UNIVERSIDAD NACIONAL DE INGENIERÍA

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

PROCESAMIENTO DIGITAL DE UNA REGIÓN DE INTERÉS EN UNA IMAGEN USANDO MATLAB

INFORME DE SUFICIENCIA

PARA OPTAR EL TÍTULO PROFESIONAL DE INGENIERO ELECTRÓNICO

PRESENTADO POR:

MIGUEL GRÁNDEZ VÁSQUEZ
PROMOCIÓN
1983-I

LIMA – PERÚ
2006

PROCESAMIENTO DIGITAL DE UNA REGIÓN DE INTERÉS EN UNA IMAGEN USANDO MATLAB

Dedico este trabajo a:

A mis padres Doña Carmela y Don Miguel por sus enseñanzas y el esfuerzo que pusieron en mi educación y en mi formación como persona,

A mis hijos Claudia Ariana, Elsa María y Miguel Angel para que germine en ellos la semilla del trabajo y el tesón en el logro de sus proyectos,

A mi único y gran amor……Deris…….por ser la luz que me ilumina siempre.

SUMARIO

Este informe pretende explicar el funcionamiento del procedimiento **ROIDEMO** escrito en MATLAB que realiza procesamiento digital de áreas seleccionadas en una imagen, y en las cuales aplica diversas operaciones como control de brillo y contraste, aplicación de ecualización de histograma, filtro pasabajo, inserción de ruido gaussiano entre otros.

El Capítulo I presenta las bases teóricas para el procesamiento digital de señales: adquisición y almacenamiento de imágenes digitales, técnicas de procesamiento basadas en puntos de imagen y procesamiento basado en una región de la imagen.

El capítulo II presenta el formato de señales con las cuales trabaja el MATLAB y revisa conceptos de realce y restauración de imágenes así como una presentación breve de las transformadas y la codificación en dicho entorno.

El Capítulo III describe el funcionamiento del procedimiento **roidemo** construído en MATLAB y se discute su estructura a nivel de las subrutinas que lo conforman.

El capítulo IV muestra el estudio de algunas aplicaciones prácticas en las cuales puede se utilizar el procedimiento **roidemo**.

Se tiene en la parte final del informe dos anexos que complementan el estudio realizado. El **ANEXO A** que hace una descripción de los comandos más importantes usados en el procedimiento y el **ANEXO B** que contiene un listado completo del procedimiento mencionado.

ÍNDICE

CAPITULO IV

APLICACIONES USANDO ARCHIVOS EXTERNOS AL PROCEDIMIENTO ROIDEMO **59**

PRÓLOGO

El procesamiento digital de imágenes, incluye un conjunto de técnicas que operan sobre la representación digital de una imagen, a objeto de destacar algunos de los elementos que conforman la escena, de modo que se facilite su posterior análisis, bien sea por parte de un usuario (humano) o un sistema de visión artificial. En general, las técnicas de procesamiento de imágenes son aplicadas cuando resulta necesario realzar o modificar una imagen para mejorar su apariencia o para destacar algún aspecto de la información contenida en la misma, o cuando se requiere, medir, contrastar o clasificar algún elemento contenido en la misma. También se utilizan técnicas de procesamiento, cuando se requiere combinar imágenes o porciones de las mismas o reorganizar su contenido.

En estos últimos tiempos el área de Electrónica a evolucionado en torno al software que es utilizado para reemplazar muchas soluciones que antes eran implementadas en su mayor parte con hardware. En el caso puntual del tratamiento de imágenes, la existencia de muchos lenguajes especializados de programación como el MATLAB han permitido realizar procesos y controles de una manera mucho más eficiente y versátil teniendo en cuenta la rapidez y precisión de las computadoras disponibles en la actualidad.

De esta manera las aplicaciones de las técnicas usadas en el procedimiento que es motivo del presente informe son muy amplias y abarcan desde el mejoramiento de imágenes escaneadas de páginas de textos importantes hasta el tratamiento de fotografías en las cuales se pretende resaltar áreas específicas de interés.

El presente trabajo que revisa la secuencia de instrucciones y conjunto de ellas que usa el MATLAB en el procedimiento **roidemo** para aplicar un cierto proceso en una determinada región dentro de una imagen, quiere enfatizar en el hecho que dichas instrucciones pueden simular y manipular datos de imágenes obtenidas de la vida real con una potencia en el procesamiento y mejoramiento de las mismas, de tal forma que reemplazan a procedimientos físicos engorrosos y tediosos y que años atrás eran utilizados con una menor eficiencia en los resultados.

CAPITULO I

BASES DEL PROCESAMIENTO DIGITAL DE IMÁGENES

1.1 Adquisición y almacenamiento de imágenes digitales

Las imágenes digitales representan información visual asociada con una escena ambiental real que correspondería a lo que observamos con el sentido de la vista o bien información no visible pero que puede ser medida utilizando sensores apropiados tales como radiación infraroja, ultravioleta, rayos X ultrasonidos, etc.

