CENTRO DE INVESTIGACIONES EN OPTICA, A.C.

REPORTE FINAL DE RESIDENCIA PROFESIONAL

Enero – Junio 2009

ELABORADO POR

Juan Carlos Salinas Hernández

ESPECIALIDAD

Ingeniería Electrónica

PROYECTO

Sistema de Visión Aplicados a la Mecatrónica

Centro de Investigaciones en Óptica A.C.

Departamento de Metrología Óptica

Dr. Francisco Javier Cuevas de la Rosa

León Guanajuato, México

Tuxtla Gutiérrez, Chiapas Junio del 2009

AGRADECIMIENTOS

- A mi familia, quienes fueron parte fundamental para poder realizar mi estancia en el CIO.

- Al Departamento de Formación Académica, quienes me brindaron apoyo y amistad durante toda nuestra estancia.

- A los alumnos del CIO, quienes me brindaron su amistad y apoyo además de ayudarme a la comprensión de ciertos temas.

- Al Dr. Julio Cesar Estrada Rico y al Dr. Manuel de la Torre Ibarra por permitirme asistir a sus clases de Visón por computadora y Computación 2 respectivamente.

- Al personal de la Biblioteca y Comedor, quienes me dieron su apoyo y amistad.

- A mis compañeros y amigos del ITTG.

- A mi asesor interno en el ITTG, el Dr. Héctor Ricardo Hernández de León, quien me apoyo durante toda mi estancia en el CIO.

- Y principalmente al Dr. Francisco Javier Cuevas de la Rosa por haberme aceptado en su proyecto y haberme brindado su apoyo y su conocimiento para la realización de mi proyecto de residencia profesional.

INTRODUCCIÓN

A continuación se enumeran los antecedentes científicos en el área de investigación motivo del proyecto propuesto:

1.1 Desarrollo de técnicas para el procesamiento digital de imágenes conteniendo patrones de franjas y/o interferogramas.

1.1.1 Técnicas para recuperar fase a partir de patrones de franjas ruidosas.

En metrología óptica el uso de arreglos interferometricos y de proyección de luz estructurada han sido de gran utilidad para llevar a cabo la cuantificación de las cantidades físicas que están codificadas. El propósito de estas técnicas es primero obtener imágenes de patrones de franjas o interferogramas generadas de los arreglos experimentales y posteriormente tratarlas digitalmente mediante algoritmos para calcular y decodificar el término de fase, el cual está relacionado con la cantidad física que se está cuantificando.

Estas imágenes en la mayoría de las ocasiones vienen mal contrastadas además de tener una gran cantidad de ruido inmerso, lo cual complica el cálculo del término de fase. Lo recomendable para obtener una buena aproximación del término de fase es obtener diferentes imágenes variando los parámetros del arreglo experimental y digitalizando las imágenes resultantes.

Desgraciadamente debido a las condiciones mecánicas y de inestabilidad propias del mismo experimento a veces no es posible obtener varias imágenes relacionadas con la cantidad física de interés. El desenvolvimiento llega a complicarse cuando las diferencias absolutas de fase entre pixeles contiguos en otros puntos diferentes a lugares donde existen discontinuidades en la función arcotangente es mayor de pi.

Estas discontinuidades pueden deberse por ruido de alta frecuencia, de alta amplitud, saltos discontinuos de fase, y un submuestreo local en el interferograma o patrón de franjas.

Huntley y Goldstei consideraron el desenvolvimiento de fase mediante el aislamiento de discontinuidades erróneas antes de empezar con el proceso de desenvolvimiento. Ghiglia, Mastin y Romero utilizando las ideas de Fried y Hudgin de utilizar la integración de mínimos cuadrados para el problema de desenvolvimiento. Marroquín extendió la técnica de integración de mínimos cuadrados agregando un término de regularización para la solución en la forma de una norma de potenciales.

1.1.2 Algoritmos de optimización aplicados en procesado digital de imágenes y visión por computadora.

Las tareas de procesado digital de imágenes y visión por computadora tienen una diversidad de áreas de aplicación en distintas áreas tales como metrología óptica, inspección industrial, diagnóstico médico, reconocimiento óptico de caracteres y percepción remota, entre otras. Entonces el desarrollo de técnicas y algoritmos que resuelvan problemas de optimización en visión por computadora han sido punta de lanza en el área de metrología óptica. Entre los problemas a los que se hace referencia podemos mencionar entre otros, el trato de imágenes inmersas en ruido, optimización de materiales de corte a partir de imágenes digitales, el reconocimiento de patrones contenidos en imágenes, procesamiento de patrones de franjas y la medición de cantidades físicas a través de imágenes que contienen información de objetos bajo estudio.

Entre las técnicas y algoritmos se encuentran: técnicas de regularización, descenso de gradiente, redes neuronales, técnicas espaciales de mejoramiento de imágenes (PDI), técnicas frecuenciales para el mejoramiento de imágenes (interferogramas y patrones de franjas) y algoritmos genéticos.

JUSTIFICACIÓN

En años recientes el desarrollo de técnicas y algoritmos robustos y adaptivos de inteligencia artificial ha sido una de las líneas de investigación científicas de mayor relevancia para resolver problemas de complejidad alta en diferentes áreas de la ciencia. Debido a la complejidad de los problemas actuales en el área de procesado digital de imágenes la aplicación de estas técnicas ha sido un nicho científico importante para su aplicación.

ÍNDICE

DESARROLLO

Procesamiento Digital de Imágenes

El termino procesamiento digital de imágenes habla sobre la manipulación y análisis de imágenes por computadora.

El procesamiento de imagen puede considerarse como un tipo especial del procesamiento digital en dos dimensiones, el cual se usa para revelar información sobre imágenes y que involucra hardware, software y soporte teórico.

Definición de una imagen digital

El termino imagen se refiere a una función bidimensional de intensidad de luz *f*(*x,y*), donde *x* y *y* denotan las coordenadas espaciales y el valor de *f* en cualquier punto (*x,y*) es proporcional al brillo (o nivel de gris) de la imagen en ese punto.

Una imagen digital es una imagen *f*(*x, y*) que ha sido discretizada en coordenadas espaciales y en brillo. Una imagen digital puede considerarse como una matriz cuyos índices del renglón y columna identifican un punto en la imagen y el correspondiente valor del elemento de la matriz que identifica el nivel de intensidad de luz en ese punto.

Los elementos de tal arreglo digital son llamados *elementos de imagen*, *elementos de pintura*, *pixels* o *pels* (estos dos últimos son abreviaturas del ingles *picture elements*).

Elementos de los sistemas de procesamiento de imágenes

En la adquisición de imágenes deben existir dos elementos básicos. El primero es algún dispositivo básico que sea sensible a una determinada banda del espectro de energía electromagnético como son las bandas de rayos-x, el ultravioleta, el visible o el infrarrojo, y que produce una señal eléctrica proporcional al nivel de energía censado.

El segundo es el digitalizador que convierte la salida del dispositivo físico de censado a forma digital. En esta categoría se agrupan a las cámaras CCDs (Charge-Coupled Devices) que tiene la ventaja de la velocidad de capitulación (hasta 1/10,000 seg.) pero un costo elevado, los scanners y cámaras de video.

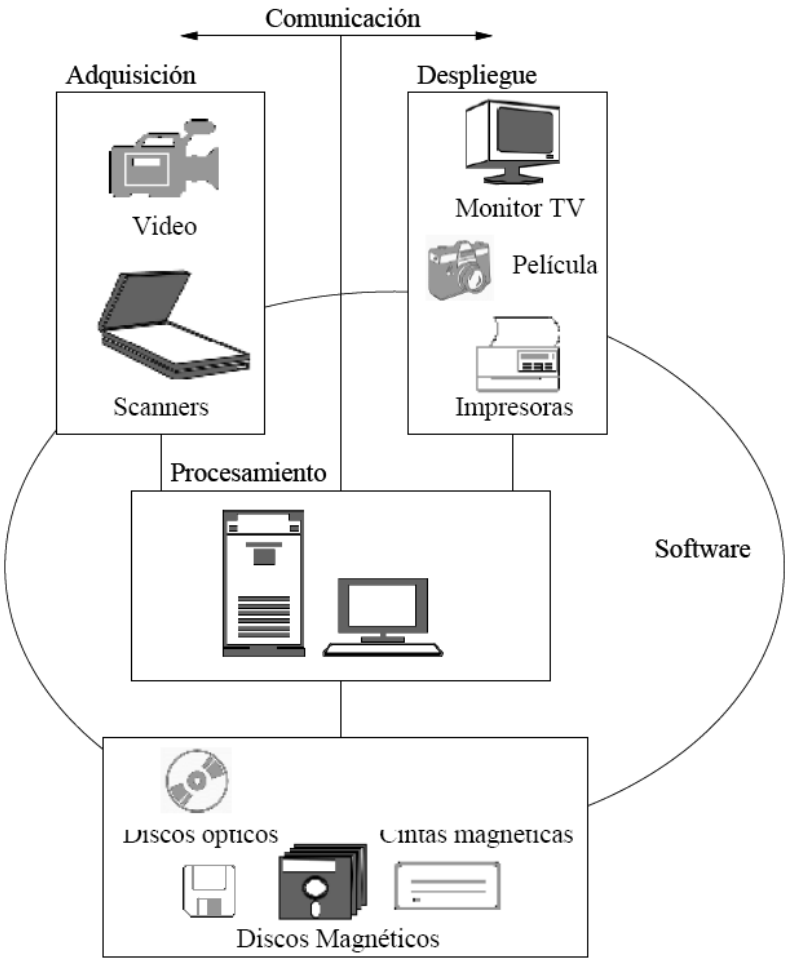
El almacenamiento es un punto crítico debido a la gran cantidad de información usada. Por ejemplo, una imagen en 8 bits de tamaño 1024 *x* 1024 pixeles requieren un megabyte de espacio para su almacenamiento.

En el procesamiento, ya existen computadoras con microprocesadores especializados en procesamiento de imágenes que permiten un manejo rápido de las operaciones de matrices y acceso a memoria para aplicaciones de procesamiento de marcos (frames).

En cuanto al despliegue de las imágenes, se han usado los monitores de T.V. y monitores de computadoras. Los resultados desplegados en el monitor pueden ser fotografiados por una cámara enfocada a la cara del tubo de rayos catódicos o generar directamente una señal de video para grabarse.

A continuación se muestran elementos funcionales básicos de un sistema de procesamiento de imágenes: adquisición, almacenamiento, procesado, comunicaciones, despliegue y software.

Dentro de cada caja se dan ejemplos de dispositivos usados en tales sistemas.



TRANSFORMACIÓN DE IMÁGENES

Por transformación de imágenes se entiende el proceso de modificar el contenido de una imagen original para obtener una nueva. El objetivo de cualquier transformación estriba en la necesidad de preparar la imagen con el fin de realizar un posterior análisis de cara a su interpretación. La interpretación entra dentro de un proceso de percepción de nivel superior que debe estar implícito en toda aplicación de visión artificial.

TRANSFORMACIONES BÁSICAS

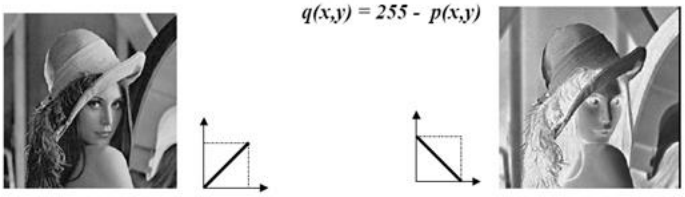
Operaciones Individuales

a) Operador Identidad

$$q(x,y) = p(x,y)$$

b) Operador Inverso o negativo

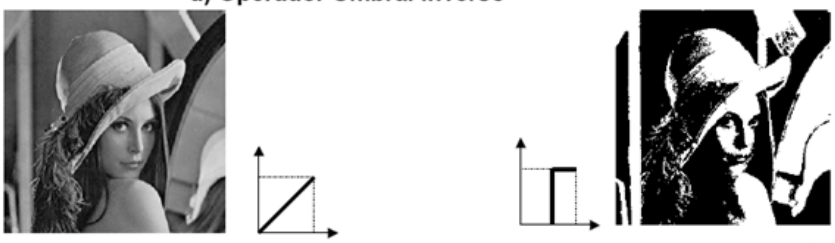
$$q(x,y) = 255 - p(x,y)$$



c) Operador Umbral

$$q(x,y) = 0 \text{ para } p(x,y) < u$$
$$q(x,y) = 255 \text{ para } p(x,y) > u$$

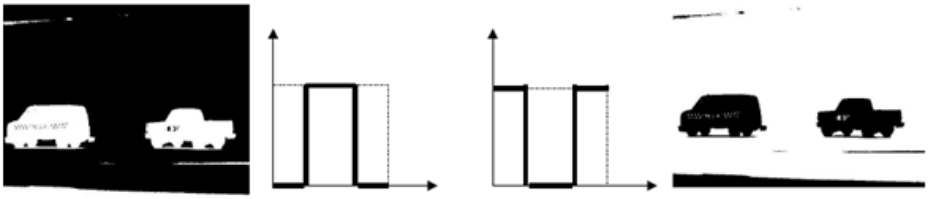
d) Operador Umbral Inverso



e) Operador Intervalo de Umbral binario

$q(x,y) = 0$ *para* $p(x,y) < u1$ *ó* $p(x,y) > u2$
$q(x,y) = 255$ *para* $u1 > p(x,y) < u2$

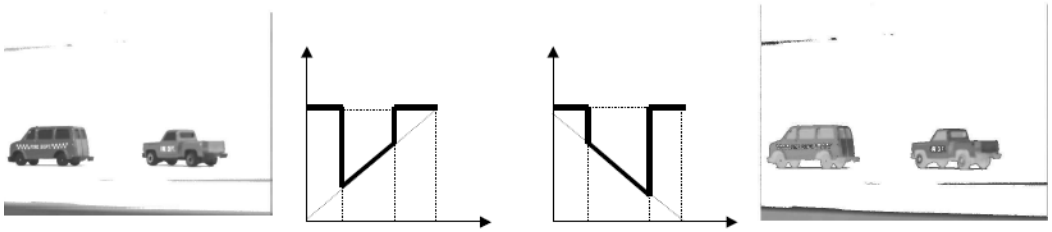
f) Operador Intervalo de Umbral binario inverso



h) Operador Intervalo de Umbral en Gris

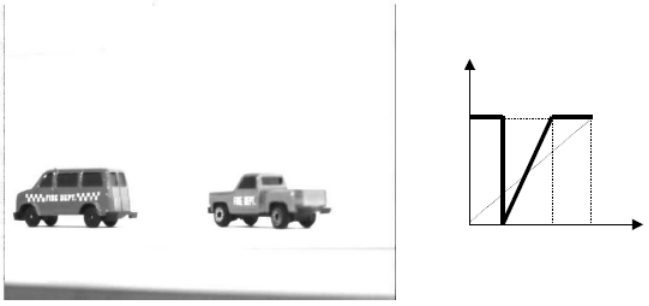
$q(x,y) = 255$ *para* $p(x,y) < u1$ *ó* $p(x,y) > u2$
$q(x,y) = p(x,y)$ *para* $u1 > p(x,y) < u2$

i) Operador Intervalo de Umbral en Gris inverso

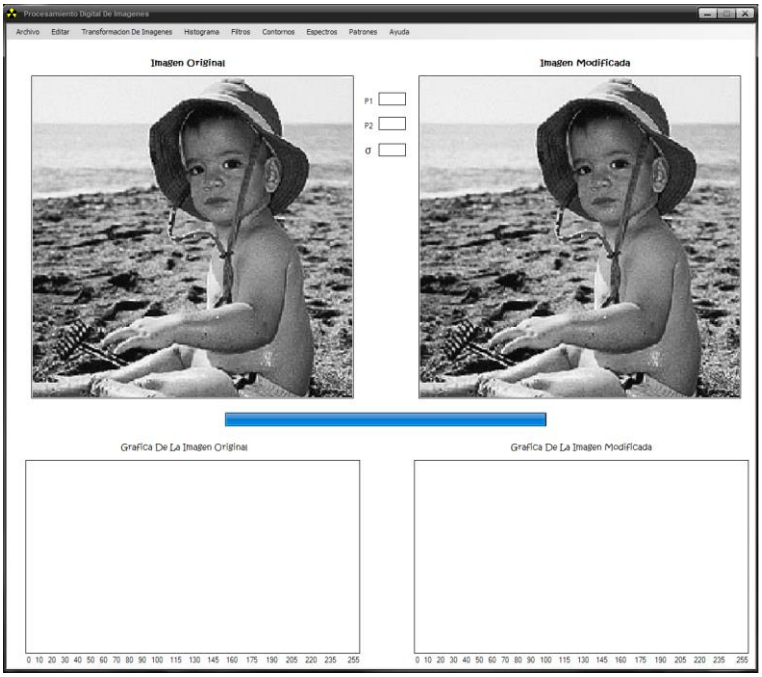


j) Operador de extensión

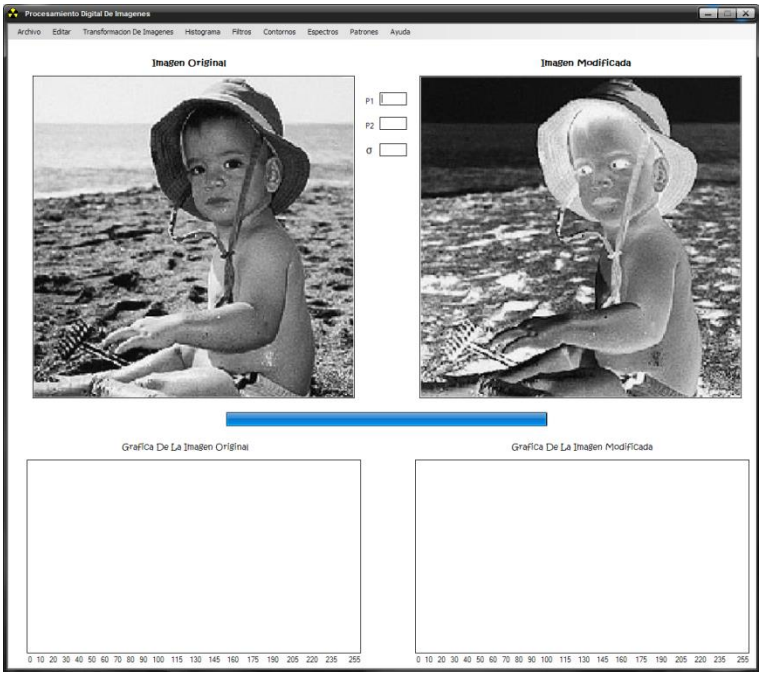
$q(x,y) = 255$ *para* $p(x,y) < u1$ *ó* $p(x,y) > u2$
$q(x,y) = 255*(p(x,y)-u1)/(u2-u1)$ *para* $u1 > p(x,y) < u2$



