    }
  }
  //For anidado que permite obener la matriz I - resto
  for(int i=0; i<M; i++){
```

```
  int k = i%n;
  for(int j=0; j<M; j=j+n){
   try{
    if( i==j+k )
     I_menos_resto[i][j+k] = 1 - (double) (n-1)/M;
    else
     I_menos_resto[i][j+k] =  - (double) (n-1)/M;
   }
   catch(Exception e){
   }
  }
 }
  IRporS = new double[M][N];
 //El producto entre I - resto y la matriz S. El resultado se almacena en IRporS
 producto(I_menos_resto, S, IRporS);
 //El producto entre IRporS y la matriz de la imagen degradada
 producto(IRporS, GR, RR);
 producto(IRporS, GG, RG);
 producto(IRporS, GB, RB);
 //Con este codigo se llenan las lineas blancas en caso de existir
 if(M%n!=0){
  for(int i=M; i>0; i--){
   for(int j=0; j<M; j++){
    if( (M-i)%n == 0 ) {
     try{
      RR[i][j] = RR[i+1][j];
      RG[i][j] = RG[i+1][j];
      RB[i][j] = RB[i+1][j];
     }
     catch(Exception e){}
    }
   }
  }
 }
 //Con este for se ubica en pantalla la imagen restaurada
 for(int i=0; i<M; i++){
  for(int j=0; j<M; j++){
   if(orientacion==HORIZONTAL){
set( i+2*M+20, j+60, color((int)RR[i][j], (int)RG[i][j], (int)RB[i][j]));
   }
   else{
    set( j+2*M+20, i+60, color((int)RR[i][j], (int)RG[i][j], (int)RB[i][j]));
   }
  }
 }
}

//Este metodo calcula el producto entre dos matrices. mat1 y mat y lo almacena en prod
```

```
void producto(double mat1[][], double mat2[][], double prod[] ]){
  int f1, f2, fp;
  int c1, c2, cp;
  f1 = mat1.length;
  c1 = mat1[0].length;
  f2 = mat2.length;
  c2 = mat2[0].length;
  fp = prod.length;
  cp = prod[0].length;
  for(int i=0; i<f1 ; i++)
    for(int j=0; j<c2 ; j++)
      for(int k=0; k<c1  ; k++)
        prod[i][j]=prod[i][j] + mat1[i][k]*mat2[k][j];
}
```

ANEXO B.4: Código para calcular el desempeño (ISNR)

```
public void calcularError(){
  double numeradorR = 0.0, numeradorG = 0.0, numeradorB = 0.0;
  double denominadorR = 0.0, denominadorG = 0.0, denominadorB = 0.0;
  double py[] = new double[100];
  double px[] = new double[100];
  py[0] = 1.0;
  px[0] = 0;
  for(int i=10; i<=M/2; i=i+10){
    px[i/10] = i;
    numeradorR = 0.0;
    denominadorR = 0.0;
    degradarImagen( imagenEntrada, i, HORIZONTAL);
    restaurarImagen( imagenEntrada, i, HORIZONTAL);
    for(int j=0; j<N; j++){
      for(int k=0; k<M; k++){
        numeradorR = numeradorR + Math.pow(OR[j][k] - Math.max(GR[j][k], 256 ), 2);
      }
    }
    for(int j=0; j<M; j++){
      for(int k=0; k<M; k++){
        denominadorR = denominadorR + Math.pow(OR[j][k] - Math.max(RR[j][k], 256), 2);
      }
    }
    py[i/10] = numeradorR/denominadorR;
  }
   Plot2 grafico = new Plot2();
   grafico.setValores(px, py);
   jXGraphError.addPlots(new Color(0, 0, 0), grafico);}
```