El proceso de adquisición de la imagen involucra un sensor apropiado para detectar el tipo de fuente de información visual o emisión y convertirla en una señal eléctrica que se convertirá en un arreglo de cantidades binarias las cuales se pueden almacenar o procesar utilizando una computadora. La imagen digital corresponde a un arreglo de dos dimensiones (2D) que se podría denotar como $f(x,y)$ en donde cada punto se denomina pixel y tiene asociadas las coordenadas espaciales definidas por x e y. La imagen tiene un tamaño de NxM pixels en donde N corresponde al ancho de la imagen y M corresponde al largo de la imagen. Cada pixel corresponde a un valor de intensidad representativa de la información visual o emisión que se ha adquirido. Tal valor binario requiere un determinado numero de bits para representar la

información y lo mas usual es 8 bits que corresponde a un byte o bien, 16 bits o 32 bits que corresponden a 2 bytes y 4 bytes respectivamente. Las imágenes tri-dimensionales (3D) se denotan como *f(x,y,z)* en donde cada punto se denomina voxel y tiene asociadas tres coordenadas espaciales definidas por **x**, **y** , **z**. En este caso el tamaño total sería NxMxP voxels y es equivalente a manejar P imágenes bidimensionales cada una de tamaño NxM pixels. Una vez adquirida la imagen se puede procesar y/o almacenar en disco duro, cintas magnéticas, discos compactos (CD), etc.

1.2 Técnicas de Procesamiento basadas en Puntos de la Imagen

Estas técnicas consisten en algoritmos que modifican el valor de un pixel basados únicamente en el valor previo de tal pixel o en su localización. Ningún otro valor de pixel se involucra en la transformación. El procesamiento se realiza desarrollando un barrido pixel por pixel dentro de la imagen a procesar. Si la transformación a aplicar, depende solo del valor original del pixel, en su implantación, puede resultar de utilidad el uso de tablas de búsqueda (LUT/Look-Up Table). Si por el contrario, se considera además del valor previo del pixel, la posición del mismo, puede resultar necesario utilizar fórmulas o una combinación de las mismas con tablas de búsqueda. De manera general estas técnicas no modifican las relaciones espaciales dentro de la imagen y en consecuencia no pueden modificar el grado de detalle contenido en las mismas, son simples y pueden resultar útiles solas o en conjunto con otras técnicas mas complejas.

1.2.1 Histograma de una Imagen

El histograma de una imagen es ampliamente utilizado como herramienta tanto cualitativa como cuantitativa. Este corresponde a un gráfico de la distribución de valores de intensidad de los pixeles de una imagen (niveles de gris) o de una porción de la misma. Podemos denotar como **h(i)**, el número de pixeles que dentro de la región de interés tiene el valor de intensidad **i**, donde **i = 0, 1, 2, ..., L-1** es el número posible de niveles de gris para la imagen. Los valores **h(i)**, corresponderán entonces a los valores del histograma. El gráfico del histograma es bidimensional y en él se grafica **h(i)** en función de **i**. Tal gráfico, puede proporcionar importante información acerca del brillo y contraste de una imagen así como de su rango dinámico. En la figura 1.1 se muestra el dibujo de un histograma típico.

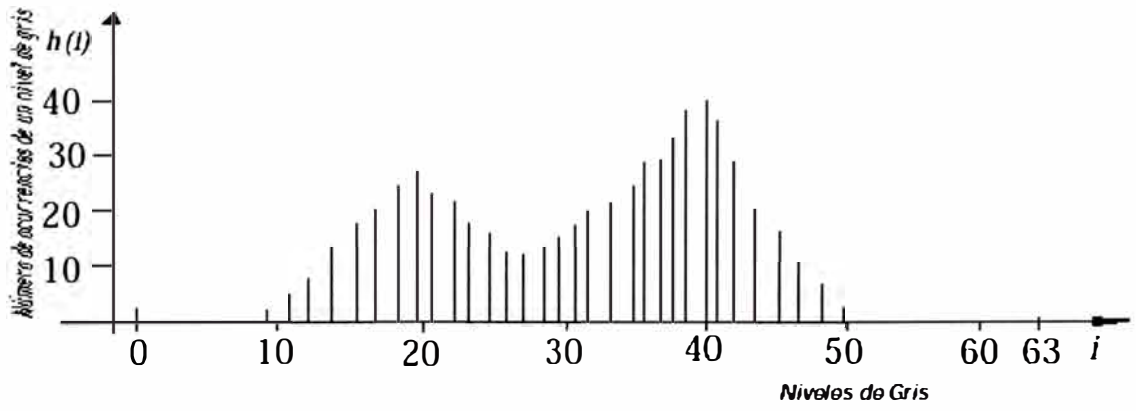


Figura 1.1 Histograma para los niveles de intensidad de una imagen, con I= 64.

Si bien el histograma de la figura anterior no representa a una imagen real, resulta posible a partir del mismo, deducir alguna información hipotética acerca de lo que sería la imagen. Por ejemplo, la imagen tiene 64 niveles de gris, sin embargo, tal

rango no es utilizado de manera completa, pues no se tienen pixeles con valores superiores a 50.