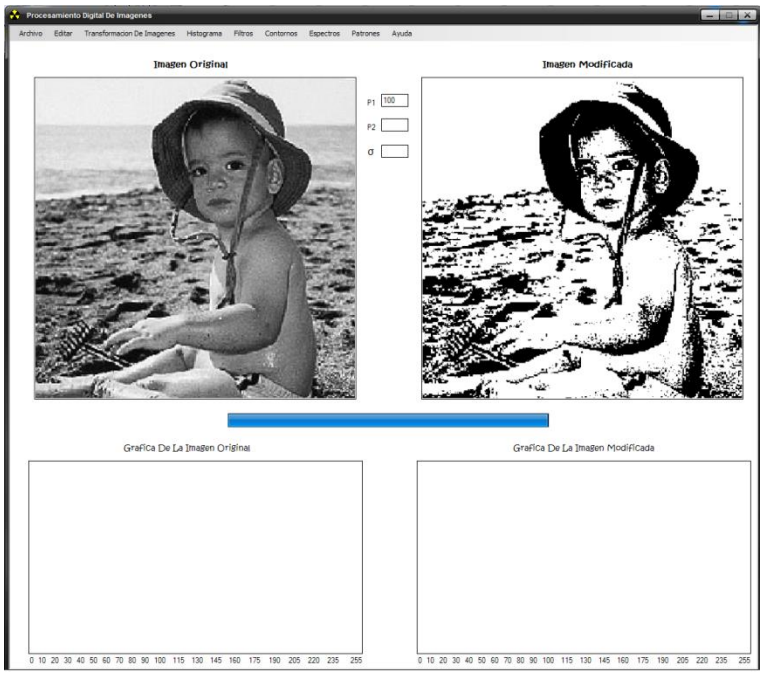
Imágenes obtenidas con software realizado



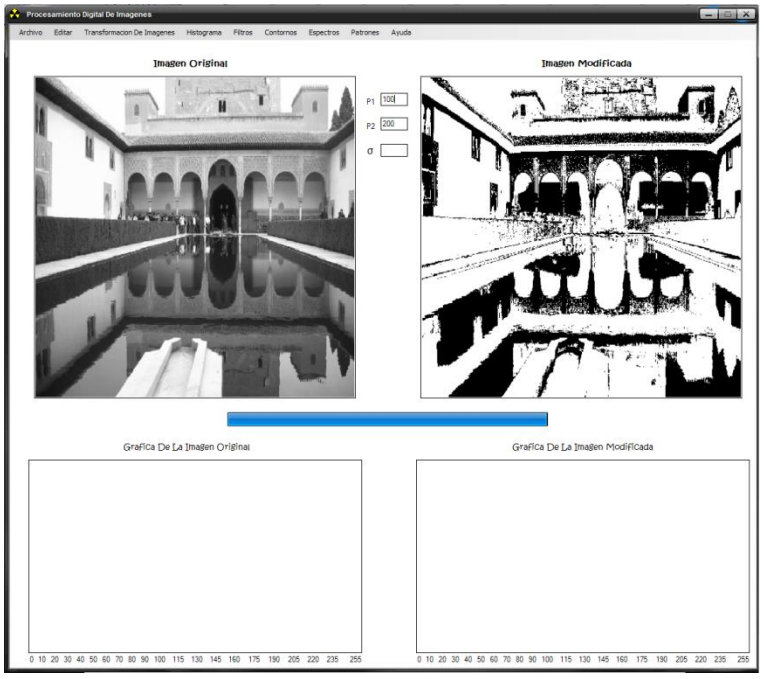
Operador Identidad



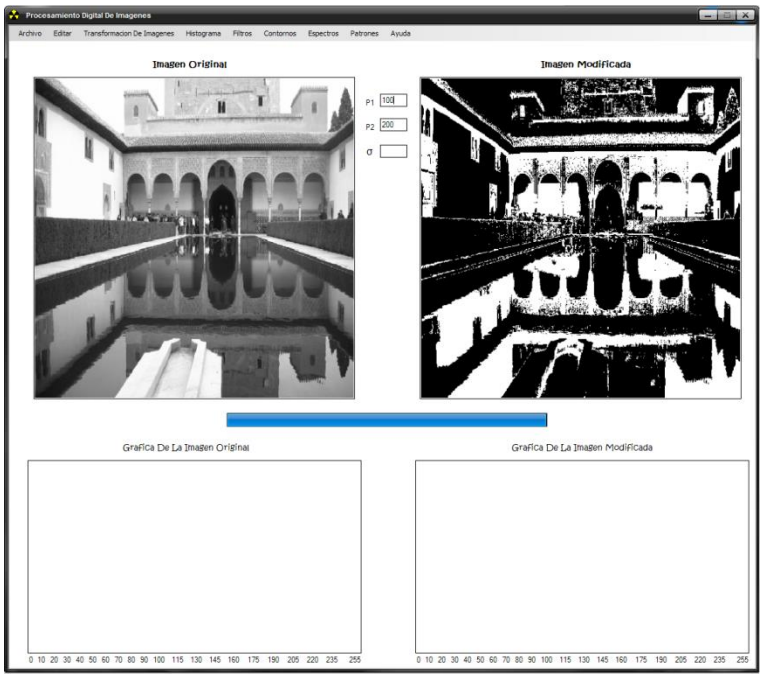
Operador Inverso o Negativo



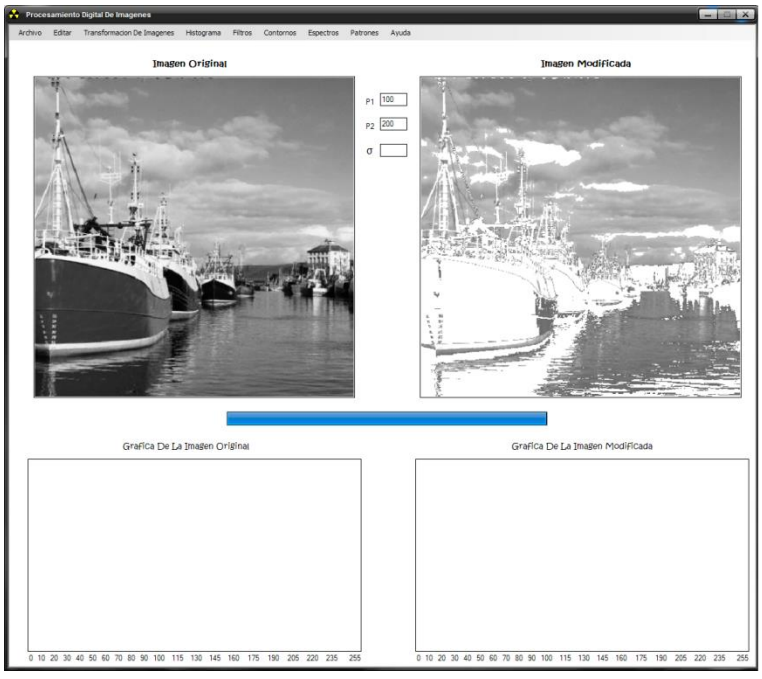
Operador Umbral



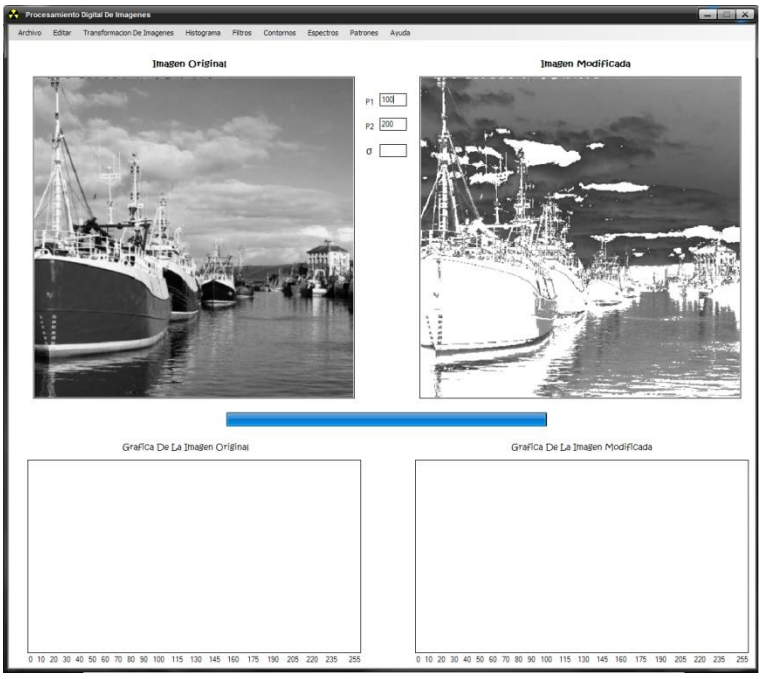
Operador Intervalo de Umbral Binario



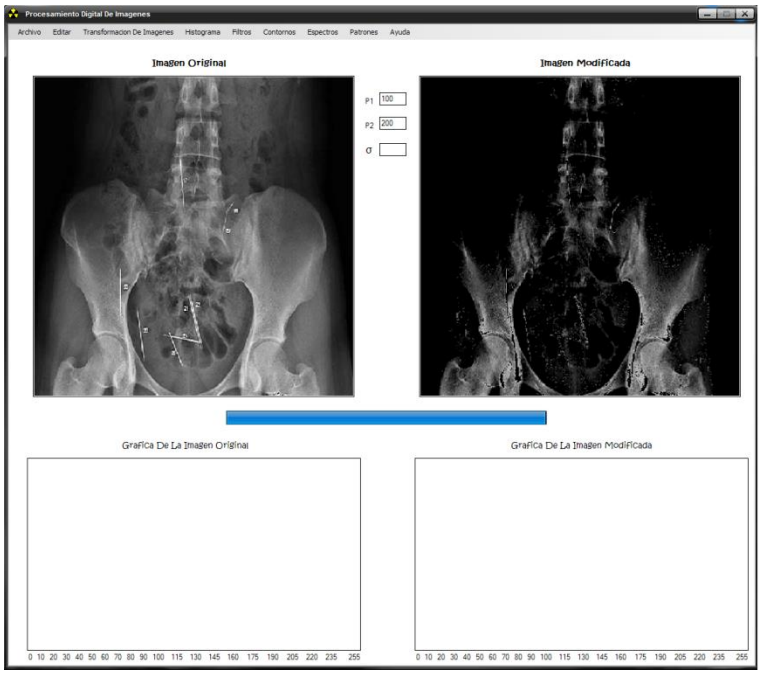
Operador Intervalo de Umbral Binario Inverso



Operador de Umbral de la Escala de Grises



Operador de Umbral de la Escala de Grises Invertido



Operador de Extensión

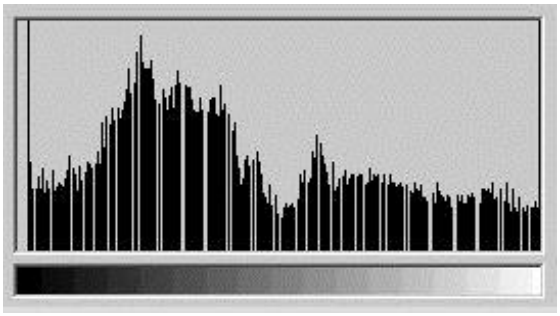
HISTOGRAMA DE UNA IMAGEN

El histograma de una imagen contiene la información de la probabilidad de aparición de las distintas tonalidades de color que se pueden dar en cada caso, ya que podemos trabajar en distintos tipos de colores o en escala de grises. En el caso de una imagen en color, no podemos hablar de un único histograma que caracterice a la imagen sino de tres histogramas, uno para cada color (RGB, por ejemplo). Si bien el uso del histograma y sus posteriores modificaciones son más aplicables a imágenes en escala de grises.

La creación de un histograma, en el caso de una imagen en escala de 256 tonalidades del blanco al negro (escala de grises), se realiza fácilmente mediante medios informáticos, de manera que, en primer lugar crearemos un vector (array) que contenga 256 posiciones, una por cada nivel de gris, el algoritmo recorrerá cada uno de los pixeles de la imagen, aumentando en una unidad el valor guardado en la posición del array correspondiente al tono del pixel en cuestión.

Ejemplo de histograma, imagen en escala de grises:

Imagen Original



Histograma de la imagen

El histograma proporciona una descripción de la apariencia global de una imagen. De forma que si los niveles de gris están concentrados hacia el extremo oscuro del rango de la escala de gris, la apariencia global de la imagen será oscura; mientras que si sucede justo lo contrario, la imagen correspondiente será brillante. Por su parte, un histograma que presente un perfil estrecho corresponderá a una imagen de bajo contraste y un histograma con una dispersión considerable a una imagen de alto contraste.

Procesado de histogramas

Como se comentó en el apartado anterior, el histograma de una imagen nos da una idea de la apariencia global de la misma, por lo que la modificación de éste provocará también una modificación de aquella. Este hecho da lugar a un conjunto de técnicas de procesado que básicamente podemos dividir en dos grupos: ecualización y especificación de histograma. Ambas técnicas se basan en la aplicación de una determinada función (T) a los niveles de gris de la imagen original (r) para obtener los niveles de gris de la imagen transformada (s):

$$s=T(r)$$

[Nota.- Trabajamos con niveles de gris normalizados, es decir, si la imagen en cuestión es una imagen en escala de grises de 8 bits (es decir, 256 niveles) utilizaremos los niveles: 0, 1/255, 2/255, ..., 1]

En cualquier caso, la función T(r) debe cumplir dos requisitos:

a) T(r) es de valor único y monótonamente creciente en el intervalo $0\leq r\leq 1$ (de forma que se preserve el orden entre blanco y negro de la escala de grises)

b) $0\leq T(r)\leq 1$ para $0\leq r\leq 1$ (de manera que se garantice que la aplicación es coherente con el rango de valores de pixel permitidos)

Dependiendo de nuestro objetivo (pero eso sí, siempre cumpliendo las dos premisas anteriores) podremos utilizar distintos tipos de aplicación.

También se comentó anteriormente que el histograma de una imagen da una idea de la probabilidad de que aparezca cada nivel de gris disponible en la misma, podríamos decir, por tanto, que correspondería a la función densidad de probabilidad de dichos niveles de gris. Si entendemos de esta forma el histograma y teniendo en cuenta el teorema de cambio de variable (teoría elemental de probabilidades), convendremos que la función densidad de probabilidad de los niveles de gris de la imagen transformada tendrá la siguiente expresión:

$$p_S(s) = \left[p_r(r) \frac{dr}{ds} \right]_{r=T^{-1}(s)}$$

Expresión válida si consideramos los niveles de gris como cantidades continuas (no discretas como realmente son). Por simplicidad, supondremos que esto es así en todos los desarrollos matemáticos que realicemos.

Ecualización global del histograma

La ecualización global del histograma de una imagen, al igual que otros muchos tratamientos que se basan en la manipulación del histograma, son mucho más usados para imágenes en escala de grises, por lo que será este caso en cuestión el que analicemos más profundamente. El tratamiento que vamos a estudiar en este caso es el de **ecualización** del mismo.

La ecualización del histograma consiste, muy básicamente y a grandes rasgos, en una expansión del histograma de la imagen, dotando al mismo de mayor linealidad y haciendo que éste ocupe el ancho del espectro de tonalidades grises por completo, ello implica unas mejoras en la imagen que serán expuestas a continuación:

- **Una mayor utilización de los recursos disponibles:** al ecualizar el histograma, vemos como los tonos que antes estaban más agrupados, ahora se han separado, ocupando todo el rango de grises, por lo que la imagen se está enriqueciendo al tener niveles de gris más distintos entre sí, mejorando, por tanto, la apariencia visual de la imagen.

- **Un aumento del contraste**: esta ventaja es consecuencia del punto anterior, ya que si hacemos que el histograma de la imagen ocupe todo el rango de grises, estamos aumentando la distancia entre el tono más claro y el más oscuro, convirtiendo a éstos, en blanco y negro y consecuentemente aumentando el contraste de la imagen.

- Constituye una **regulación óptima y automática del contraste** de la imagen. Evitando los ajustes manuales con los que no se consigue un equilibrio óptimo entre el blanco y el negro.

A su vez, aparecen algunos inconvenientes que surgen a la hora de ecualizar la imagen, algunos de ellos se detallan a continuación:

 Pérdida de información: puede ocurrir que a algunos pixeles que en la imagen original tenían distintos niveles de gris se les asigne, tras la ecualización global, al mismo nivel de gris. Por otro lado, hay casos en los que dos niveles de gris muy próximos se separen, dejando huecos en el histograma.

 En ocasiones, las bandas horizontales, fruto de una deficiente digitalización pueden resultar intensificadas, **resaltando aún más este error indeseado**.

Procesado del histograma hasta su ecualización

Vamos a describir las distintas operaciones que se le aplican al histograma de una imagen para el caso de que ésta esté en escala de grises de 8 bits, es decir 256 tonos distintos de gris.

Partimos del histograma original, el primer paso para trabajar con el mismo es el de normalización, para ello hay que normalizar tanto el eje horizontal (niveles de gris) como el vertical (aparición del nivel de gris en cuestión), de manera que ambas magnitudes queden comprendidas entre 0 y 1. Para normalizar el eje horizontal, es decir, el que indica el nivel de gris, dividimos cada magnitud entre 255, con lo que queda: 0, 1/255, 2/255...255/255=1, obteniendo un rango de niveles de gris comprendido entre 0 y 1. Para la normalización del eje vertical, debemos dividir cada componente entre el número total de pixeles, así en el caso más extremo, es decir en el que toda la imagen fuese de un mismo color, la representación sería una única barra de altura igual a 1.

El siguiente paso consiste en la acumulación del histograma, esto consiste en sustituir cada magnitud, por el valor de ella misma más el total de la suma de las anteriores, con ello conseguimos que el histograma (que reflejaba la función "densidad de probabilidad" para cada color) se transforme en una representación de la función de distribución. Quedando una función creciente, cuyo valor máximo será siempre 1. La acumulación del histograma, sería equivalente a la siguiente expresión:

$$s = T(r) = \int_0^r p_r(w)dw \quad para\ 0 \le r \le 1$$

Si los niveles de gris fuesen cantidades continuas (infinitos niveles de gris). Con lo que conseguiríamos que:

$$p_s(s) = \left[p_r(r)\frac{dr}{ds} \right]_{r=T^{-1}(s)} = \left[p_r(r)\frac{1}{p_r(r)} \right]_{r=T^{-1}(s)} = \left[1\right]_{r=T^{-1}(s)} = 1 \quad para\ 0 \le s \le 1$$

Es decir, la función densidad de probabilidad de los niveles de gris de la imagen resultante sería uniforme (como era nuestro objetivo).

Aplicando la siguiente expresión, obtendremos los nuevos niveles de gris (haciendo que el histograma se extienda hasta el negro), además conseguiremos que éstos vuelvan a encontrarse entre 0 y 255:

$$s^* = Int\left[\frac{s - s_{min}}{1 - s_{min}} 255 + 0.5\right]$$

· Donde "Int" representa la operación de tomar el entero más cercano por defecto.

· s_{min} es el menor valor de s distinto de cero.

· En nuestro caso, 255 representa el número de niveles de gris menos uno.

<u>Imagen obtenida con el software realizado</u>

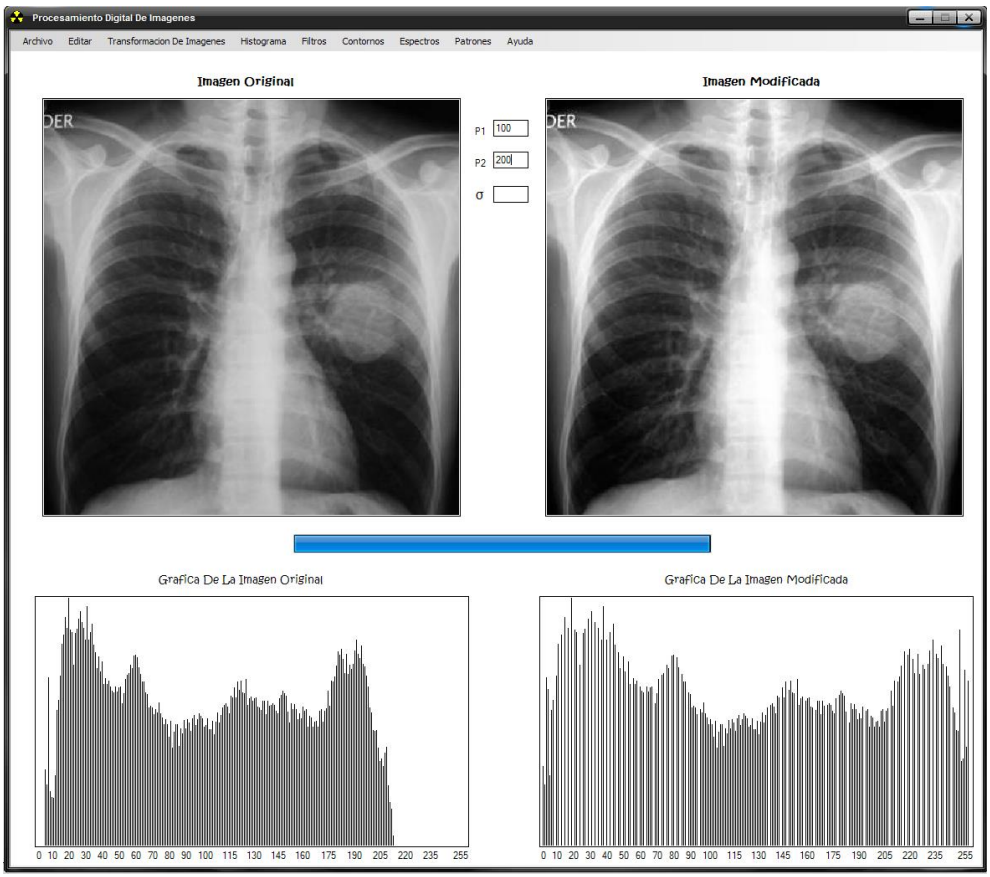
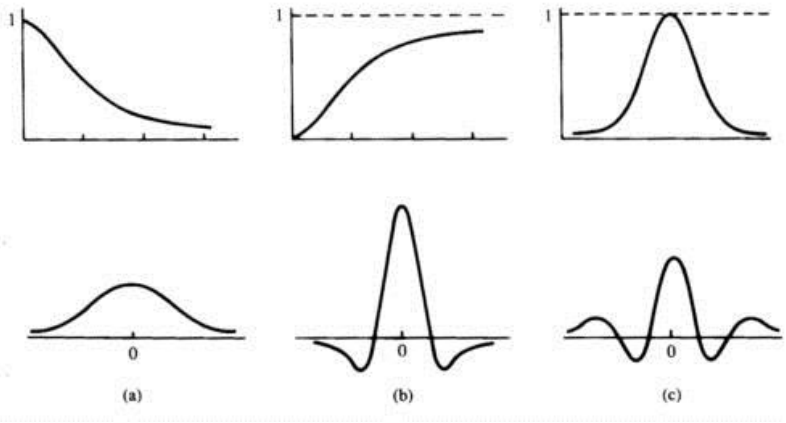


Imagen normal y ecualizada, cada una con su respectivo Histograma

FILTROS ESPACIALES

Los filtros espaciales tienen como objetivo modificar la contribución de determinados rangos de frecuencias a la formación de la imagen. El término espacial se refiere al hecho de que el filtro se aplica directamente a la imagen y no a una transformada de la misma, es decir, el nivel de gris de un píxel se obtiene directamente en función del valor de sus vecinos.

Los filtros espaciales pueden clasificarse basándose en su linealidad: ***filtros lineales*** y ***filtros no lineales***. A su vez los ***filtros lineales*** pueden clasificarse según las frecuencias que dejen pasar: los ***filtros paso bajo*** atenúan o eliminan las componentes de alta frecuencia a la vez que dejan inalteradas las bajas frecuencias; los ***filtros paso alto*** atenúan o eliminan las componentes de baja frecuencia con lo que agudizan las componentes de alta frecuencia; los ***filtros paso banda*** eliminan regiones elegidas de frecuencias intermedias.



Arriba: secciones de filtros en frecuencia con simetría circular.
Abajo: secciones correspondientes a filtros espaciales. (a) Filtro paso bajo. (b) Filtro paso alto. (c) Filtro paso banda.

La forma de operar de los filtros lineales es por medio de la utilización de máscaras que recorren toda la imagen centrando las operaciones sobre los píxeles que se encuadran en la región de la imagen original que coincide con la máscara y el resultado se obtiene mediante una computación (suma de convolución) entre los píxeles originales y los diferentes coeficientes de las máscaras.

Los ***filtros espaciales no lineales*** también operan sobre entornos. Sin embargo, su operación se basa directamente en los valores de los píxeles en el entorno en consideración. Unos ejemplos de filtros no lineales habituales son los filtros mínimo, máximo y de mediana que son conocidos como ***filtros de rango***.

FILTROS PASO BAJO

Su objetivo es suavizar la imagen, son útiles cuando se supone que la imagen tiene gran cantidad de ruido y se quiere eliminar. También pueden utilizarse para resaltar la información correspondiente a una determinada escala (tamaño de la matriz de filtrado); por ejemplo en el caso de que se quiera eliminar la variabilidad asociada a los tipos de cubierta presentes en la imagen uniformizando de esta manera su respuesta. Existen varias posibilidades:

- **Filtro de la Media**

De entre la multitud de máscaras de filtro paso bajo destaca especialmente la máscara de ***media***, que es la que efectúa el promedio de los valores del entorno. El filtro espacial de media reemplaza el valor de un píxel por la media de los valores del punto y sus vecinos. Su efecto es el difuminado o suavizado de la imagen y se aplica junto con el de mediana para eliminar ruidos.

Normalmente el tamaño de la máscara se toma en función de la cantidad de suavizado que queramos aplicar en cada momento. La visualización del resultado es el único medio de saber si hemos elegido el tamaño adecuado.

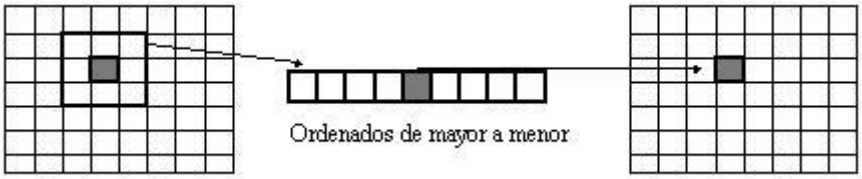
Se puede observar que el efecto final del filtro de la media es un suavizado de la imagen por reducción o redistribución del valor de los píxeles. Este filtro tiene el resultado opuesto a los de detección de bordes, donde el objetivo de los filtros es acentuar las diferencias, por esta razón el filtro de la media es un filtro paso bajo. También hay que notar que este filtro no modifica la imagen en las zonas donde el valor de los píxeles son el mismo, en oposición a los detectores de bordes que ponen estas regiones a cero.

En resumen, la media, como el resto de los filtros de suavizado, suaviza los contornos y otros detalles de forma de los objetos aparezcan menos definidos.

- **Filtro de la Mediana**

Los filtros de suavizado lineales o filtros paso bajo tienden a "difuminar los ejes" a causa de que las altas frecuencias de una imagen son atenuadas. La visión humana es muy sensible a esta información de alta frecuencia. La preservación y el posible realce de este detalle es muy importante al filtrar. Cuando el objetivo es más la reducción del ruido que el difuminado, el empleo de los ***filtros de mediana*** representan una posibilidad alternativa.

A menudo, las imágenes digitales se corrompen con ruido durante la transmisión o en otras partes del sistema. Esto se ve a menudo en las imágenes convertidas a digital de una señal de la televisión. Usando técnicas del filtrado de ruido, el ruido puede ser suprimido y la imagen corrompida se puede restaurar a un nivel aceptable. En aplicaciones de ingeniería eléctrica, el ruido se elimina comúnmente con un filtro paso bajo. El filtrado paso bajo es satisfactorio para quitar el ruido gaussiano pero no para el ruido impulsivo. Una imagen corrupta por ruido impulsivo tiene varios píxeles que tienen intensidades visiblemente incorrectas como 0 o 255. Hacer un filtrado paso bajo alterarán estas señales con los valores extremos sobre la vecindad del píxel. Un método mucho más eficaz para eliminar el ruido impulsivo es el filtrado de mediana.



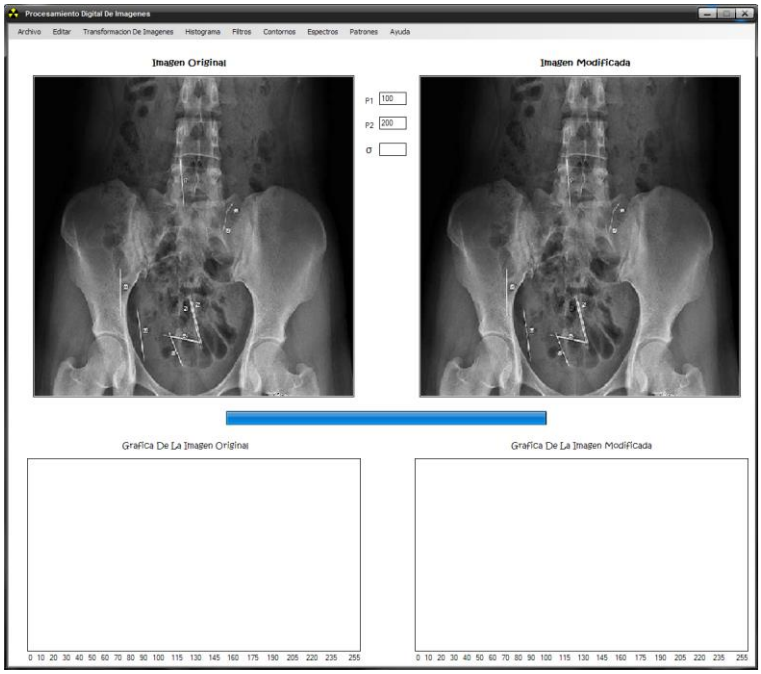
En el filtrado de mediana, el nivel de gris de cada píxel se reemplaza por la mediana de los niveles de gris en un entorno de este píxel, en lugar de por la media. Recordar que la mediana ***M*** de un conjunto de valores es tal que la mitad de los valores del conjunto son menores que ***M*** y la mitad de los valores mayores que ***M***, es decir en un conjunto ordenado de mayor a menor o viceversa, sería el valor de la posición central.

El filtro de la mediana no puede ser calculado con una máscara de convolución, ya que es un filtro no lineal. Podemos ver como este tipo de filtro elimina totalmente el punto que tenía un valor muy diferente al resto de sus vecinos. Como se selecciona el valor de centro, el filtrado de mediana consiste en forzar que puntos con intensidades muy distintas se asemejen más a sus vecinos, por lo que observamos que el filtro de mediana es muy efectivo para eliminar píxeles cuyo valor es muy diferente del resto de sus vecinos, como por ejemplo eliminando ruido de la imagen.

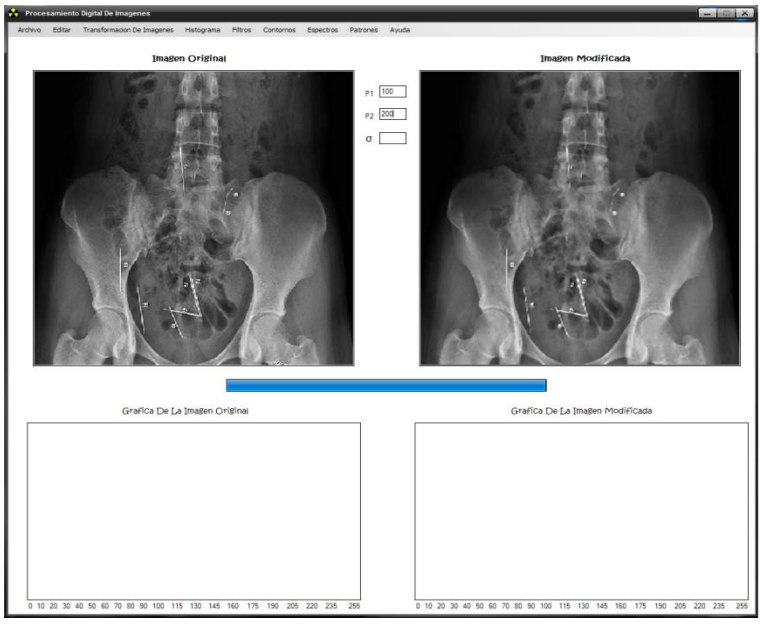
Tiene la ventaja de que el valor final del pixel es un valor real presente en la imagen y no un promedio, de este modo se reduce el efecto borroso que tienen las imágenes que han sufrido un filtro de media.

Además el filtro de la mediana es menos sensible a valores extremos. El inconveniente es que resulta más complejo de calcular ya que hay que ordenar los diferentes valores que aparecen en los pixeles incluidos en la ventana y determinar cuál es el valor central.

Imágenes obtenidas con software realizado



Filtro de la Media



Filtro de la Mediana

DETECCIÓN DE BORDES

Uno de los más importantes y sencillos procesados es la ***detección de bordes***. Importante porque de él se puede empezar a extraer importante información de la imagen, como pueden ser las formas de los objetos que la componen, y sencillo porque los operadores de detección de bordes son simples máscaras de convolución. Estos operadores son utilizados en aplicaciones para el reconocimiento de formas, aplicaciones industriales, militares, etc.

Dentro de las numerosas aplicaciones para la detección de bordes, los artistas digitales lo usan para crear imágenes con contornos deslumbrantes pues la salida de un detector de bordes puede ser agregada a una imagen original para realzar los bordes. La detección de bordes es a menudo el primer paso en la segmentación de imagen, que es un campo del análisis de la imagen, y se utiliza para agrupar los píxeles en regiones para determinar una composición de la imagen. La detección de bordes también es usada en el registro de imagen, el cual alinea dos imágenes que podrían ser adquiridas en momentos separados y de sensores diferentes.

Los ***bordes de una imagen*** contienen mucha de la información de la imagen. Los bordes cuentan donde están los objetos, su forma, su tamaño, y también sobre su textura. Los ejes o bordes se encuentran en zonas de una imagen donde el nivel de intensidad cambian bruscamente, cuanto más rápido se produce el cambio de intensidad, el eje o borde es más fuerte.

En general, los bordes de objetos en una imagen los podemos distinguir por los cambios más o menos bruscos de valor entre dos o más píxeles adyacentes. Podemos realizar una clasificación general de los bordes según sea su dirección en:

- *Bordes verticales*, cuando píxeles conectados verticalmente tienen valores diferentes respecto de los anteriores o posteriores.
- *Bordes horizontales*, cuando tenemos píxeles conectados horizontalmente, y estos tienen distintos valores respecto de los anteriores o posteriores.
- *Bordes oblicuos*, cuando tenemos una combinación de las componentes horizontales y verticales.
-

La diferencia entre los valores de los píxeles nos indica lo acentuado del borde, de forma que a mayores diferencias tenemos bordes más marcados y a menores tenemos unos bordes suavizados.

El proceso de detección de bordes se basa en realizar un incremento del contraste en las zonas donde hay una mayor diferencia entre las intensidades, y en una reducción de éste donde no tenemos variación de intensidad.

Los filtros utilizados para la detección de bordes son ***filtros diferenciales***, que se basan en la derivación o diferenciación. Dado que el promediado de los píxeles de una región tiende a difuminar o suavizar los detalles y bordes de la imagen, y esta operación es análoga a la integración, es de esperar que la diferenciación tenga el efecto contrario, el de aumentar la nitidez de la imagen, resaltando los bordes.

Derivada de primer orden

Muchas técnicas basadas en la utilización de máscaras para la detección de bordes utilizan máscaras de tamaño 3x3 o incluso más grandes. La ventaja de utilizar máscaras grandes es que los errores producidos por efectos del ruido son reducidos mediante medias locales tomadas en los puntos en donde se superpone la máscara. Por otro lado, las máscaras normalmente tienen tamaños impares, de forma que los operadores se encuentran centrados sobre los puntos en donde se calculan los gradientes.

$$\frac{\partial f(x,y)}{\partial x} = \Delta_x = \frac{f(x+d_x,y) - f(x,y)}{dx} \qquad \Delta_x = f(i+1,j) - f(i,j)$$

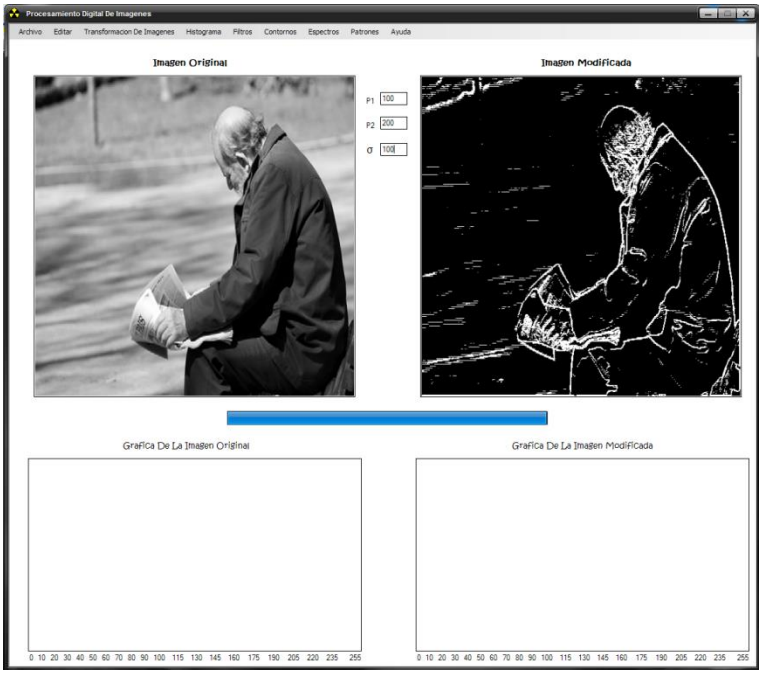
$$\frac{\partial f(x,y)}{\partial y} = \Delta_y = \frac{f(x,d_y+y) - f(x,y)}{dy} \qquad \Delta_y = f(i,j+1) - f(i,j)$$

Los operadores de gradiente común (o gradiente ortogonal) encuentran bordes horizontales y verticales. Estos operadores trabajan mediante convolución. Los operadores de ***Prewitt***, ***Sobel***, ***Roberts*** y ***Frei-Chen*** son operadores dobles o de dos etapas.