1.2.2 Realce de imágenes por modificación del contraste

Una de las imperfecciones más comunes de las imágenes digitales, es el pobre contraste resultante de un rango de intensidad reducido en comparación al rango disponible de niveles de gris (por ejemplo de 0 a 255 niveles). El contraste de una imagen, puede mejorarse mediante el re-escalamiento de la intensidad de cada pixel. Según este método, el nivel de gris correspondiente a un pixel en la imagen de entrada y que denotaremos por **i**, se modifica de acuerdo a una transformación específica. Tal transformación g=T(i), relaciona la intensidad de entrada **i**, con la intensidad de salida **g** y usualmente se representa mediante un dibujo o una tabla. A manera de ejemplo, la figura 1.2a muestra una imagen de 4 x 4 pixeles, donde cada pixel se ha representado con 3 bits, de modo que en total sería posible representar 8 niveles de gris. La transformación que relaciona la intensidad de entrada con la intensidad de salida, se muestra en la figura 1.2b. De acuerdo a tal transformación, para cada pixel de la imagen de entrada, se obtiene la correspondiente intensidad en la imagen de salida. El resultado obtenido en este caso particular se muestra en la figura 1.2c, en donde podemos observar que el contraste entre las zonas obscuras y claras dentro de la imagen, se incrementa apreciablemente. Eligiendo apropiadamente la transformación específica, puede modificarse de manera casi arbitraria el contraste y rango dinámico de la imagen. En general, los programas de procesamiento de imágenes permiten al usuario definir de manera interactiva la

función de transformación, operando sobre un gráfico como el de la figura 1.2b para establecer tal función.

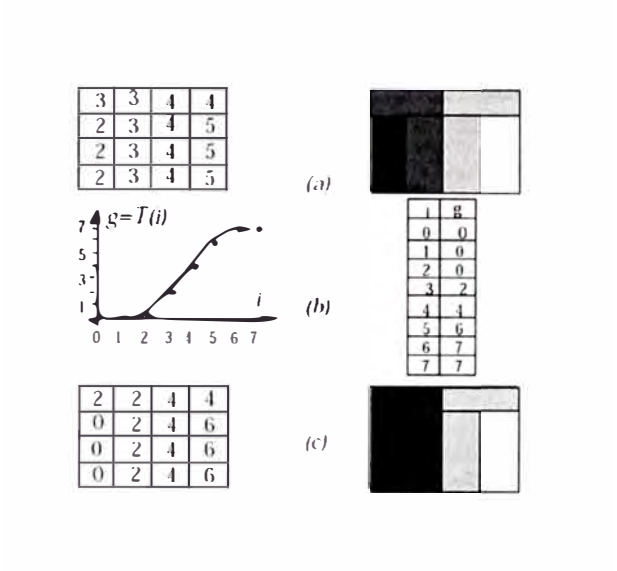


Figura 1.2 Ejemplo de la modificación de la escala de grises:

(a) Imagen de 4 x 4 pixeles, con cada pixel representado por 3 bits

(b) Función de transformación de los niveles de gris

(c) Resultado de modificar la imagen en a), usando la transformación de niveles de gris especificada en b).

Algunas Transformaciones de uso frecuente

a) Negativo de la Imagen

Las imágenes en negativo, son parecidas a los negativos fotográficos y son muy fáciles de producir mediante el uso de tablas de búsqueda. La idea es convertir aquellas porciones de la imagen que son claras en oscuras y las que son oscuras en claras. En la figura 1.4, se muestra una transformación que tiene tal efecto y que es equivalente a inicializar la tabla de búsqueda, con valores que son el resultado de restar el valor del pixel de entrada del máximo valor posible del pixel (L-1). La

negación de la imagen, puede resultar de utilidad cuando se quiere apreciar los detalles en las porciones brillantes de una imagen, pues el ojo humano, es más capaz de discernir los detalles en áreas oscuras de una imagen que en las áreas más brillantes. En la figura 1.6 se muestra un ejemplo de esta técnica al desplegar una imagen angiográfica. En la figura 1.6a se muestra la imagen original con su correspondiente histograma, mientras que en la figura 1.6b se presenta el negativo de la imagen asi como su correspondiente histograma, el cual corresponde a una figura especular del histograma de la imagen original.