La detección de bordes se realiza en dos pasos, en el primero se aplica una máscara para buscar bordes horizontales, y en el segundo paso buscamos los verticales, el resultado final es la suma de ambos. Se muestran algunas máscaras de convolución comunes a continuación. Los detectores de fila (horizontales) son H_h y los detectores de columna (verticales) son H_v:

	Roberts	*Prewitt*	*Sobel*	*Frei-Chen*
H_h	0 0 -1 0 1 0 0 0 0	1 0 -1 1 0 -1 1 0 -1	1 0 -1 2 0 -2 1 0 -1	1 0 -1 $\sqrt{2}$ 0 $-\sqrt{2}$ 1 0 -1
H_v	-1 0 0 0 1 0 0 0 0	-1 -1 -1 0 0 0 1 1 1	-1 -2 -1 0 0 0 1 2 1	-1 $-\sqrt{2}$ -1 0 0 0 1 $\sqrt{2}$ 1

Imagen obtenida con el software realizado



Detección de Bordes

LA TRANSFORMADA DE FOURIER

Muchas técnicas de procesado de señal se hacen en un espacio matemático conocido como el *dominio de la frecuencia*. Para representar datos en el dominio de la frecuencia, algunas transformaciones son necesarias. Quizás la más estudiada es la ***transformada de Fourier (TF)***.

Por una señal continua entenderemos una función continua de una o varias dimensiones. Podemos encontrar ejemplos de distintos tipos de señales en los muy diversos aparatos de medida asociados al estudio de la física, química, biología, medicina, etc. Así por ejemplo, los distintos tipos de electrogramas que son usados en medicina son señales unidimensionales ya que se representan por una o varias curvas en función del tiempo, así como una señal de audio que va a un altavoz. Sin embargo, los distintos tipos de radiografías, así como todas las imágenes en 2D, son señales bidimensionales y los resultados de la tomografía axial computarizada y la resonancia nuclear magnética son señales tridimensionales.

Un prisma es un ejemplo común de cómo una señal es una composición de las señales de frecuencias que varían: mientras que la luz blanca pasa a través de un prisma, el prisma rompe la luz en sus componentes de frecuencia que revelan un espectro completo de color.

La frecuencia espacial de una imagen se refiere al rango en el cual las intensidades del píxel cambian. Las de alta frecuencia se concentran alrededor de los ejes que dividen la imagen en cuadrantes. Las esquinas tienen frecuencias más bajas, las frecuencias espaciales bajas se observan en áreas grandes de valores casi constantes.

Debido a su amplia gama de usos en el procesado de imagen, la transformada de Fourier es una de las más populares, y se aplica a una función continua de longitud infinita ***f(x)***. La expresión matemática de dicho cálculo es:

$$F(u) = \int_{-\infty}^{\infty} f(x) e^{j2\pi ux} dx$$

Donde $j = \sqrt{-1}$, y la variable ***u*** que aparece en la función ***F(u)*** representa a las frecuencias. Puede demostrarse además que esta transformación tiene inversa, es decir que dada la función ***F(u)*** podemos a partir de ella calcular la función ***f(x)***. La expresión matemática de dicha transformada inversa es:

$$f(x) = \int_{-\infty}^{\infty} F(u) e^{j2\pi ux} dx$$

Estas dos funciones ***f(x)*** y ***F(u)*** se denominan un ***par de transformadas de Fourier***. Es importante señalar que aunque las funciones que definen a las imágenes son funciones reales sus transformadas de Fourier son funciones complejas con parte real y parte imaginaria. Así pues ***F(u)*** se expresará de forma general como:

$$F(u) = R(u) + jI(u)$$

Donde ***R(u)*** denota la parte real e ***I(u)*** la parte imaginaria. Como todo número complejo para cada valor de ***u***, ***F(u)*** puede expresarse en términos de su ***módulo |F(u)|***, también conocido como ***espectro de frecuencia***, y de su ***ángulo de fase*** $\phi(u)$. Es decir, ***F(u)*** también puede expresarse como:

$$F(u) = |F(u)| e^{j\phi(u)} \quad \text{donde} \quad |F(u)| = [R^2(u) + I^2(u)]^{1/2}$$
$$\phi(u) = \tan^{-1}\left[\frac{I(u)}{R(u)}\right]$$

En el caso de que la señal sea una función de dos dimensiones los conceptos que hemos introducido para el caso unidimensional se generalizan a este caso de forma directa. Así el par de transformadas Fourier en notación matemática se expresan de la siguiente manera, donde ***u*** y ***v*** son variables de frecuencias.

$$F(u,v) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x,y) e^{-j2\pi(ux+vy)} dx dy \qquad \text{TF directa en 2D}$$

$$f(x,y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} F(u,v) e^{j2\pi(ux+vy)} du dv \qquad \text{TF inversa en 2D}$$

Rápidamente llega a ser evidente que las dos operaciones son muy similares con un signo menos en el exponente que es la única diferencia. Por supuesto, las funciones que se aplican son diferentes, una es una función espacial, la otra es una función frecuencial. Hay también un correspondiente cambio en las variables. En el dominio de la frecuencia, ***u*** representa la frecuencia espacial a lo largo del *eje x* de las imágenes originales y ***v*** representa la frecuencia espacial a lo largo del *eje y*. En el centro de la imagen ***u*** y ***v*** tienen su origen.

La transformada discreta de Fourier

Al trabajar con imágenes digitales, nunca nos dan una función continua, sino que debemos trabajar con un número finito de muestras discretas. Estas muestras son los píxeles que componen una imagen.

El análisis computarizado de imágenes requiere la ***transformada discreta de Fourier (DFT)***. La transformada discreta de Fourier es un caso especial de la transformada continua de Fourier.

En el caso discreto unidimensional, el par de transformadas de Fourier queda:

$$F(u) = \frac{1}{N} \sum_{x=0}^{N-1} f(x) e^{-j2\pi ux/N} \quad \text{para } u = 0, 1, 2, \ldots, N-1$$

$$f(x) = \sum_{u=0}^{N-1} F(u) e^{j2\pi ux/N} \qquad \text{para } x = 0, 1, 2, \ldots, N-1$$

En el caso bidimensional, el par de transformadas de Fourier discretas, para imágenes de tamaño ***M x N***, vendrán dadas por las siguientes expresiones:

$$F(u,v) = \frac{1}{MN} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x,y) e^{-j2\pi (ux/M + vy/N)} \qquad \begin{array}{l} \text{para } u = 0, 1, 2, \ldots, M\text{-}1 \\ \text{para } v = 0, 1, 2, \ldots, N\text{-}1 \end{array}$$

$$f(x,y) = \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} F(u,v) e^{j2\pi (ux/M + vy/N)} \qquad \begin{array}{l} \text{para } x = 0, 1, 2, \ldots, M\text{-}1 \\ \text{para } y = 0, 1, 2, \ldots, N\text{-}1 \end{array}$$

Cuando ***M = N*** algunas de las expresiones anteriores pueden expresarse de forma más sencilla. En particular el par de transformadas de Fourier tendrían las siguientes expresiones:

$$F(u,v) = \frac{1}{N} \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x,y) e^{-j2\pi (ux+vy)/N} \qquad \begin{array}{l} \text{para } u = 0, 1, 2, \ldots, N\text{-}1 \\ \text{para } v = 0, 1, 2, \ldots, N\text{-}1 \end{array}$$

$$f(x,y) = \frac{1}{N} \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} F(u,v) e^{j2\pi (ux+vy)/N} \qquad \begin{array}{l} \text{para } x = 0, 1, 2, \ldots, N\text{-}1 \\ \text{para } y = 0, 1, 2, \ldots, N\text{-}1 \end{array}$$

La transformada rápida de Fourier

La transformada discreta de Fourier es de cómputo intensivo requiriendo multiplicaciones N^2 complejas para un conjunto de ***N*** elementos. Se agrava este problema al trabajar con datos bidimensionales, como las imágenes.

Una imagen del tamaño ***M x M*** requerirá $(M^2)^2$ o M^4 multiplicaciones complejas. Afortunadamente, se descubrió que la transformada discreta de Fourier de longitud ***N*** se podría reescribir como la suma de dos transformadas de Fourier de longitud ***N/2***. Este concepto se puede aplicar recurrentemente al conjunto de datos hasta que se reduce a transformadas de solamente dos puntos.

Esta técnica de división y conquista se conoce como la ***transformada rápida de Fourier (FFT)***, que reduce el número de multiplicaciones complejas de N^2 al orden $N \log_2 N$. Estos ahorros son especialmente substanciales en el procesado de imagen. La FFT es separable, lo que incluso vuelve las transformadas de Fourier más fáciles de hacer.

Debido a la separabilidad, podemos reducir la operación de FFT de una operación bidimensional a dos operaciones unidimensionales. Primero procesamos la FFT de las filas de una imagen y en seguida seguimos con la FFT de las columnas.

Para una imagen del tamaño ***M x N***, esto requiere ***N + M*** FFTs para ser computadas. Del orden de $NM \log_2 NM$ cómputos son requeridos para transformar nuestra imagen.

Se debe recordar que la FFT no es una transformada diferente de la DFT, pero sí una familia de algoritmos más eficientes para lograr la transformada de datos. Generalmente cuando uno acelera un algoritmo, esta aceleración viene con un coste, con la FFT, el coste es complejidad. Hay complejidad en la ejecución de la contabilidad y del algoritmo. Los ahorros de cómputo, sin embargo, no se realizan a expensas de la exactitud.

Visualización del espectro

Hay que superar algunas dificultades al mostrar el espectro de frecuencia de una imagen. La primera surge debido al amplio rango dinámico de los datos resultantes de la transformada discreta de Fourier. En la imagen original el valor de un píxel será un número entero entre [0,255], representando el grado de intensidad, pero en la imagen que representa el espectro de Fourier los valores de los píxeles son números en punto flotante y no están limitado a los valores de [0,255]. Estos datos deben ser escalados de nuevo para transformarlos en un formato visible, de forma que no exceda la capacidad del dispositivo de visualización. Una cuantización lineal simple no proporciona siempre los mejores resultados, pues muchas veces se pierden los puntos de baja amplitud. El término cero de la frecuencia es generalmente el componente simple más grande, es también el punto menos interesante al examinar el espectro de la imagen. Una solución común a este problema es representar el logaritmo del espectro mejor que el espectro por sí mismo. La función que se aplica a la imagen del espectro para su representación es una ***función de compresión de rango dinámico***. La expresión matemática genérica de esta transformación para el caso de rangos muy grandes es:

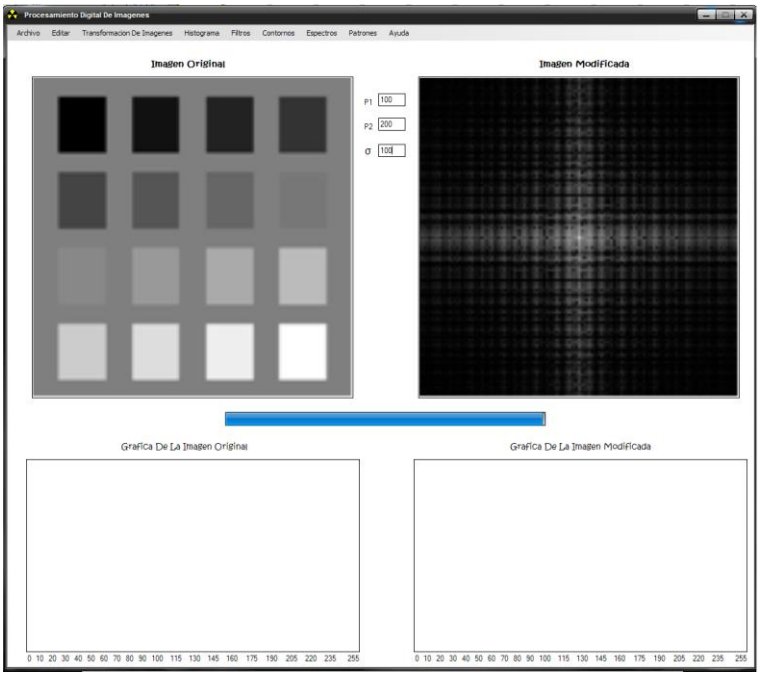
$$D(u,v)= c \log(1+|H(u,v)|)$$

Donde $|H(u,v)|$ es la magnitud de los datos a mostrar en frecuencia y c es una constante de escala que en el caso de una imagen con rango R toma el siguiente valor:

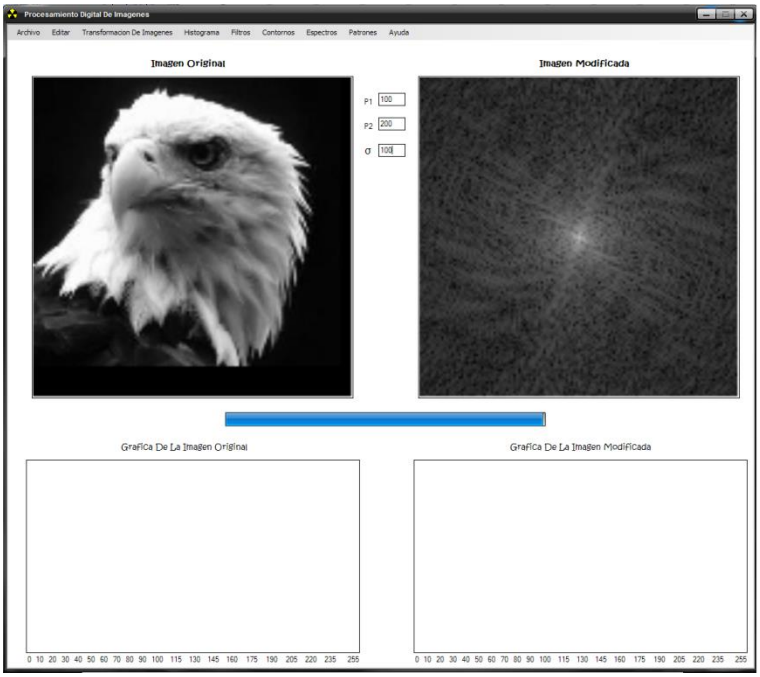
$$c = 255 / \log(1+|R|)$$

La suma de 1 asegura que el valor 0 del píxel no consigue pasar por la función del logaritmo.

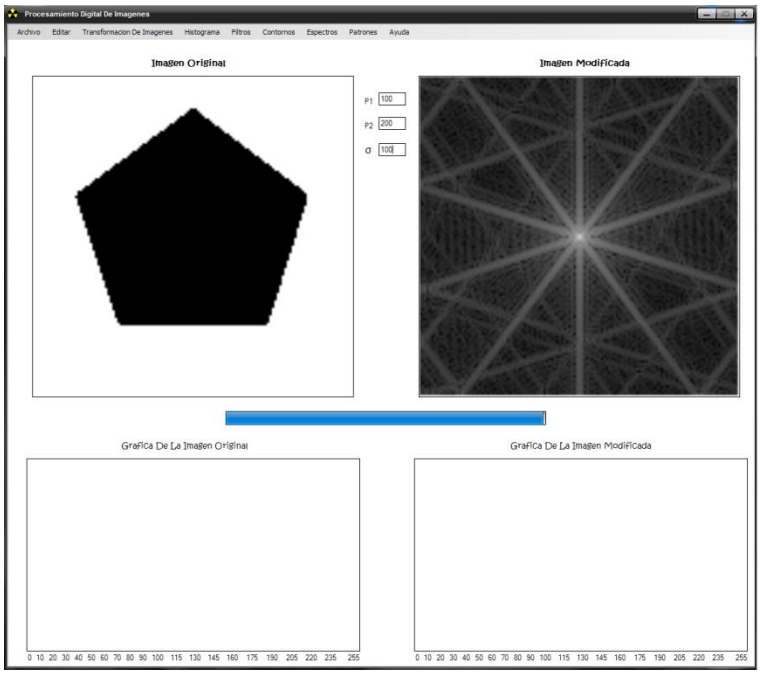
Imágenes obtenidas con software realizado



Espectro de Fourier 1



Espectro de Fourier 2



Espectro de Fourier 3

Teorema de Convolución

La gran importancia de la operación de convolución en el dominio frecuencial radica en el hecho de que la TF de la convolución de dos funciones es igual al producto de las TFs de dichas funciones, es decir:

$$f(x) \otimes g(x) \Leftrightarrow F(u)G(u)$$

Esto indica que la convolución en el domino de las ***x*** también se puede obtener realizando la transformada inversa de Fourier al producto ***F(u)G(u)***. Un resultado análogo al visto en la ecuación anterior es que la convolución en el dominio de frecuencias se reduce a la multiplicación en el dominio de las ***x***, es decir:

$$f(x)g(x) \Leftrightarrow F(u) \otimes G(u)$$

Estos dos resultados se conocen habitualmente con el nombre de ***Teorema de Convolución***, el cual implica que podemos calcular la convolución de dos funciones multiplicando sus correspondientes TF y al resultado aplicarle la TF inversa. En el caso de señales discretas, como en el caso de las imágenes, las distintas longitudes que pudieran tener las sucesiones de puntos de cada una de las funciones son posibles causas de errores en el cálculo final de la convolución, es por ello que ambas funciones han de definirse en una misma cantidad de puntos por cada eje.

Dado que la convolución bidimensional es análoga formalmente a la ecuación de la convolución unidimensional, el teorema de convolución en dos dimensiones se expresa entonces por las siguientes relaciones:

$$f(x,y) \otimes g(x,y) \Leftrightarrow F(u,v)G(u,v)$$
$$f(x,y)g(x,y) \Leftrightarrow F(u,v) \otimes G(u,v)$$

FILTRADO FRECUENCIAL

El filtrado en el dominio frecuencial incluye técnicas que están basadas en la modificación de la transformada de Fourier de la imagen. Los filtros paso bajo atenúan o eliminan los componentes de alta frecuencia en el dominio de Fourier, mientras dejan las bajas frecuencias sin alterar (esto es, éste tipo de filtros dejan "pasar" las frecuencias bajas). Las altas frecuencias son características de bordes, curvas y otros detalles en la imagen, así el efecto de un filtro paso bajo es el de difuminar la imagen. De igual forma, los filtros pasa alto, atenúan o eliminan las bajas frecuencias. Éstas son las responsables de las pequeñas variaciones de las características de una imagen, tal como pueden ser el contraste global y la intensidad media. El resultado final de un filtro paso alto es la reducción de estas características, y la correspondiente aparición de bordes y otros detalles de curvas y objetos.

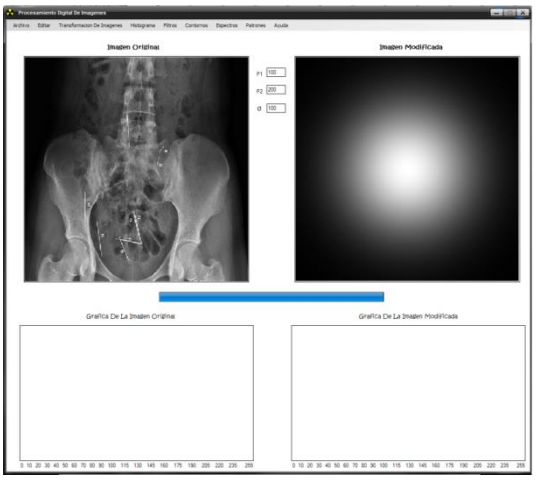
Anteriormente hemos visto ya cómo filtrar datos de la imagen mediante convoluciones en el dominio espacial, es también posible y muy común el filtrado en el dominio frecuencial. Tal y como enuncia el teorema de convolución, convolucionar dos funciones en el dominio espacial es igual que multiplicar sus espectros en el dominio frecuencial. De esta forma el proceso del filtrado en el dominio frecuencial es absolutamente simple:

1. Transformar los datos de la imagen al dominio frecuencial mediante la FFT.
2. Multiplicar el espectro de la imagen con alguna máscara de filtrado.
3. Transformar el espectro de vuelta al dominio espacial.

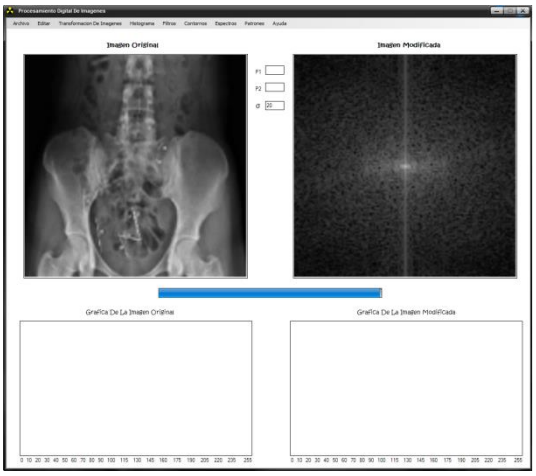
- **Filtros Gaussianos**

Simulan una distribución gaussiana bivariante. El valor máximo aparece en el pixel central y disminuye hacia los extremos tanto más rápido cuanto menor sea el parámetro de desviación típica s. El resultado será un conjunto de valores entre 0 y 1. Para transformar la matriz a una matriz de números enteros se divide toda la matriz por el menor de los valores obtenidos. La ecuación para calcularla es:

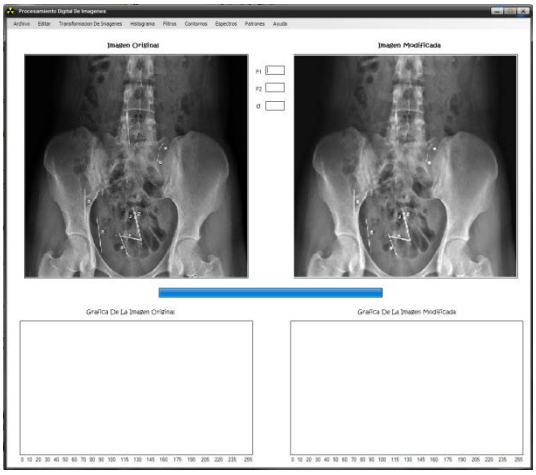
$$g(x,y) = e^{-\frac{x^2+y^2}{2*s^2}}$$
$$G(x,y) = \frac{g(x,y)}{min_{x,y}(g(x,y))}$$



Modulo del Filtro Gaussiano



Modulo de la Imagen



Filtro Gaussiano

RECONOCIMIENTO DE FORMAS CON IMÁGENES BINARIAS USANDO DESCRIPTORES DE FOURIER

Existen diferentes métodos para el reconocimiento de formas en imágenes binarias, algunos de estos métodos se basan en las propiedades geométricas tales como su área y su centro de masa. Cuando es posible describir la forma del objeto completamente por su contorno, existe la posibilidad de analizarlo de forma alternativa utilizando *Descriptores de Fourier* lo cual presenta la ventaja de disminuir el trabajo computacional.

En este trabajo se hace una revisión de esta técnica y se presentan los resultados obtenidos con diferentes imágenes de objetos.

La determinación de formas de objetos en imágenes binarias es un problema que encuentra diversas aplicaciones cuando se requiere detectar objetos con determinadas características. Muchas veces cuando se tiene una imagen de un objeto, este se puede encontrar colocado en diferentes partes del campo de visión, rotado o amplificado. Este tipo de situaciones hace difícil el detectar automáticamente si se trata de un objeto en particular, sin embargo esto se puede simplificar si se obtienen algunas propiedades geométricas, tales como su área y centro de masa. Si se representa una imagen como una función binaria f(x,y) tal que f(x,y)=1 en el objeto y f(x,y)=0 en otra parte (fig. 1), entonces el área del objeto se puede representar por la integral de dicha función sobre todo el campo de visión.

$$A = \iint_I f(x,y)\,dxdy \quad \textbf{(1)}$$

Para determinar la posición del objeto en el campo de visión es posible determinar el centro de la masa para escogerlo como un punto representativo de este. En un objeto bidimensional el momento sobre el eje *x* esta dado por:

$$\bar{x} \iint_I f(x,y)\,dxdy = \iint_I x f(x,y)\,dxdy \quad \textbf{(2)}$$

Mientras que para el eje *y* es:

$$\bar{y} \iint_I f(x,y)\,dxdy = \iint_I y f(x,y)\,dxdy \quad \textbf{(3)}$$

En donde

$(\bar{x}, \bar{y})$

Son las coordenadas del centro de masa

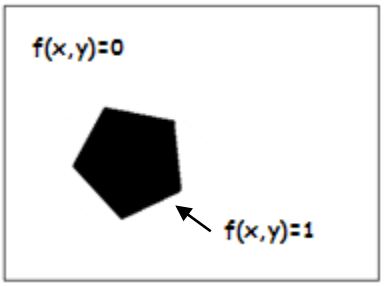


Figura 1.- Imagen binaria f(x,y) la cual toma valores de cero y uno

En una imagen digital el área del objeto binario se calcula con la sumatoria de todos los pixeles que corresponden a este de tal forma que la ecuación (1) será ahora:

$$A = \sum_{i=1}^{n} \quad \sum_{j=1}^{m} f_{ij} \quad (4)$$

Igualmente para las ecuaciones (2) y (3) las coordenadas del centro de la masa en forma discreta estarán dadas por:

$$\bar{x} = \frac{\sum_{i=1}^{n} \quad \sum_{j=1}^{m} i\, f_{ij}}{A} \quad \textbf{(5)} \qquad \bar{y} = \frac{\sum_{i=1}^{n} \quad \sum_{j=1}^{m} j\, f_{ij}}{A} \quad \textbf{(6)}$$

DESCRIPTORES POLARES DE FOURIER

Una forma de caracterizar el contorno de un determinado objeto se puede llevar a cabo tomando las distancias del centro de masa al contorno de diferentes ángulos, tomados a partir de uno de los ejes coordenados para obtener un vector **r** (fig. 2).

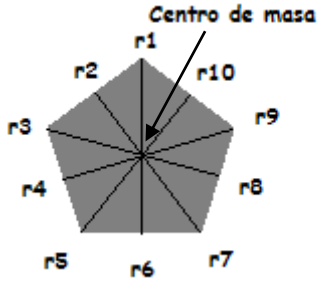


Figura 2.- Distancia del centro de masa a diferentes puntos del contorno

De esta forma se obtiene una secuencia de valores **r** de **N** elementos igualmente espaciados angularmente (vector característico), los Descriptores Polares de Fourier estarán dados por la transformada discreta de Fourier de esta secuencia:

$$R(m) = \sum_{n=0}^{N-1} r(n) exp\left(-\frac{i2\pi nm}{N}\right) \quad \textbf{(7)}$$

m = 0,1,2,…,N-1

Para hacer que estos descriptores sean invariantes a la rotación es posible utilizar la propiedad de desplazamiento de la transformada de Fourier. Como se sabe, la propiedad de desplazamiento dice que se tiene una función *r(x)* con su respectiva transformada de Fourier *R(ω)*, entonces cuando existe un desplazamiento $r(x - x_0)$ la transformada de Fourier será:

$$F\{r(x - x_0)\} = R(\omega) \exp(-i2\pi x_0 \omega) \quad \textbf{(8)}$$

Al obtener el modulo de la ecuación (8) este será invariante para cualquier desplazamiento x_0. En forma discreta el modulo R(m) estará dado por:

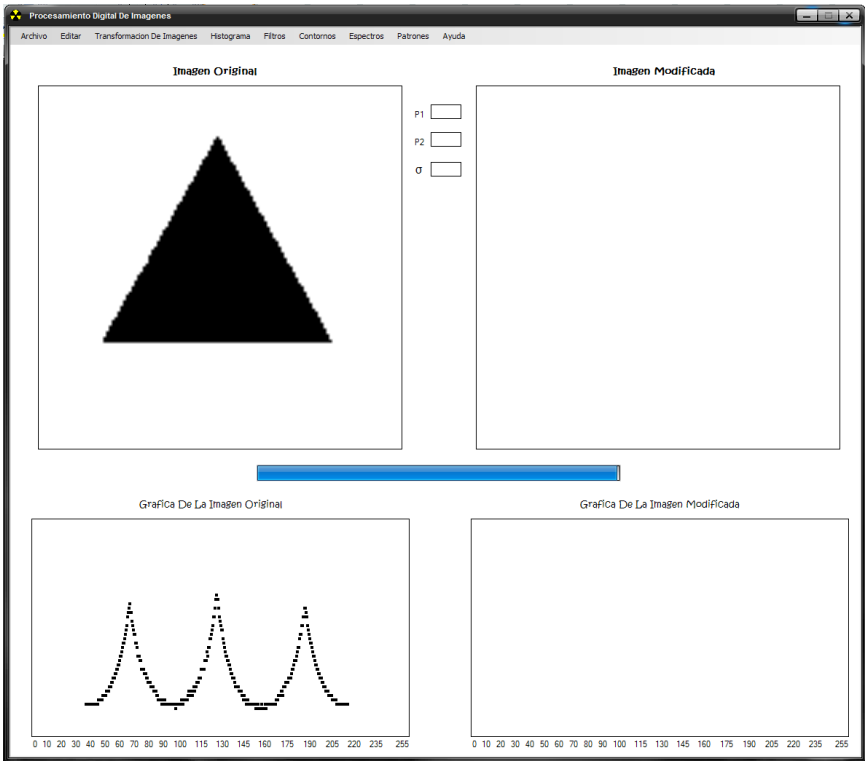
$$|R(m)| = \sqrt{Re\{R(m)\}^2 + Im\{R(m)\}^2} \quad \textbf{(9)}$$

m = 0,1,2,…,N-1

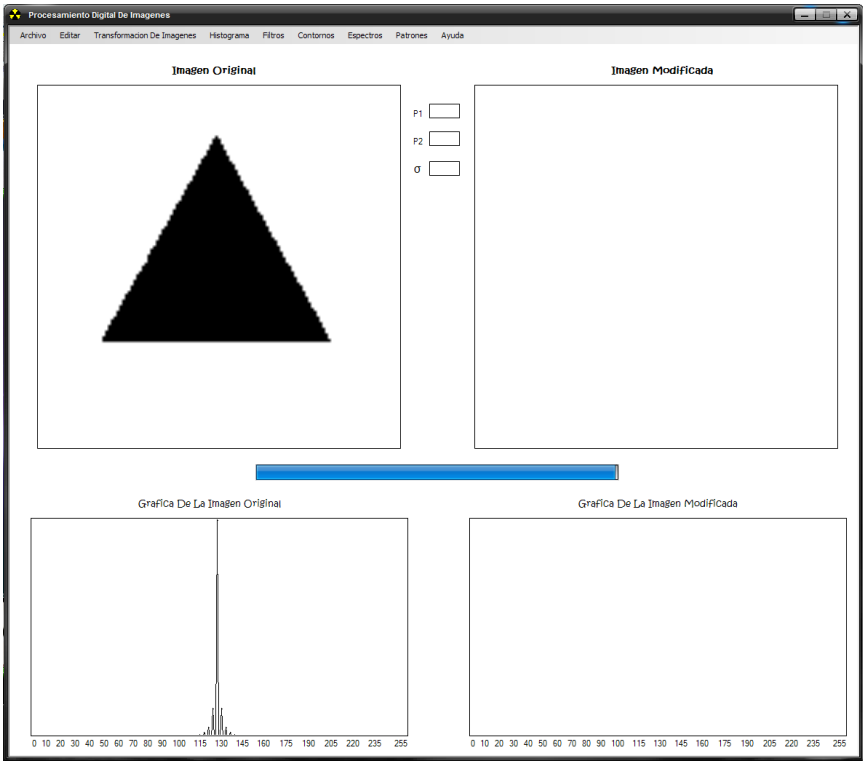
El vector característico es invariante a la escala si se normaliza haciéndolo de tal forma que:

$$r(n) = \frac{r(n)}{r_{max}} \quad \textbf{(10)}$$

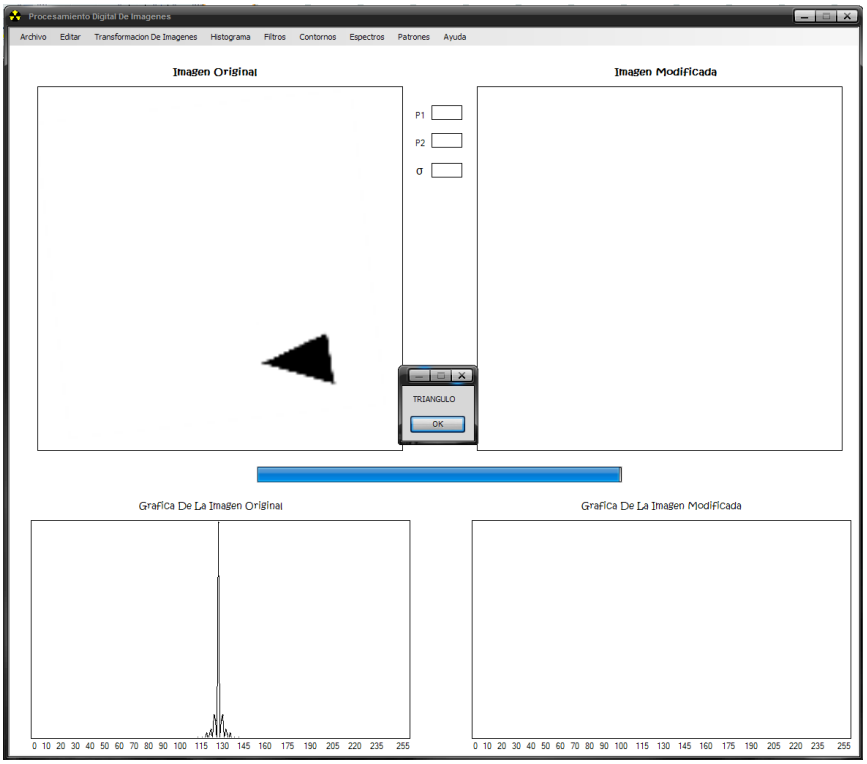
Imágenes obtenidas con software realizado



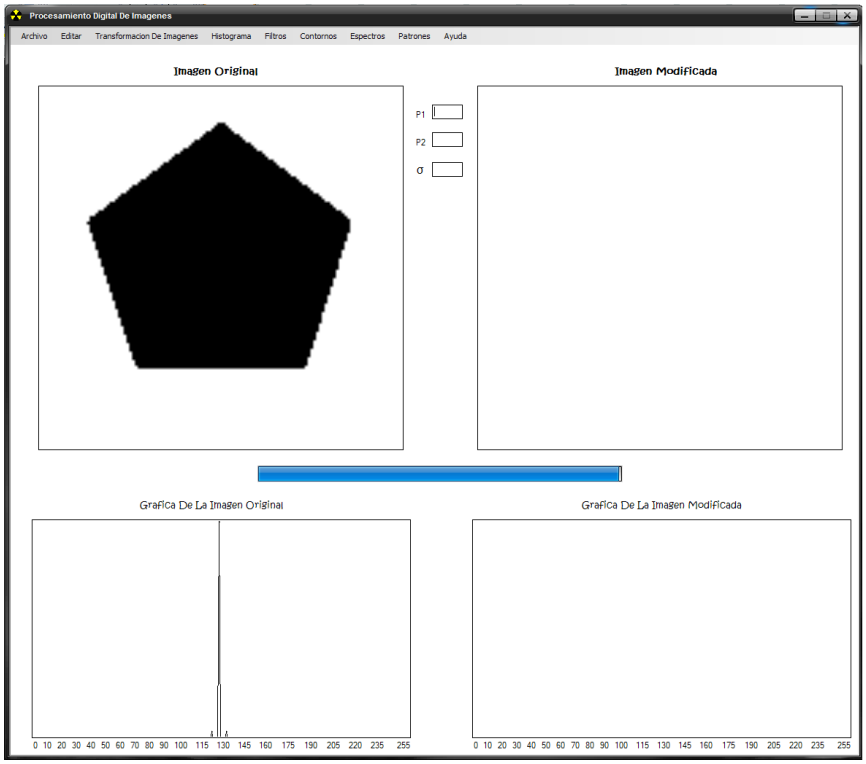
Grafica del Vector Característico



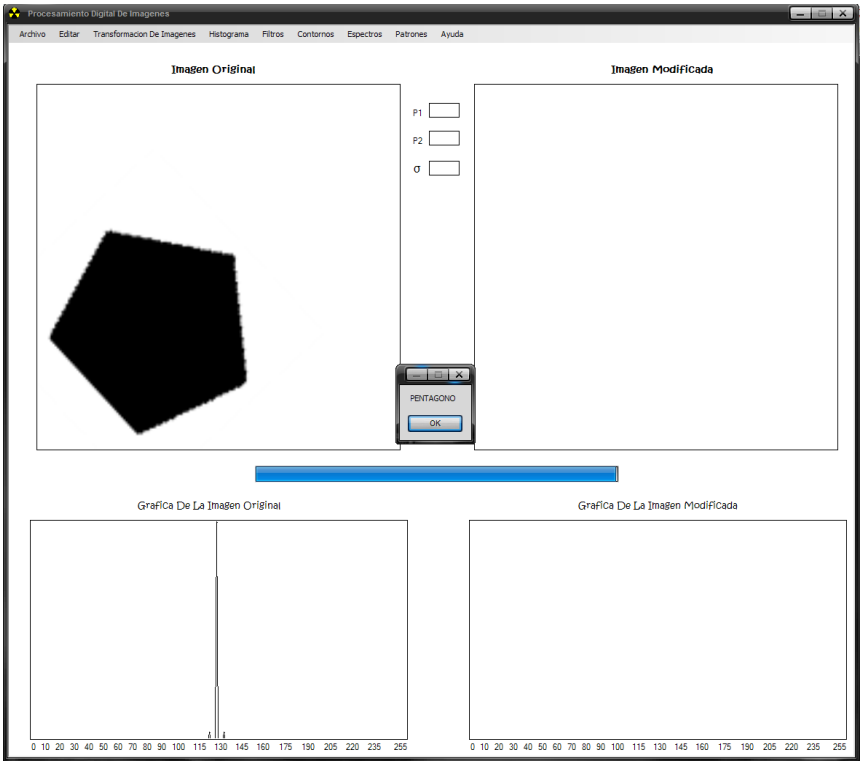
Grafica de sus Descriptores Polares de Fourier