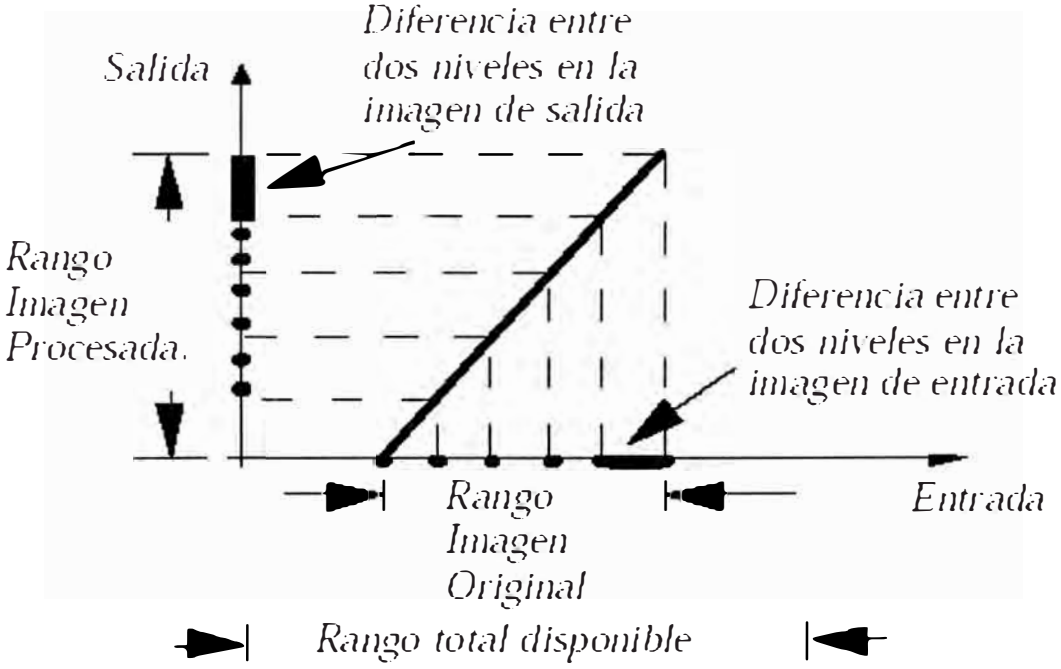


Figura 1.3 Efecto de la modificación del contraste de una imagen, cuyo rango original es pequeño en relación al rango total disponible.

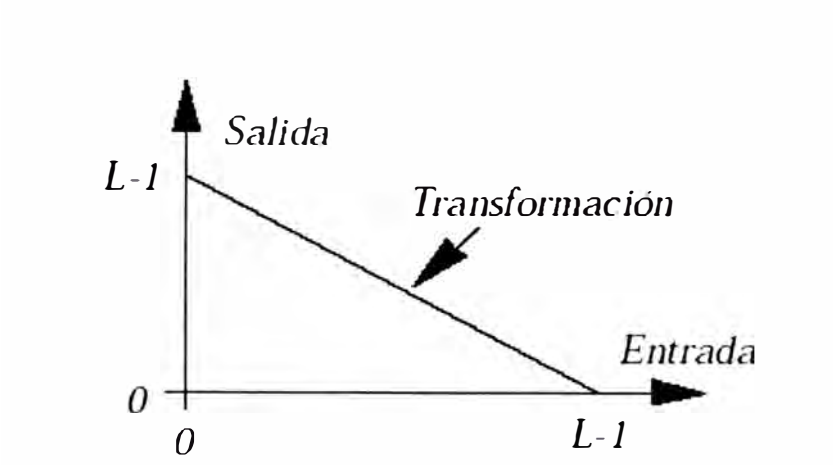


Figura 1.4 Transformación utilizada para obtener el negativo de una imagen.

b) Control del brillo de una imagen

En ciertas ocasiones, la apariencia de una imagen puede realzarse visualmente ajustando el brillo de la misma. Esto se logra sumando o restando un valor constante a cada pixel de la imagen de entrada. El efecto de tal transformación sobre el histograma de la imagen, es desplazarlo hacia la derecha (zona más brillante), en caso de que se sume un valor constante o por el contrario, lo desplaza hacia la izquierda (zona mas oscura) cuando se resta un valor constante. En la figura 1.6 se muestra un ejemplo de esta técnica, al modificar la imagen angiográfica de la figura 1.6a aumentando su brillo, lo cual se traduce en una imagen con tonalidades más claras mostrada en la figura 1.6c. Por su parte su histograma se desplaza hacia los valores de mayor intensidad.

c) Binarización de imágenes

La binarización es una técnica que permite convertir imágenes con niveles de gris, en una imagen binaria (blanco y negro). De acuerdo a tal técnica, los valores de pixel en la imagen de entrada que son menores a un cierto umbral pre-especificado, son convertidos a negro, mientras que los pixeles con valores mayores al umbral, son convertidos a blanco. En la figura 1.5a se muestra la transformación que permite realizar la binarización. En algunas ocasiones se desea realizar una binarización tal que a una banda especificada por dos umbrales, se les asigne el color blanco, mientras que los pixeles de la imagen de entrada cuyos valores están fuera de la banda especificada, se les asigne el color negro. Esta transformación se muestra en la figura 1.5b, por su parte en la figura 1.7b se muestra el resultado obtenido al binarizar la imagen angiográfica mostrada en la figura 1.7a, utilizando la transformación especificada en la figura 1.5a con un umbral arbitrario de valor 128.