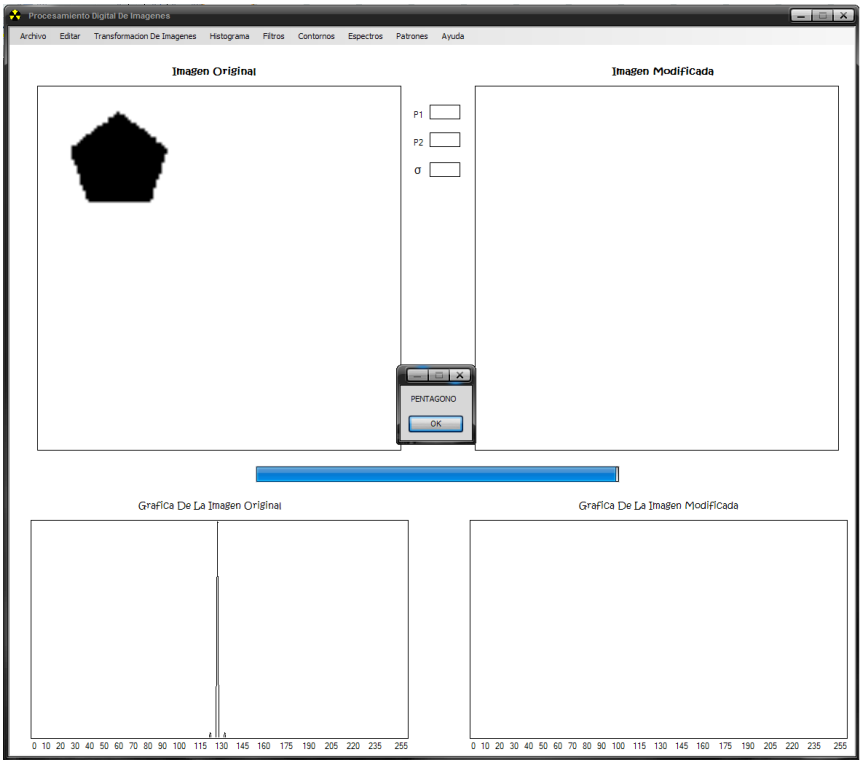
Reconocimiento de Forma y Grafica de sus Descriptores Polares de Fourier con la imagen rotada y a diferente escala



Grafica de sus Descriptores Polares de Fourier



Reconocimiento de Forma y Grafica de sus Descriptores Polares de Fourier con la imagen rotada



Reconocimiento de Forma y Grafica de sus Descriptores Polares de Fourier con la imagen rotada y a diferente escala

CONCLUSIÓN

En este trabajo de Residencia Profesional estudie algunas técnicas y algoritmos indispensables para el procesamiento digital de imágenes, tales como Transformación de Imágenes, Ecualización, Filtros Espaciales y Frecuenciales, Detección de Bordes y Descriptores de Fourier.

Debido al corto tiempo de mi estancia no pude continuar con otras técnicas tales como algoritmos genéticos y redes neuronales que son también de gran importancia en el campo del procesamiento digital de imágenes.

El software desarrollado durante mi estancia está enfocado hacia el reconocimiento de formas con imágenes binarias usando los Descriptores de Fourier, lo cual presenta la ventaja de que computacionalmente es menor el trabajo que el de otras técnicas. Los *Descriptores de Fourier* **son invariantes a la escala, la traslación y la rotación**, además de que muestran una buena descripción del contorno del objeto, por lo tanto este software puede ser utilizado para reconocimiento de cualquier objeto, tanto para su clasificación como para su detección.

Sin duda alguna la estancia en el CIO ha sido de las mejores experiencias que he tenido, además de que ha sido de mucha productividad y ayuda ya que he ampliado mis conocimientos y he reforzado mi pasión por la robótica y la inteligencia artificial.

BIBLIOGRAFÍA

- “Tratamiento Digital de Imágenes” Rafael C. González, Richard E. Woods Addison-Wesley / Díaz de Santos, 1996

- “Visión por computador, Imágenes digitales y aplicaciones” 2da. Edición
Gonzalo pajares Martinsanz, Jesús M. de la Cruz García
Alfaomega – Ra-Ma

- Reconocimiento de Formas con Imágenes Binarias Usando Descriptores de Fourier”
Dr. Francisco Javier Cuevas de la Rosa
Centro de Investigaciones en Óptica A.C.

ANEXO

PROGRAMA REALIZADO PARA DEMOSTRAR LAS TÉCNICAS Y ALGORITMOS EXPLICADOS EN ESTE REPORTE

COMPILADOR UTILIZADO

Visual C# 2008

Inicialización del programa:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {

//Declaracion de Variables Globales

        Bitmap image, imagen1, ecualizado;
        int i, j, p1, p2;
        int[] acumula;
        Color colorpixel,pixelc, npix, pix;

        int[,] matriz;
        int[,] Histograma;

        public Form1()
        {
            InitializeComponent();
        }

        private void button3_Click(object sender, EventArgs e)
        {
            Close();   // funcio para cerrar la ventana actual
        }


// Abrir una Imagen

        private void button1_Click(object sender, EventArgs e)
        {
// Abre una ventana de dialogo para abrir un archivo
            OpenFileDialog oFD = new OpenFileDialog();
// Titulo de la ventana de dialogo
            oFD.Title = "Seleccionar imagen";
//Tipos de imagenes que se podran abrir
            oFD.Filter = "Imágenes|*.jpg;*.gif;*.png;*.bmp|Todos (*.*)|*.*";
// El nombre del archivo seleccionado se almacenara en el texto del textbox1
            oFD.FileName = this.textBox1.Text;
            if (oFD.ShowDialog() == DialogResult.OK)
            {
                this.textBox1.Text = oFD.FileName;
// se carga a image la imagen selleccionada
                image = new Bitmap(this.textBox1.Text);
                imagen1 = image;
                ecualizado = image;

//se coloca la imagen en el picturebox1
                pictureBox1.Image = image;
                pictureBox3.Image = null;

            }
        }
// Convertir Imagen a Matriz

        private void button2_Click(object sender, EventArgs e)
        {
            image = new Bitmap(this.textBox1.Text);
            matriz = new int[image.Height, image.Width];          // el tamaño de la matriz esta
dado por

                                                  // el ancho y alto de la imagen cargada

// se indica el tamano del progressbar y su condicion inicial
                    progressBar1.Value = 0;
                    progressBar1.Maximum = image.Height;


// ciclo para convertir la imagen en una matriz

                    for (i = 0; i < image.Height; i++)
                    {
                        for (j = 0; j < image.Width; j++)
                        {
                            colorpixel = image.GetPixel(j, i);
                            matriz[i, j] = colorpixel.B;

                        }
                    }


// Operador Identidad

            if (radioButton1.Checked == true)
            {
                for (i = 0; i < image.Height; i++)
                {
                    for (j = 0; j < image.Width; j++)
                    {
                        matriz[i, j] = matriz[i, j];

                        // Se Imprime La Imagen Resultante
                        pixelc = Color.FromArgb(matriz[i, j], matriz[i, j], matriz[i, j]);
                        imagen1.SetPixel(j, i, pixelc);
                        pictureBox2.Image = imagen1;
                    }
                    progressBar1.Value = i;
                }
            }
```

```
// Operador Inverso o Negativo

            if (radioButton2.Checked == true)
            {
                for (i = 0; i < image.Height; i++)
                {
                    for (j = 0; j < image.Width; j++)
                    {
                        matriz[i, j] = 255 - matriz[i, j];

                        // Se Imprime La Imagen Resultante
                        pixelc = Color.FromArgb(matriz[i, j], matriz[i, j], matriz[i, j]);
                        imagen1.SetPixel(j, i, pixelc);
                        pictureBox2.Image = imagen1;
                    }
                    progressBar1.Value = i;
                }
            }


// Operador Umbral

            if (radioButton3.Checked == true)
            {

                p1 = Convert.ToInt32(textBox2.Text);

                for (i = 0; i < image.Height; i++)
                {
                    for (j = 0; j < image.Width; j++)
                    {

                        if (matriz[i, j] <= p1)
                        {
                            matriz[i, j] = 0;
                        }


                        if (matriz[i, j] > p1)
                        {
                            matriz[i, j] = 255;
                        }
                        // Se Imprime La Imagen Resultante
                        pixelc = Color.FromArgb(matriz[i, j], matriz[i, j], matriz[i, j]);
                        imagen1.SetPixel(j, i, pixelc);
                        pictureBox2.Image = imagen1;
                    }
                    progressBar1.Value = i;
                }
            }


// Operador Intervalo De Umbral Binario

            if (radioButton4.Checked == true)
            {

//los valores introducidos en los texbox se convierten de string a entero

                p1 = Convert.ToInt32(textBox2.Text);
                p2 = Convert.ToInt32(textBox3.Text);

                for (i = 0; i < image.Height; i++)
                {
                    for (j = 0; j < image.Width; j++)
                    {

                        if (matriz[i, j] <= p1 || (matriz[i, j] >= p2))
                        {
                            matriz[i, j] = 255;
                        }


                        if (p1 < (matriz[i, j]) && (matriz[i, j]) < p2)
                        {
                            matriz[i, j] = 0;
                        }
                        // Se Imprime La Imagen Resultante
                        pixelc = Color.FromArgb(matriz[i, j], matriz[i, j], matriz[i, j]);
                        imagen1.SetPixel(j, i, pixelc);
                        pictureBox2.Image = imagen1;
                    }
                    progressBar1.Value = i;
                }
            }


// Operador Intervalo De Umbral Binario Invertido

            if (radioButton5.Checked == true)
            {

                p1 = Convert.ToInt32(textBox2.Text);
                p2 = Convert.ToInt32(textBox3.Text);

                for (i = 0; i < image.Height; i++)
                {
                    for (j = 0; j < image.Width; j++)
                    {

                        if (matriz[i, j] <= p1 || (matriz[i, j] >= p2))
                        {
                            matriz[i, j] = 0;
                        }


                        if (p1 < (matriz[i, j]) && (matriz[i, j]) < p2)
                        {
                            matriz[i, j] = 255;
                        }
                        // Se Imprime La Imagen Resultante
```

```
                    pixelc = Color.FromArgb(matriz[i, j], matriz[i, j], matriz[i, j]);
                    imagen1.SetPixel(j, i, pixelc);
                    pictureBox2.Image = imagen1;
                }
                progressBar1.Value = i;
            }
        }

// Operador De Umbral De La Escala De Grises

        if (radioButton6.Checked == true)
        {

            p1 = Convert.ToInt32(textBox2.Text);
            p2 = Convert.ToInt32(textBox3.Text);

            for (i = 0; i < image.Height; i++)
            {
                for (j = 0; j < image.Width; j++)
                {

                    if (matriz[i, j] < p1 || (matriz[i, j] > p2))
                    {
                        matriz[i, j] = 255;
                    }


                    if (p1 < (matriz[i, j]) && (matriz[i, j]) < p2)
                    {
                        matriz[i, j] = matriz[i, j];
                    }

                    // Se Imprime La Imagen Resultante
                    pixelc = Color.FromArgb(matriz[i, j], matriz[i, j], matriz[i, j]);
                    imagen1.SetPixel(j, i, pixelc);
                    pictureBox2.Image = imagen1;
                }
                progressBar1.Value = i;
            }
        }

// Operador De Umbral De La Escala De Grises Invertido

        if (radioButton7.Checked == true)
        {

            p1 = Convert.ToInt32(textBox2.Text);
            p2 = Convert.ToInt32(textBox3.Text);

            for (i = 0; i < image.Height; i++)
            {
                for (j = 0; j < image.Width; j++)
                {

                    if (matriz[i, j] < p1 || (matriz[i, j] > p2))
                    {
                        matriz[i, j] = 255;
                    }


                    if (p1 < (matriz[i, j]) && (matriz[i, j]) < p2)
                    {
                        matriz[i, j] = 255 - matriz[i, j];
                    }
                    // Se Imprime La Imagen Resultante
                    pixelc = Color.FromArgb(matriz[i, j], matriz[i, j], matriz[i, j]);
                    imagen1.SetPixel(j, i, pixelc);
                    pictureBox2.Image = imagen1;
                }
                progressBar1.Value = i;
            }
        }

// Operador De Extension

        if (radioButton8.Checked == true)
        {

            p1 = Convert.ToInt32(textBox2.Text);
            p2 = Convert.ToInt32(textBox3.Text);

            for (i = 0; i < image.Height; i++)
            {
                for (j = 0; j < image.Width; j++)
                {

                    if (matriz[i, j] < p1 || (matriz[i, j] > p2))
                    {
                        matriz[i, j] = 0;
                    }


                    if (p1 < (matriz[i, j]) && (matriz[i, j]) < p2)
                    {
                        matriz[i, j] = ((matriz[i, j] - p1) * (255 / (p2 - p1)));
                    }
                    // Se Imprime La Imagen Resultante
                    pixelc = Color.FromArgb(matriz[i, j], matriz[i, j], matriz[i, j]);
                    imagen1.SetPixel(j, i, pixelc);
                    pictureBox2.Image = imagen1;
                }
                progressBar1.Value = i;
            }
        }

    }


// Código para obtener el histograma de la imagen


    private void button4_Click(object sender, EventArgs e)
    {
        int _B;

        Bitmap _Imagen = new Bitmap(this.textBox1.Text);
        Color _Color;

        Histograma = new int[_Imagen.Height, _Imagen.Width];

        Graphics _Picture = pictureBox3.CreateGraphics();
        Pen _Negro = new Pen(Color.Black);
        int _i, _j, _k;

        progressBar1.Maximum = _Imagen.Height; // Ancho de la imagen

        for (_i = 1; _i <= 256; _i++)
        {
            Histograma[1, _i] = 0;
        }

            for (_i = 0; _i < _Imagen.Height; _i++)
            {
                for (_j = 0; _j < _Imagen.Width; _j++)
                {

                    _Color = _Imagen.GetPixel(_j, _i);
                    _B = _Color.B;

                    Histograma[1, _B] = Histograma[1, _B] + 1;

                }
                progressBar1.Value = _i;
            }

        _k = 2;
        _i = 0;

        do
        {
            if (_i <= 256)
            {
                _Picture.DrawLine(_Negro, _k, 300, _k, Convert.ToInt32(300 - (0.08 *
(Histograma[1, _i]))));
            }
            _i = _i + 1;
            _k = _k + 2;
        }
        while (_k < 515);

    }


// Código para obtener el tono del pixel que se esté marcando con el mouse


private void pictureBox1_MouseMove(object sender, MouseEventArgs e)
    {
        if (checkBox1.Checked == true)
        {

            Bitmap _Imagen = new Bitmap(this.pictureBox1.Image);
            int r, g, b;

            textBox4.Text = Convert.ToString(e.X);
            textBox5.Text = Convert.ToString(e.Y);

            r = _Imagen.GetPixel(e.X, e.Y).R;
            g = _Imagen.GetPixel(e.X, e.Y).G;
            b = _Imagen.GetPixel(e.X, e.Y).B;

            textBox6.Text = Convert.ToString(r);

            pictureBox5.BackColor = Color.FromArgb(255, r, g, b);
        }
    }
//-------------ECUALIZACION DEL HISTOGRAMA-----------------------------------------
private void ecualizarHistogramaToolStripMenuItem_Click(object sender, EventArgs e)
    {

        if (pictureBox1.Image != null)
        {
                //--------------------se obtiene la imagen ----------------------------------
            image = new Bitmap(this.pictureBox1.Image);
            matriz = new int[image.Height, image.Width];

            progressBar1.Value = 0;
            progressBar1.Maximum = image.Height;

            for (i = 0; i < image.Height; i++)
            {
                for (j = 0; j < image.Width; j++)
                {
                    colorpixel = image.GetPixel(j, i);
                    matriz[i, j] = colorpixel.B;

                }
            }

            //------------------------------------------------------------

            float[] Ecualizado;
            int c1;
            float c2;

            Ecualizado = new float[256];

            for (int k = 0; k < image.Width; k++)
            {
                for (int l = 0; l < image.Height; l++)
                {
                    Ecualizado[matriz[l, k]]++; //---------se obtienen el histograma----------------------
----------
                }
```

```
            }
                //--------------------proceso de ecualizacion-----------------------------------
            for (int l = 0; l < 256; l++)
            {
                c2 = (float)Ecualizado[l] / (image.Height * image.Width);
                Ecualizado[l] = c2;
                Ecualizado[l] *= 255;
                if (l > 0)
                {
                    Ecualizado[l] += Ecualizado[l - 1];
                }
            }

            for (int l = 0; l < image.Height; l++)
            {
                for (int k = 0; k < image.Width; k++)
                {

                    c1 = (int)Ecualizado[matriz[l, k]];
                    matriz[l, k] = (Byte)c1;

                    pix = Color.FromArgb(matriz[l, k], matriz[l, k], matriz[l, k]);
                    imecualizada.SetPixel(k, l, pix);
                    pictureBox2.Image = imecualizada;
                }

                progressBar1.Value = l;

            }

        }
        else
        {
            MessageBox.Show("Tiene que abrir una imagen !!!");

        }
    }


//--------------------fin de la ecualizacion-------------------------------------------------------
//-------------filtro de la media-----------------------------------------


private void mediaToolStripMenuItem_Click(object sender, EventArgs e)
    {

        if (pictureBox1.Image != null)
        {
            image = new Bitmap(this.pictureBox1.Image);

            progressBar1.Value = 0;
            progressBar1.Maximum = image.Height;
            int fila = image.Height;
            int colum = image.Width;
            matriz = new int[fila, colum];

                //----------------se obtiene la imagen y se pasa a escala de grises---------------
-
            for (int f = 0; f < fila; f++)
            {
                for (int c = 0; c < colum; c++)
                {
                    numpix = image.GetPixel(c, f);
                    int r = Convert.ToInt32(numpix.R);
                    int g = Convert.ToInt32(numpix.G);
                    int b = Convert.ToInt32(numpix.B);
                    int a = Convert.ToInt32(numpix.A);
                    int pro = (r + g + b) / 3;
                    salpix = Color.FromArgb(a, pro, pro, pro);
                    matriz[f, c] = pro;
                    gris.SetPixel(c, f, salpix);
                }
            }