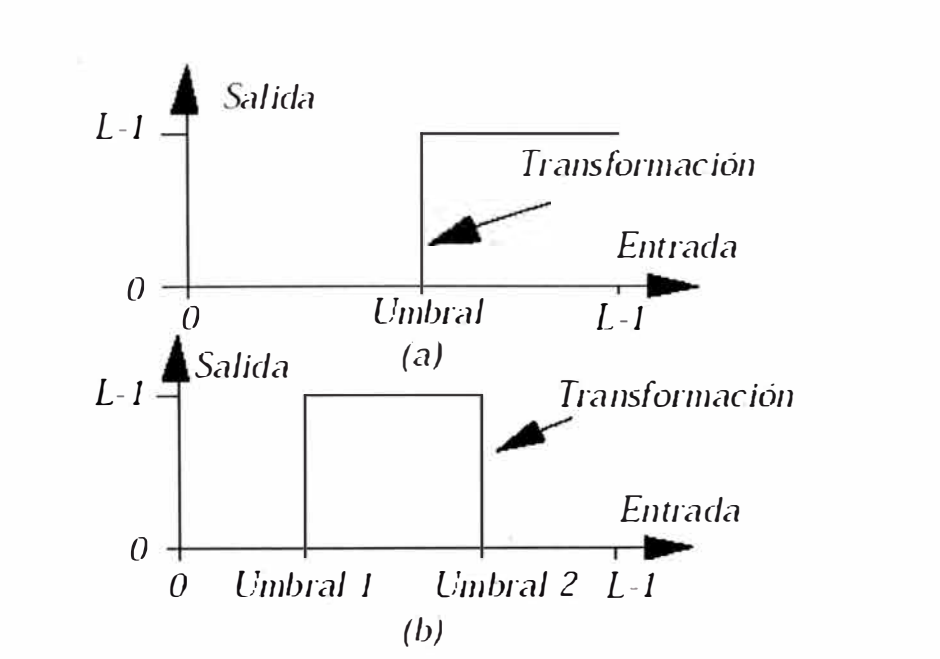


Figura 1.5 Transformaciones utilizadas para binarizar una imagen:

(a) transformación para la binarización ; (b) binarización de una banda.

d) Ampliación del contraste

A esta técnica también se le conoce como dilatación del histograma (histogram stretching). La misma combina el uso del histograma con la utilización de las tablas de búsqueda o LUT's, la razón para ello es que el histograma constituye una herramienta ideal para examinar el contraste de una imagen.

Para ampliar el contraste, se realiza en el histograma una búsqueda desde los valores más pequeños de niveles de gris, hacia el máximo valor. Cuando se consiga que el número de pixeles correspondiente a un nivel de gris dado, supera un cierto umbral pre-establecido, se habrá determinado el umbral inferior (umbral 1), que estará especificado por el nivel de gris para el cual ocurre el evento mencionado. A continuación, se realiza una búsqueda en el histograma desde el valor más elevado de nivel de gris, hacia los valores más pequeños. Cuando el número de pixeles para un nivel de gris dado, supere el umbral pre-establecido, se habrá determinado el umbral superior (umbral 2) en la escala de niveles de gris. Una vez determinados los umbrales 1 y 2, se procesa la imagen mediante una transformación tal que a los pixeles de la imagen cuyo valor es inferior al umbral 1, se les asigna el valor de cero, por otra parte, si los pixeles de la imagen de entrada son superiores al valor del umbral 2, entonces se les asigna el máximo valor de gris (L-1). Por su parte, los pixeles comprendidos entre los dos umbrales son escalados de manera lineal. En la figura 1.6 se muestra la obtención de los umbrales 1 y 2 así como la función de transformación que se genera a partir de los mismos.

El resultado de la ampliación del contraste será una imagen que utiliza más apropiadamente todo el rango disponible de niveles de gris y como consecuencia de

ello, tendrá una apariencia más balanceada. En la figura 1.7a se muestra una imagen angiográfica así como su correspondiente histograma. En la figura 1.7c se muestra la imagen luego de ser sometida al realce de contraste así como su histograma, se puede observar que el histograma ha sido expandido para ocupar todo el rango disponible, también se aprecia que la imagen de salida presenta mayor contraste y en consecuencia resulta fácil percibir todas las estructuras que la componen.

1.2.3 Perfil en una imagen

En muchas ocasiones, cuando se procesa una imagen de manera interactiva, resulta muy conveniente, representar gráficamente el valor de los pixeles que componen por ejemplo una línea o una columna o de manera general, según una recta que tenga dirección arbitraria. Tal perfil permite determinar información importante referente a la imagen, como por ejemplo los valores máximos y mínimos de nivel de gris, según la dirección elegida así como información referente al nivel de ruido y a la naturaleza de los contornos presentes en la misma. Tal información si bien no es global, sino asociada a la dirección elegida, puede ser importante y ayudar, por ejemplo, a decidir el valor de un umbral a utilizar durante el proceso de segmentación o puede orientar en referencia al tipo de procesamiento que es necesario desarrollar sobre la imagen objeto de estudio. Generalmente, esta función se implanta de manera que el usuario al utilizar algún dispositivo de entrada/salida como por ejemplo el ratón, pueda fijar de manera interactiva la dirección según la cual se desea graficar el perfil de la imagen. El programa en este caso, permite fijar de manera arbitraria la posición del perfil, para ello, el usuario puede dibujar un

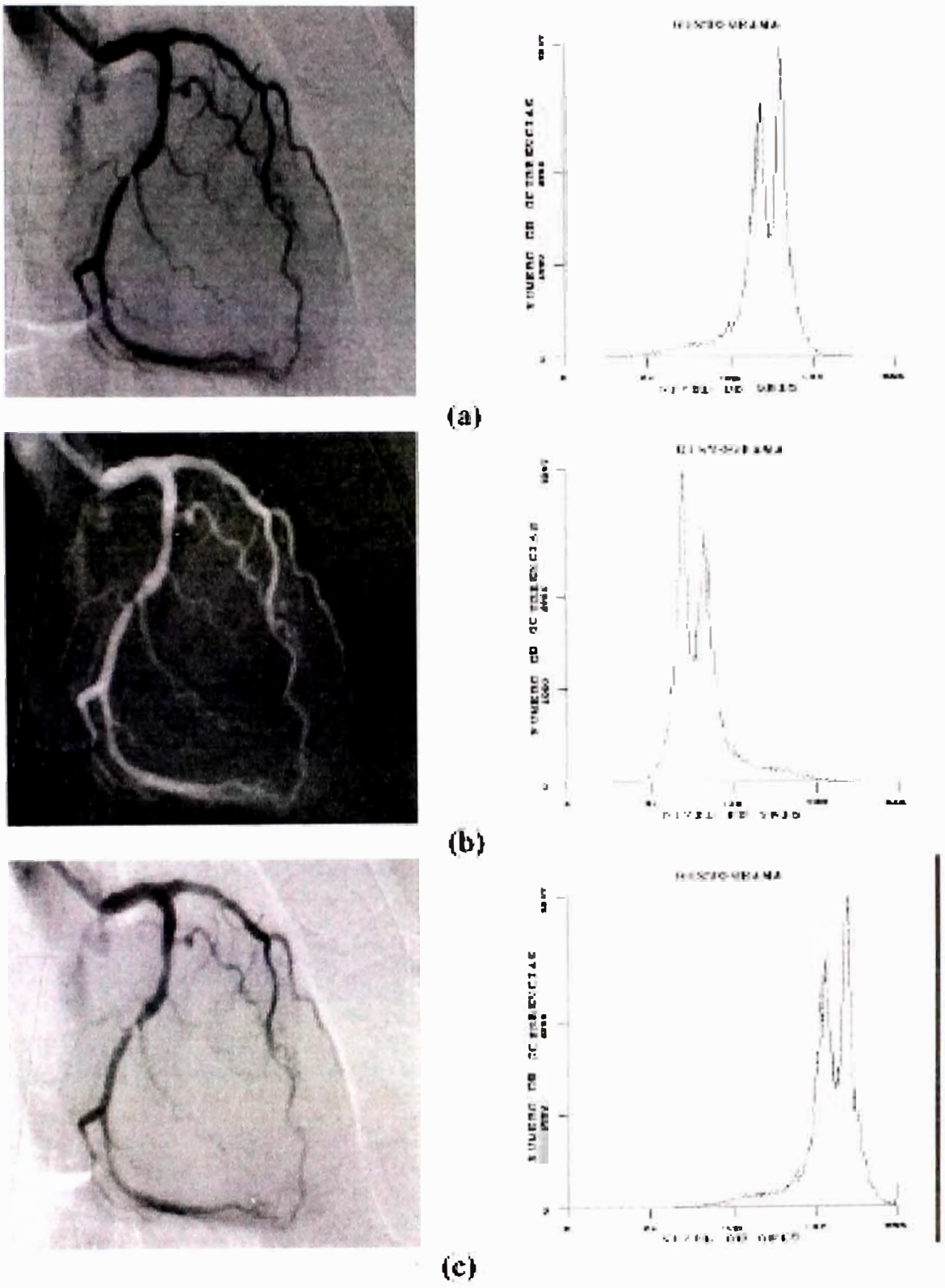


Figura 1.6 Ilustración de las técnicas de realce (a) imagen original con su histograma, (b) negativo de la imagen (c) cambio en el nivel de brillo.