            //-----------------------------------------------------------------

                //----------declaracion de variables-----------------------------------------------
            int c1 = 1;
            ushort a1;
            Byte r1 = 2;
            Byte T = 128;

                //-------------comienza el filtrado-----------------------------------------
            for (int x = 0; x < image.Height; x++)
            {
                for (int y = 0; y < image.Width; y++)
                {
                    a1 = 1;
                    c1 = 0;
                    for (int l = x - r1; l <= x + r1; l++)
                    {
                        for (int k = y - r1; k <= y + r1; k++)
                        {
                            if (l >= 0 & l < image.Height & k >= 0 & k < image.Width)
                            {
                                c1 += matriz[l, k];
                            }
                            else
                            { a1++; }
                        }
                    }

                    c1 = ((c1 - matriz[x, y]) / ((r1 * 2 + 1) * (r1 * 2 + 1) - a1));
                    if (Math.Abs(c1 - matriz[x, y]) > T)
                        matriz[x, y] = (byte)c1;

                    pix = Color.FromArgb(matriz[x, y], matriz[x, y], matriz[x, y]);
                    immedia.SetPixel(y, x, pix);
                    pictureBox2.Image = immedia;  //---------se imprime la imagen filtrada-----------
----


                }
                progressBar1.Value = x;
            }

        }

        else
        {
            MessageBox.Show("Tiene que abrir una imagen !!!");

        }

    }
//--------------------fin del filtrado de la media-------------------------------------------------
---



//-------------Filtro de la Mediana-----------------------------------------
private void filtradoPorLaMedianaToolStripMenuItem_Click(object sender, EventArgs e)
    {

        if (pictureBox1.Image != null)
        {
            image = new Bitmap(this.pictureBox1.Image);

            progressBar1.Value = 0;
            progressBar1.Maximum = image.Height;
            int fila = image.Height;
            int colum = image.Width;


            matriz = new int[fila, colum];

                //------------se obtiene la imagen y se pasa a escala de grises--------------------
------------
            for (int f = 0; f < fila; f++)
            {
                for (int c = 0; c < colum; c++)
                {
                    numpix = image.GetPixel(c, f);
                    int r = Convert.ToInt32(numpix.R);
                    int g = Convert.ToInt32(numpix.G);
                    int b = Convert.ToInt32(numpix.B);
                    int a = Convert.ToInt32(numpix.A);
                    int pro = (r + g + b) / 3;
                    salpix = Color.FromArgb(a, pro, pro, pro);
                    matriz[f, c] = pro;
                    gris.SetPixel(c, f, salpix);
                }
            }

            for (int x = 0; x < image.Height; x++)
            {
                for (int y = 0; y < image.Width; y++)
                {
                    for (int t = 0; t < 8; t++)
                    {
                        acumula[t] = 0;
                    }
                    int ni = 0;
                    for (int i = x - 1; i <= x + 1; i++)
                    {
                        for (int j = y - 1; j <= y + 1; j++)
                        {
                            if (!(x == i && j == y)) //---se obtiene la ventana de 3x3----------------------
----
                            {
                                if (i > -1 && i < image.Height && j > -1 && j < image.Width && i < (x +
2) && j < (y + 2))
                                {
                                    acumula[ni] = matriz[i, j];
                                    ni++;
                                }
                            }
                        }
                    }

                //-----comienza el proceso de filtrado-----------------------------------------------
                    for (int k = 0; k < acumula.Length; k++)
                    {
                        for (int l = 0; l < acumula.Length - 1; l++)
                        {
                            if (acumula[l] > acumula[l + 1])
                            {
                                int aux = acumula[l];
                                acumula[l] = acumula[l + 1];
                                acumula[l + 1] = aux;
                            }
                        }
                    }
                    if (x > -1)
                    {
                        if (x == 0)
                        {
                            if (y == 0)
                            {
                                mediana = acumula[6];
                            }
                            if (y == image.Width - 1)
                            {
                                mediana = acumula[6];
                            }
                            if (y != image.Width - 1)
                            {
                                if (y != 0)
                                {
                                    mediana = acumula[5];
                                }
                            }
                        }
                        if (y == 0)
                        {
                            if (x != 0)
                            {
```

```
                        mediana = acumula[5];
                    }
                }

                if (x == image.Height - 1)
                {
                    if (y == 0)
                    {
                        mediana = acumula[6];
                    }
                    if (y == image.Width - 1)
                    {
                        mediana = acumula[6];
                    }
                }
                else
                {
                    mediana = acumula[4];
                }
            }


            pix = Color.FromArgb(mediana, mediana, mediana);
            immediana.SetPixel(y, x, pix);
            pictureBox2.Image = immediana; //-----se imprime la imagen filtrada-----------
-----
        }
        progressBar1.Value = x;
    }

}
else
{
    MessageBox.Show("Tiene que abrir una imagen !!!");

}
}

//---------------termina el proceso de filtrado-----------------------------------------------------------
-------
//----------Filtro Gaussiano-------------------------------------------------

private void gausianoToolStripMenuItem_Click(object sender, EventArgs e)
{


if (pictureBox1.Image != null)
{

    if (textBox7.Text != "")
    {

        image = new Bitmap(this.pictureBox1.Image);

        progressBar1.Value = 0;
        progressBar1.Maximum = image.Height;
        int fila = image.Height;
        int columna = image.Width;
        matriz = new int[fila, columna];

            //-------se obtiene la imagen y se pasa a escala de grises-------------------------
---------
        for (int f = 0; f < fila; f++)
        {
            for (int c = 0; c < columna; c++)
            {
                numpix = image.GetPixel(c, f);
                int r = Convert.ToInt32(numpix.R);
                int g = Convert.ToInt32(numpix.G);
                int b = Convert.ToInt32(numpix.B);
                int a = Convert.ToInt32(numpix.A);
                int pro = (r + g + b) / 3;
                salpix = Color.FromArgb(a, pro, pro, pro);
                matriz[f, c] = pro;
                gris.SetPixel(c, f, salpix);
            }
        }
        //-----------------------------------------------------------------

        double max = 0, d0 = 0, max1 = 0;

        //-----------------------DECLARACION DE MATRICES-----------------------------------
---
        matriz_D0 = new double[fila, columna];
        mask_pb_Gauss = new double[fila, columna];
        convo_S_FpbG = new double[fila, columna];
        double[,] mask_pb_Gauss_aux = new double[fila, columna];
            double[,] matriz_nr_Gauss = new double[fila, columna], matriz_ni_Gauss =
new double[fila, columna];

        //-----------------------------------------------------------------------------------
----------
        int au, au1;

        au = fila / 2;
        au1 = columna / 2;


        //------------MODULO DE FILTRO PASABAJAS GAUSSIANO----------------------
----
        int xn = 0;
        d0 = Convert.ToDouble(textBox7.Text);
        xn = (au - (fila / 2)) - 1;

        for (int x = 0; x < fila; x++)
        {
            for (int y = 0; y < columna; y++)
            {
                matriz_D0[x, y] = Math.Pow(Math.Pow((Convert.ToDouble(x)) -
(Convert.ToDouble(au)), 2) + Math.Pow((Convert.ToDouble(y)) - (Convert.ToDouble(au1)),
2), 0.5);

                if (matriz_D0[x, y] > max)
                    max = matriz_D0[x, y];
            }
        }


        filtro_pb_G = new Bitmap(fila + 1, columna + 1);

        for (int n = 0; n < fila; n++)
        {
            for (int m = 0; m < columna; m++)
            {
                double opera = Math.Pow(matriz_D0[n, m], 2) / (2 * Math.Pow(d0, 2));
                mask_pb_Gauss[n, m] = Math.Exp(-opera);


                if (mask_pb_Gauss[n, m] > max1)
                {
                    max1 = mask_pb_Gauss[n, m];
                }
            }
        }




        for (int n = 0; n < fila; n++)
        {
            for (int m = 0; m < columna; m++)
            {
                mask_pb_Gauss[n, m] = (255 * mask_pb_Gauss[n, m]) / max1;

                int cap = Convert.ToInt32(mask_pb_Gauss[n, m]);
                Color pix1 = Color.FromArgb(cap, cap, cap);
                filtro_pb_G.SetPixel(n, m, pix1); //--------se imprime el filtro ----------
---
            }
            progressBar1.Value = n;
        }

        pictureBox2.Image = filtro_pb_G;


    }

    else
    {
        MessageBox.Show("Introduzca El Valor De σ");
    }

}

else
{
    MessageBox.Show("Tiene que abrir una imagen !!!");
}

}

//-----------termina el proceso de filtrado-----------------------------------------------------------
---

//----------Obtencion de Bordes-------------------------------------------------

private void imagenOriginalToolStripMenuItem2_Click(object sender, EventArgs e)
{

    if (pictureBox1.Image != null)
    {
            //-------------------Variables para Contornos--------------------

            Bitmap contorno, gradientex, gradientey;
            int[] vect1 = { 1, 0, -1, -2, 0, 2, -1, 0, 1 };
            int[] vect2 = { -1, -2, -1, 0, 0, 0, 1, 2, 1 };
            int[] gx = new int[9];
            int[] gy = new int[9];
             int gra, g1, g2;
        image = new Bitmap(this.pictureBox1.Image);

        progressBar1.Value = 0;
        progressBar1.Maximum = image.Height;


        int fila = image.Height;
        int colum = image.Width;
        matriz = new int[fila, colum];
        for (int f = 0; f < fila; f++)
        {
            for (int c = 0; c < colum; c++)
            {
                numpix = image.GetPixel(c, f);
                int r = Convert.ToInt32(numpix.R);
                int g = Convert.ToInt32(numpix.G);
                int b = Convert.ToInt32(numpix.B);
                int a = Convert.ToInt32(numpix.A);
                int pro = (r + g + b) / 3;
                salpix = Color.FromArgb(a, pro, pro, pro);
                matriz[f, c] = pro;
                gris.SetPixel(c, f, salpix);
            }
        }

        //-----------------------------------------------------------------

        g1 = 0; g2 = 0; gra = 0;

        for (int x = 0; x < image.Height; x++)
        {
```

```
                for (int y = 0; y < image.Width; y++)
                {
                    for (int t = 0; t < 9; t++)
                    {
                        acumula[t] = 0;
                        gx[t] = 0;
                        gy[t] = 0;
                    }
                    int ni = 0; //----obtencion de la ventana de 3x3----------------------------------------
---------
                    for (int i = x - 1; i <= x + 1; i++)
                    {
                        for (int j = y - 1; j <= y + 1; j++)
                        {
                            if (i > -1 && i < image.Height - 1 && j > -1 && j < image.Width - 1 && i <
(x + 2) && j < (y + 2) && x > 0 && y > 0)
                            {
                                acumula[ni] = matriz[i, j];
                                ni++;
                            }
                        }
                    }

                     //---------obtencion del gradiente en X y gradiente en Y-------------------------------
------------------
                    for (int ko = 0; ko < 9; ko++)
                    {
                        gx[ko] = acumula[ko] * vect1[ko];
                        gy[ko] = acumula[ko] * vect2[ko];
                    }

                    g1 = gx[0] + gx[1] + gx[2] + gx[3] + gx[4] + gx[5] + gx[6] + gx[7] + gx[8];
                    g2 = gy[0] + gy[1] + gy[2] + gy[3] + gy[4] + gy[5] + gy[6] + gy[7] + gy[8];

                    if (g1 < 0)
                    {
                        g1 = g1 * (-1);
                    }
                    if (g2 < 0)
                    {
                        g2 = g2 * (-1);
                    }

                    gra = g1 + g2;


                    if (gra > 100)
                    {
                        contorno.SetPixel(y, x, Color.White);
                    }

                    if (gra < 100)
                    {
                        contorno.SetPixel(y, x, Color.Black);
                    }

                    pictureBox2.Image = contorno; //---------se imprime la imagen obtenida--------
-
                }

                progressBar1.Value = x;

            }
        }

        else
        {
            MessageBox.Show("Tiene que abrir una imagen !!!");

        }
    }


//----------termina el proceso de obtencion de contornos--------------------------------------------
-----



//------Espectro de Fourier-----------------------------------------------------


private void imagenModificadaToolStripMenuItem2_Click(object sender, EventArgs e)
/////Transformada de Fourier  === FFT ===
    {
        if (pictureBox1.Image != null)
        {

            int k, khat, bit, h;
            int M = image.Height;
            int N = image.Width;
            double[,] matriz_real = new double[M, N];
            double[,] matriz_imaginario = new double[M, N];
            double w, y_real, y_imaginario, z_real, z_imaginario;
            double[,] matriz_aux_real = new double[M, N];
            double[,] matriz_aux_imaginario = new double[M, N];
            double[,] matriz_modulo = new double[M, N];

            //Se obtiene la imagen y se convierte a escala de grises

            image = new Bitmap(this.pictureBox1.Image);

            progressBar1.Value = 0;
            progressBar1.Maximum = image.Height;


            int fila = image.Height;
            int colum = image.Width;


            matriz = new int[fila, colum];

            for (int f = 0; f < fila; f++)
            {
                for (int c = 0; c < colum; c++)
                {
                    numpix = image.GetPixel(c, f);
                    int r = Convert.ToInt32(numpix.R);
                    int g = Convert.ToInt32(numpix.G);
                    int b = Convert.ToInt32(numpix.B);
                    int a = Convert.ToInt32(numpix.A);
                    int pro = (r + g + b) / 3;
                    salpix = Color.FromArgb(a, pro, pro, pro);
                    matriz_real[f, c] = pro;
                    matriz_imaginario[f, c] = 0;
                    gris.SetPixel(c, f, salpix);
                }
            }
            //Procesado de la FFT de los pixeles extraidos y escalados.
            //Primera pasada por la FFT(filas)
            for (int j = 0; j < N; j++) //el indice j son las filas
            {
                //Sacamos de una en una las filas de la matriz de pixeles
                //y las desplazamos N/2.

                double[] In_real = new double[N];
                double[] In_imaginario = new double[N];

                for (int i = 0; i < N; i++)
                {
                    In_real[i] = (matriz_real[j, i] * Math.Cos(Math.PI * i)) + (matriz_imaginario[j, i]
* Math.Sin(Math.PI * i));
                    In_imaginario[i] = (-matriz_real[j, i] * Math.Sin(Math.PI * i)) +
(matriz_imaginario[j, i] * Math.Cos(Math.PI * i));
                }


                //Comienzo de la inversion de bits
                double[] Out_real = new double[N];
                double[] Out_imaginario = new double[N];
                for (k = 0, khat = 0; k < N; k++)
                {
                    Out_real[khat] = In_real[k];
                    Out_imaginario[khat] = In_imaginario[k];
                    for (bit = N / 2; (khat & bit) != 0; bit >>= 1)
                        khat ^= bit;

                    khat ^= bit;

                }
                //fin de la inversion

                //calculo de la fft de la fila j de la matriz de pixeles

                for (int n = 2; n <= N; n <<= 1)
                {
                    w = 2 * Math.PI / n;
                    for (int m = 0; m < N; m += n)
                    {
                        for (int x = 0; x < n / 2; x++)
                        {
                            y_real = Out_real[m + x];
                            y_imaginario = Out_imaginario[m + x];
                            z_real = Out_real[m + x + n / 2] * Math.Cos(x * w) + Out_imaginario[m
+ x + n / 2] * Math.Sin(x * w);
                            z_imaginario = -(Out_real[m + x + n / 2]) * Math.Sin(x * w) +
Out_imaginario[m + x + n / 2] * Math.Cos(x * w);
                            Out_real[m + x] = (y_real + z_real) / 2;
                            Out_imaginario[m + x] = (y_imaginario + z_imaginario) / 2;
                            Out_real[m + x + n / 2] = (y_real - z_real) / 2;
                            Out_imaginario[m + x + n / 2] = (y_imaginario - z_imaginario) / 2;
                        }
                    }
                }

                //metemos el resultado de la FFT de la fila en su fila
                //correspondiente en la matriz
                for (int r = 0; r < N; r++)
                {
                    matriz_real[j, r] = Out_real[r];
                    matriz_imaginario[j, r] = Out_imaginario[r];
                }
            }



            //Segunda pasada por la FFT(columnas)
            for (int j = 0; j < N; j++) //el indice j son las columnas
            {
                //Sacamos de una en una las columnas de la matriz de pixeles
                //y las desplazamos N/2.
                double[] In_real = new double[N];
                double[] In_imaginario = new double[N];
                for (int i = 0; i < N; i++)
                {
                    In_real[i] = (matriz_real[i, j] * Math.Cos(Math.PI * i)) + (matriz_imaginario[i, j]
* Math.Sin(Math.PI * i));
                    In_imaginario[i] = (-matriz_real[i, j] * Math.Sin(Math.PI * i)) +
(matriz_imaginario[i, j] * Math.Cos(Math.PI * i));
                }

                //Comienzo de la inversion de bits
                double[] Out_real = new double[N];
                double[] Out_imaginario = new double[N];
                for (k = 0, khat = 0; k < N; k++)
                {
                    Out_real[khat] = In_real[k];
                    Out_imaginario[khat] = In_imaginario[k];
                    for (bit = N / 2; (khat & bit) != 0; bit >>= 1)
                        khat ^= bit;

                    khat ^= bit;

                }
                //fin de la inversion

                //calculo de la fft de la columna j de la matriz de pixeles
```

```
                for (int n = 2; n <= N; n <<= 1)
                {
                    w = 2 * Math.PI / n;
                    for (int m = 0; m < N; m += n)
                    {
                        for (int x = 0; x < n / 2; x++)
                        {
                            y_real = Out_real[m + x];
                            y_imaginario = Out_imaginario[m + x];
                            z_real = Out_real[m + x + n / 2] * Math.Cos(x * w) + Out_imaginario[m
+ x + n / 2] * Math.Sin(x * w);
                            z_imaginario = -(Out_real[m + x + n / 2]) * Math.Sin(x * w) +
Out_imaginario[m + x + n / 2] * Math.Cos(x * w);
                            Out_real[m + x] = (y_real + z_real) / 2;
                            Out_imaginario[m + x] = (y_imaginario + z_imaginario) / 2;
                            Out_real[m + x + n / 2] = (y_real - z_real) / 2;
                            Out_imaginario[m + x + n / 2] = (y_imaginario - z_imaginario) / 2;
                        }
                    }
                }
                //metemos el resultado de la FFT de la columna en su columna
                //correspondiente en la matriz
                for (int r = 0; r < N; r++)
                {
                    matriz_real[r, j] = Out_real[r];
                    matriz_imaginario[r, j] = Out_imaginario[r];
                }
            }


            //fin de la segunda pasada

            //Hacemos el modulo de la FFT
            for (int g = 0; g < N; g++)
            {
                for (int f = 0; f < N; f++)
                {

                    //dividimos por el log de 10,para conseguir
                    //el logaritmo en base 10,ya que el metodo log
                    //nos hace el logaritmo natural (base e)
                    matriz_modulo[g, f] = Math.Log(1 + 100 * Math.Sqrt(Math.Pow(matriz_real[g,
f], 2) + Math.Pow(matriz_imaginario[g, f], 2))) / Math.Log(10);
                    matriz_modulo[g, f] = matriz_modulo[g, f] * 50;
                }
            }

            //metemos los pixeles de la matriz modulo
            //en la imagen modulo
            int[] pix_destino1 = new int[N * N];
            h = 0;
            for (int d = 0; d < N; d++)
            {
                for (int p = 0; p < N; p++)
                {
                    pix_destino1[h] = Convert.ToInt32(matriz_modulo[d, p]);
                    h++;
                }
            }

            h = 0;
            for (int g = 0; g < M; g++)
            {
                for (int f = 0; f < N; f++)
                {

                    pixelc = Color.FromArgb(pix_destino1[h], pix_destino1[h], pix_destino1[h]);
                    imagen1.SetPixel(f, g, pixelc);

                    pictureBox2.Image = imagen1; //----------se imprime el espectro de fourier----
---------
                    h++; ;

                }
                progressBar1.Value = g;
            }
        }
        else
        {
            MessageBox.Show("Tiene que abrir una imagen !!!");
        }

    }

//----------termina el proceso de obtencion del espectro de fourier----------------------------------
                    ----------


//======================Se abre la imagen
=============================================
//================================================================
================
private void graficarVectorCaracteristicoToolStripMenuItem_Click(object sender, EventArgs
e)
    {
        if (pictureBox1.Image != null)
        {
            image = new Bitmap(this.pictureBox1.Image);
            Graphics P = pictureBox3.CreateGraphics();

            distancia_contorno = new double[360];

            progressBar1.Value = 0;
            progressBar1.Maximum = image.Height;

            int fila = image.Height;
            int columna = image.Width;

            matriz = new int[fila, columna];

            for (int f = 0; f < fila; f++)
            {
                for (int c = 0; c < columna; c++)
                {
                    numpix = image.GetPixel(c, f);
                    int r = Convert.ToInt32(numpix.R);
                    int g = Convert.ToInt32(numpix.G);  //======Se convierte a escala de
grises ===
                    int b = Convert.ToInt32(numpix.B);
                    int a = Convert.ToInt32(numpix.A);
                    int pro = (r + g + b) / 3;
                    salpix = Color.FromArgb(a, pro, pro, pro);
                    matriz[f, c] = pro;
                    gris.SetPixel(c, f, salpix);
                }
            }

//================Comienza el proceso de obtencion del vector
caracteristico=================
//================================================================
==========

//======================Obtencion del area de la
figura================================

            double areaTot = 0;

            for (int i = 0; i < image.Height; i++)
            {
                for (int j = 0; j < image.Width; j++)
                {
                    int value = matriz[i, j];
                    if (value == 0)
                    {
                        areaTot++;
                    }
                }
            }


//======================Obtencion del centroide en
X================================

            int pixel;
            int num = 0, den = 0;
            for (int i = 0; i < image.Height; i++)
                for (int j = 0; j < image.Width; j++)
                {
                    pixel = matriz[i, j];

                    if (pixel == 0)
                    {
                        pixel = 255;
                        num += (pixel * i);
                        den += pixel;
                    }
                }
            int xc = num / den;


//========================Obtencion del centroide en
Y================================

            num = 0;
            den = 0;

            for (int i = 0; i < image.Height; i++)
            {
                for (int j = 0; j < image.Width; j++)
                {
                    pixel = matriz[i, j];

                    if (pixel == 0)
                    {
                        pixel = 255;
                        num += (pixel * j);
                        den += pixel;
                    }

                }
                progressBar1.Value = i;
            }
            int yc = num / (den);


//=========================Obtencion del vector
caracteristico==============================

            double[] rads = new double[360];
            double[] rads_norm = new double[360];
            int angle = 0;
            while (angle < 360)
            {
                int i = 1;
                int x = xc;
                int y = yc;
                int sum = 0;

                int argb = matriz[x, y];
                while ((matriz[x, y]) == 0)
                {
                    sum++;
                    x = (int)Math.Ceiling(xc + i * Math.Cos(angle * Math.PI / 180));
                    y = (int)Math.Ceiling(yc + i * Math.Sin(angle * Math.PI / 180));
                    i++;

                }
                rads[angle] = sum;
                angle++;

            }

//========================Se normaliza el vector
caracteristico================================

            double max = 0;
```

```
                for (int i = 0; i < rads.Length; i++)
                {
                    if (max < rads[i])
                    {
                        max = rads[i];
                    }
                }

                for (int i = 0; i < rads.Length; i++)
                {
                    rads[i] = rads[i] / max;
                }

                for (int i = 0; i < rads.Length; i++)
                {
                    rads_norm[i] = rads[i] * 300;
                }
//======================Se grafica el vector caracteristico ya
normalizado=====================

                int inc = 0;
                for (int k = 70; k <= 429; k++)
                {
                    P.FillEllipse(new SolidBrush(Color.Black), k, Convert.ToInt32(400 -
rads_norm[inc]), 5,5);
                    inc++;
                }
            }

            else
            {
                MessageBox.Show("Tiene que abrir una imagen !!!");
            }

        }


//======================Termina el proceso de graficacion del vector
caracteristico=================
//================================================================
===============



//================Comienza el proceso de obtencion del modulo de descriptores
polares de fourier======
//================================================================
===============
private void graficarModuloDeDescriptoresPolaresToolStripMenuItem_Click(object sender,
EventArgs e)
        {
            if (pictureBox1.Image != null)
            {
                image = new Bitmap(this.pictureBox1.Image);
                Graphics P = pictureBox3.CreateGraphics();

                distancia_contorno = new double[360];

                progressBar1.Value = 0;
                progressBar1.Maximum = image.Height;

                int fila = image.Height;
                int columna = image.Width;

                matriz = new int[fila, columna];

                for (int f = 0; f < fila; f++)
                {
                    for (int c = 0; c < columna; c++)
                    {
                        numpix = image.GetPixel(c, f);
                        int r = Convert.ToInt32(numpix.R);
                        int g = Convert.ToInt32(numpix.G);
                        int b = Convert.ToInt32(numpix.B);
                        int a = Convert.ToInt32(numpix.A);
                        int pro = (r + g + b) / 3;
                        salpix = Color.FromArgb(a, pro, pro, pro);
                        matriz[f, c] = pro;
                        gris.SetPixel(c, f, salpix);
                    }
                }


//======================Obtencion del area de la
figura========================

                double areaTot = 0;
                       double max = 0;

                for (int i = 0; i < image.Height; i++)
                {
                    for (int j = 0; j < image.Width; j++)
                    {
                        int value = matriz[i, j];
                        if (value == 0)
                        {
                            areaTot++;
                        }
                    }
                }

//======================Obtencion del centroide en
X==========================

                int pixel;
                int num = 0, den = 0;
                for (int i = 0; i < image.Height; i++)
                    for (int j = 0; j < image.Width; j++)
                    {
                        pixel = matriz[i, j];

                        if (pixel == 0)
                        {
                            pixel = 255;
                            num += (pixel * i);
                            den += pixel;
                        }
                    }

                int xc = num / den;

//========================Obtencion del centroide en
Y==========================

                num = 0;
                den = 0;

                for (int i = 0; i < image.Height; i++)
                {
                    for (int j = 0; j < image.Width; j++)
                    {
                        pixel = matriz[i, j];

                        if (pixel == 0)
                        {
                            pixel = 255;
                            num += (pixel * j);
                            den += pixel;
                        }

                    }
                    progressBar1.Value = i;
                }
                int yc = num / (den);

//========================Obtencion del vector
caracteristico========================

                double[] rads = new double[360];
                int angle = 0;
                while (angle < 360)
                {
                    int i = 1;
                    int x = xc;
                    int y = yc;
                    int sum = 0;

                    int argb = matriz[x, y];
                    while ((matriz[x, y]) == 0)
                    {
                        sum++;
                        x = (int)Math.Ceiling(xc + i * Math.Cos(angle * Math.PI / 180));
                        y = (int)Math.Ceiling(yc + i * Math.Sin(angle * Math.PI / 180));
                        i++;

                    }
                    rads[angle] = sum;
                    angle++;

                }

//======================DFT del vector
caracteristico=====================================

                int N = rads.Length;
                double[,] R= new double[2, N];

                for (int m = 0; m < N; m++)
                {
                    for (int n = 0; n < N; n++)
                    {
                        R[0, m] += rads[n] * Math.Cos(-2 * Math.PI * m * n / N); // Real
                        R[1, m] += rads[n] * Math.Sin(-2 * Math.PI * m * n / N); // Imaginary
                    }
                }

//======================Se obtiene el Modulo de
RM=====================================

                double[] R_magnitude = new double[N];
                double[] R_magnitude_1 = new double[N / 2];
                double[] R_magnitude_2 = new double[N / 2];
                double[] R_magnitude_final = new double[N];

                for (int m = 0; m < N; m++)
                {
                    R_magnitude[m] = Math.Sqrt(Math.Pow(R[0, m], 2) + Math.Pow(R[1, m], 2));
                }

//================Acomodo de los
datos================================================

                int q= 0;

                for (int n = 0; n < (N/2) - 1; n++)
                {
                    R_magnitude_1[n] = R_magnitude[n];
                }

                for (int n = 180; n < N; n++)
                {
                    R_magnitude_2[q] = R_magnitude[n];
                    q++;
                }

                for (int n = 0; n < (N / 2) - 1; n++)
                {
                    R_magnitude_final[n] = R_magnitude_2[n];
                }

                q = 0;

                for (int n = 180; n < N; n++)
                {
                    R_magnitude_final[n] = R_magnitude_1[q];
```

```
            q++;
        }


//============= Se normaliza
=================================================================
        max = 0;
        for (int i = 0; i < R_magnitude_final.Length; i++)
        {

            if (max < R_magnitude_final[i])
            {
                max = R_magnitude_final[i];
            }

        }

        for (int i = 0; i < R_magnitude_final.Length; i++)
        {
            R_magnitude_final[i] = R_magnitude_final[i] / max;
        }

        for (int i = 0; i < R_magnitude_final.Length; i++)
        {
            R_magnitude_final[i] = R_magnitude_final[i] * 300;
        }

//======================Se grafica el modulo de descriptores polares
=========================

        int inc = 0;
        Graphics _Picture = pictureBox3.CreateGraphics();
        Pen _Negro = new Pen(Color.Black);

        for (int k = -106; k <= 611; k+=2)
        {
            P.FillEllipse(new SolidBrush(Color.Black), k, Convert.ToInt32(300 -
R_magnitude_final[inc]), 2, 2);
            _Picture.DrawLine(_Negro, k, Convert.ToInt32(300 - R_magnitude_final[inc]),
k+2, Convert.ToInt32(300 -                                  R_magnitude_final[inc+1]));
            inc++;
        }
    }

    else
    {
        MessageBox.Show("Tiene que abrir una imagen !!!");

    }

}


//====================Termina el proceso de graficacion del modulo de
descriptores polares=========
//=================================================================
================


//================Comienza el proceso de reconocimiento de formas
============================
//=================================================================
================
private void reconocerFormaToolStripMenuItem_Click(object sender, EventArgs e)
    {
        if (pictureBox1.Image != null)
        {
            image = new Bitmap(this.pictureBox1.Image);
            Graphics P = pictureBox3.CreateGraphics();

            distancia_contorno = new double[360];

            progressBar1.Value = 0;
            progressBar1.Maximum = image.Height;

            int fila = image.Height;
            int columna = image.Width;

            matriz = new int[fila, columna];

            double max = 0;

            for (int f = 0; f < fila; f++)
            {
                for (int c = 0; c < columna; c++)
                {
                    numpix = image.GetPixel(c, f);
                    int r = Convert.ToInt32(numpix.R);
                    int g = Convert.ToInt32(numpix.G);
                    int b = Convert.ToInt32(numpix.B);
                    int a = Convert.ToInt32(numpix.A);
                    int pro = (r + g + b) / 3;
                    salpix = Color.FromArgb(a, pro, pro, pro);
                    matriz[f, c] = pro;
                    gris.SetPixel(c, f, salpix);
                }
            }
//======================Obtencion del area de la
figura=================================

            double areaTot = 0;

            for (int i = 0; i < image.Height; i++)
            {
                for (int j = 0; j < image.Width; j++)
                {
                    int value = matriz[i, j];
                    if (value == 0)
                    {
                        areaTot++;
                    }
                }
            }

//======================Obtencion del centroide en
X================================

            int pixel;
            int num = 0, den = 0;
            for (int i = 0; i < image.Height; i++)
                for (int j = 0; j < image.Width; j++)
                {
                    pixel = matriz[i, j];

                    if (pixel == 0)
                    {
                        pixel = 255;
                        num += (pixel * i);
                        den += pixel;
                    }
                }

            int xc = num / den;

//========================Obtencion del centroide en
Y================================

            num = 0;
            den = 0;
            for (int i = 0; i < image.Height; i++)
            {
                for (int j = 0; j < image.Width; j++)
                {
                    pixel = matriz[i, j];
                    if (pixel == 0)
                    {
                        pixel = 255;
                        num += (pixel * j);
                        den += pixel;
                    }
                }
                progressBar1.Value = i;
            }
            int yc = num / (den);

//=========================Obtencion del vector
caracteristico==============================

            double[] rads = new double[360];
            int angle = 0;
            while (angle < 360)
            {
                int i = 1;
                int x = xc;
                int y = yc;
                int sum = 0;

                int argb = matriz[x, y];
                while ((matriz[x, y]) == 0)
                {
                    sum++;
                    x = (int)Math.Ceiling(xc + i * Math.Cos(angle * Math.PI / 180));
                    y = (int)Math.Ceiling(yc + i * Math.Sin(angle * Math.PI / 180));
                    i++;

                }
                rads[angle] = sum;
                angle++;

            }

//======================TDF del vector
caracteristico=====================================

            int N = rads.Length;
            double[,] R = new double[2, N];

            for (int m = 0; m < N; m++)
            {
                for (int n = 0; n < N; n++)
                {
                    R[0, m] += rads[n] * Math.Cos(-2 * Math.PI * m * n / N); // Real
                    R[1, m] += rads[n] * Math.Sin(-2 * Math.PI * m * n / N); // Imaginary
                }
            }

//======================Se obtiene el Modulo de
RM==================================

            double[] R_magnitude = new double[N];
            double[] R_magnitude_1 = new double[N / 2];
            double[] R_magnitude_2 = new double[N / 2];
            double[] R_magnitude_final = new double[N];

            for (int m = 0; m < N; m++)
            {
                R_magnitude[m] = Math.Sqrt(Math.Pow(R[0, m], 2) + Math.Pow(R[1, m], 2));
            }

            //================Acomodo de
datos=========================================

            int q = 0;

            for (int n = 0; n < (N / 2) - 1; n++)
            {
                R_magnitude_1[n] = R_magnitude[n];
            }

            for (int n = 180; n < N; n++)
            {
                R_magnitude_2[q] = R_magnitude[n];
                q++;
            }
```

```
            for (int n = 0; n < (N / 2) - 1; n++)
            {
                R_magnitude_final[n] = R_magnitude_2[n];
            }

            q = 0;

            for (int n = 180; n < N; n++)
            {
                R_magnitude_final[n] = R_magnitude_1[q];
                q++;
            }

//============== Se normaliza
============================================================

            max = 0;
            for (int i = 0; i < R_magnitude_final.Length; i++)
            {
                if (max < R_magnitude_final[i])
                {
                    max = R_magnitude_final[i];
                }
            }

            for (int i = 0; i < R_magnitude_final.Length; i++)
            {
                R_magnitude_final[i] = R_magnitude_final[i] / max;
            }

            for (int i = 0; i < R_magnitude_final.Length; i++)
            {
                R_magnitude_final[i] = R_magnitude_final[i] * 300;
            }


//========================Se reconoce la
forma=============================================

            double[] abs_circulo = new double[N];
            double[] abs_cuadrado = new double[N];
            double[] abs_pentagono = new double[N];
            double[] abs_rectangulo = new double[N];
            double[] abs_triangulo = new double[N];

            double suma_circulo = 0;
            double suma_cuadrado = 0;
            double suma_pentagono = 0;
            double suma_rectangulo = 0;
            double suma_triangulo = 0;

            double prom_circulo = 0;
            double prom_cuadrado = 0;
            double prom_pentagono = 0;
            double prom_rectangulo = 0;
            double prom_triangulo = 0;

            for (int i = 0; i < R_magnitude_final.Length; i++)
            {
                abs_circulo[i] = Math.Abs(circulo[i] - R_magnitude_final[i]);
                abs_cuadrado[i] = Math.Abs(cuadrado[i] - R_magnitude_final[i]);
                abs_pentagono[i] = Math.Abs(pentagono[i] - R_magnitude_final[i]);
                abs_rectangulo[i] = Math.Abs(rectangulo[i] - R_magnitude_final[i]);
                abs_triangulo[i] = Math.Abs(triangulo[i] - R_magnitude_final[i]);
            }



            for (int i = 0; i < R_magnitude_final.Length; i++)
            {
                suma_circulo += abs_circulo[i];
                suma_cuadrado += abs_cuadrado[i];
                suma_pentagono += abs_pentagono[i];
                suma_rectangulo += abs_rectangulo[i];
                suma_triangulo += abs_triangulo[i];
            }
            prom_circulo = suma_circulo / N;
            prom_cuadrado = suma_cuadrado / N;
            prom_pentagono = suma_pentagono / N;
            prom_rectangulo = suma_rectangulo / N;
            prom_triangulo = suma_triangulo / N;


            if (prom_circulo < prom_cuadrado && prom_circulo < prom_pentagono &&
prom_circulo < prom_rectangulo && prom_circulo < prom_triangulo)
            {
                MessageBox.Show("CIRCULO");
            }

            if (prom_cuadrado < prom_circulo && prom_cuadrado < prom_pentagono &&
prom_cuadrado < prom_rectangulo && prom_cuadrado < prom_triangulo)
            {
                MessageBox.Show("CUADRADO");
            }

            if (prom_pentagono < prom_cuadrado && prom_pentagono < prom_circulo &&
prom_pentagono < prom_rectangulo && prom_pentagono < prom_triangulo)
            {
                MessageBox.Show("PENTAGONO");
            }

            if (prom_rectangulo < prom_cuadrado && prom_rectangulo < prom_pentagono
&& prom_rectangulo < prom_circulo && prom_rectangulo < prom_triangulo)
            {
                MessageBox.Show("RECTANGULO");
            }

            if (prom_triangulo < prom_cuadrado && prom_triangulo < prom_pentagono &&
prom_triangulo < prom_rectangulo && prom_triangulo < prom_circulo)
            {
                MessageBox.Show("TRIANGULO");
            }
        }

        else
        {
            MessageBox.Show("Tiene que abrir una imagen !!!");
        }
    }


//======================Termina el proceso de reconocimiento de formas
====================
```