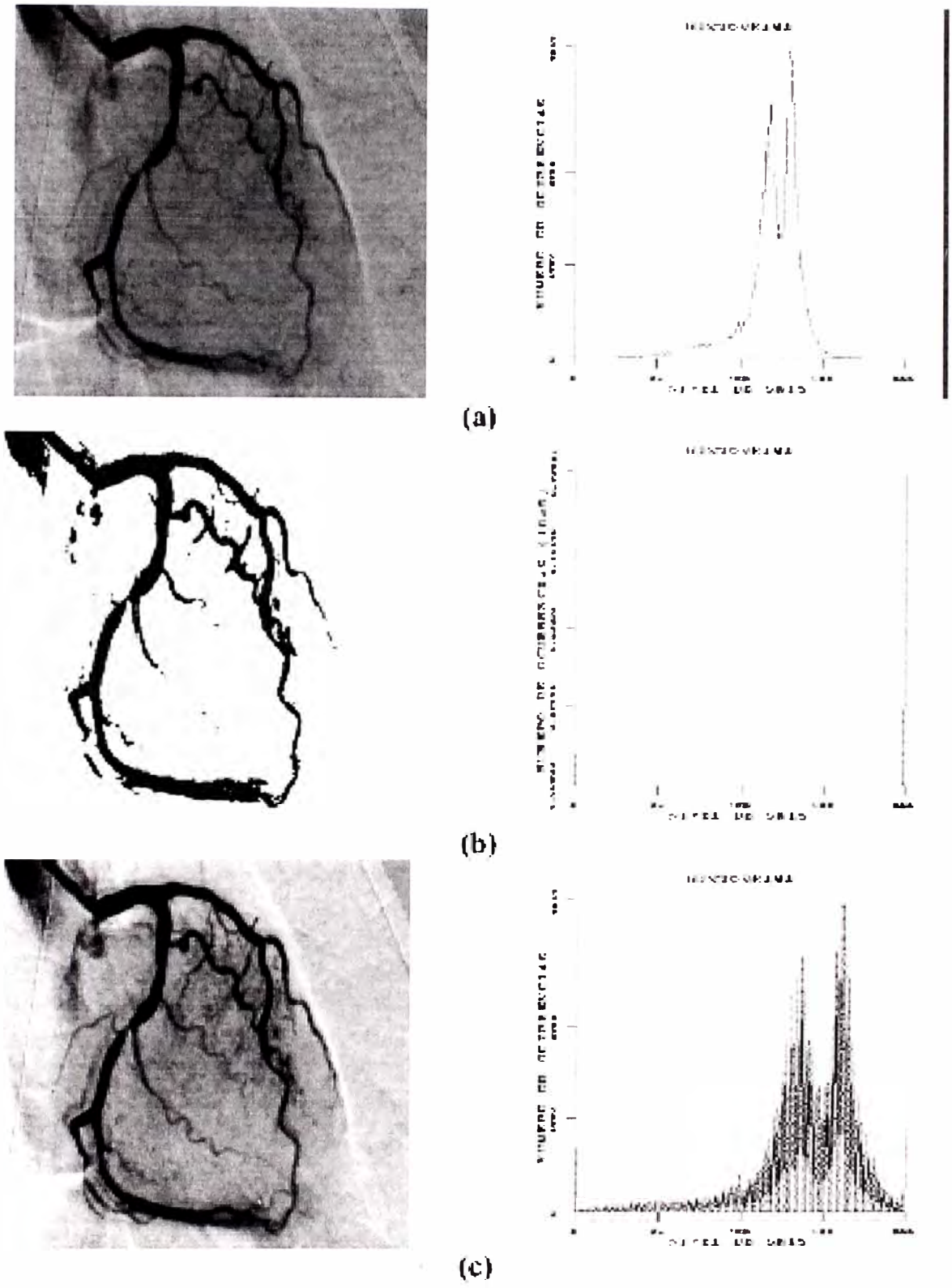


Figura 1.7 Ejemplos de técnicas de realce de imágenes (a) imagen original,

(b) binarización de la imagen (c) realce de contraste.

conjunto de puntos sobre la imagen, los cuales son seguidamente unidos por rectas, finalmente el perfil se dibuja para la trayectoria elegida (Udupa y otros, 1993).

1.2.4 Técnicas de colores falsos y seudocolor

Es bien conocido que el ojo humano, es bastante sensible al color, así el número de niveles de gris que puede discriminarse como tal, es bastante más pequeño que el número de colores. Por otra parte, las imágenes a color, son más agradables a la vista que las imágenes en blanco y negro.

La técnica de colores falsos, usualmente se emplea cuando se desea asociar a un conjunto de datos, un conjunto de colores para distinguir en los mismos, ciertos atributos, tal como ocurre cuando un sensor remoto adquiere información en la banda de infrarrojo (la cual no es visible), en tal caso, lo que se hace es asociar el color a este conjunto de datos, para apreciar mejor los detalles.

La técnica de seudocolor consiste en transformar una imagen monocromática (en niveles de gris) en una imagen a color, al asignar a cada pixel un color basado por ejemplo en su intensidad. Un método pudiera ser el siguiente: se procesa la imagen monocromática con tres filtros, una pasa bajo, uno pasa banda y uno pasa alto. La imagen procesada con el filtro pasa bajo se asigna al color azul, la imagen procesada con el filtro pasa banda se asocia al verde y la procesada con el filtro pasa alto se asocia al rojo; luego estas tres imágenes se combinan para producir una imagen a color. Usando técnicas de seudocolor, más complejas que la mencionada previamente y mediante mucha intervención humana, ha sido posible colorizar películas filmadas originalmente en blanco y negro.

1.3 Procesamiento basado en una región de la imagen

Las técnicas de procesamiento basadas en una región tienen muchas aplicaciones en la obtención de primitivas características de la ımagen como por ejemplo la extracción de contornos, para realzar los contornos, para suavızar una ımagen, para introducir borrosidad dentro de la misma y para atenuar el ruido aleatorio. Usan un grupo de pixeles dentro de la imagen a procesar, con el propósito de extraer información acerca de la misma. El grupo de pixeles que se estudia en este caso, se denomina usualmente vecindad. Por lo general la vecindad es una matriz bidimensional de valores de pixeles con un número impar de filas y columnas. El pixel de interés que normalmente es reemplazado por un nuevo valor, producto de la aplicación de un algoritmo, se ubica por lo general, en el centro de la vecindad.

Al utilizar una vecindad en el procesamiento, se puede aprovechar la información acerca del comportamiento regional de la imagen en cuestión, mejor conocida como frecuencia espacial, la cual podría definirse como la tasa de cambio de la intensidad de los pixeles dividido por la distancia sobre la cual ocurre el cambio. La frecuencia espacial tiene componentes en las direcciones horizontal y vertical dentro de la imagen. Por ejemplo, la imagen de un patrón tipo tablero de ajedrez presentará un alto contenido de frecuencia espacial, el cual aumentará en la medida que el tamaño de los cuadros disminuya. Por su parte una imagen con un bajo contenido de frecuencia espacial por lo general tiene amplias áreas con valores casi constantes de los pixeles.

Muchas de las técnicas de procesamiento basadas en una región de la imagen, al tener acceso a la información referente a la frecuencia espacial, pueden actuar

como filtros que atenúan o realzan ciertas componentes de la frecuencia espacial contenidas dentro de la imagen.

En la implantación de estas técnicas de procesamiento regional, se utilizan métodos lineales tales como la convolución o no lineales como el filtraje de mediana. En todo caso, el procedimiento que se sigue es el siguiente:

a) Se realiza una sola pasada sobre la imagen de entrada realizando un barrido pixel por pixel, según las filas y columnas.

b) Cada pixel de la imagen de entrada es procesado, considerando una vecindad del mismo y utilizando un algoritmo apropiado.

c) El nuevo valor del pixel, obtenido de acuerdo a lo especificado en b), es ubicado en la imagen de salida, ocupando la misma posición que ocupaba en la imagen de entrada.

El hecho de considerar los pixeles de una vecindad, hace que las técnicas de procesamiento basadas en una región tengan un mayor costo de cálculo numérico que las técnicas basadas en un solo punto. Este costo dependerá del tamaño de la vecindad a considerar, así como del tipo de representación numérica utilizada. Sin embargo, para la mayoría de las aplicaciones y con las computadoras disponibles actualmente, se pueden obtener muy buenos resultados en términos de tiempo de cálculo, al procesar imágenes de un tamaño mediano (256 x 256 ó 512 x 512).

1.3.1 Convolución

En procesamiento de imágenes, la convolución corresponde a la extensión del caso unidimensional, mediante la cual una señal cualquiera podía ser procesada con

un filtro arbitrario con una respuesta impulsiva conocida, para conocer los detalles de su implantación en el caso unidimensional así como sus propiedades, el lector puede revisar (Oppenheim y Schafer, 1975). Lim (1990) discute la implantación, criterios de validez y propiedades para el caso bidimensional que es el que nos concierne directamente.

Si consideramos una imagen como un arreglo bidimensional denotado por **x(i,j)** y el filtro (núcleo o máscara de convolución) con respuesta impulsiva **h(i,j)**, su convolución produce una imagen de salida **y(i,j)**, de acuerdo a la siguiente ecuación, en donde **m,n** definen la vecindad a considerar de acuerdo al tamaño del núcleo de convolución **h(i,j)**.

$$y\ (i,j) = \sum_{m\ =\ -K1}^{k2} \sum_{n\ =\ -L1}^{L2} h\ (m,n)\ \ x\ \ (i-m, j-n) \qquad (1)$$

La implantación de esta ecuación de convolución se hace de manera directa cuando el tamaño del filtro o máscara de convolución es pequeño (usualmente menor a 9 x 9 pixeles), pues en tales casos el costo computacional no es exagerado, sin embargo, cuando se tienen filtros de mayor tamaño, lo más recomendable es implantar esta ecuación de convolución mediante la utilización de la transformada rápida de Fourier, en Lim (1990) se presenta en detalle la implantación de la convolución bidimensional mediante la aplicación de algoritmos rápidos de transformada de Fourier.

Para la implantación directa de la ecuación de convolución, asumimos que la máscara de convolución es una matriz de tamaño (N1 x N2), usualmente de 3 x 3 pixeles, la cual adicionalmente define el tamaño de la vecindad dentro de la imagen de manera que sea del mismo tamaño que la máscara. En la Figura 1.8 se ilustra el

proceso de convolución, según el cual, para un pixel dado dentro de la imagen de entrada x(i,j), cada pixel de la vecindad es multiplicado por el pixel correspondiente en la máscara de convolución, así mismo cada uno de estos productos es sumado, de manera que el nuevo valor del pixel en la imagen de salida y(i,j) estará dado por la suma de todos estos productos. El procesamiento de toda la imagen se realiza desplazando la máscara y repitiendo para cada punto el mismo procedimiento.

La máscara de convolución tiene por lo general un número impar de filas y columnas y su tamaño frecuentemente es 3 x 3, su contenido depende del tipo de procesamiento que se desea implantar. Ejemplos de diferentes tipos de máscaras de convolución son presentados en (Lindley, 1991) y (Pratt, 1978).

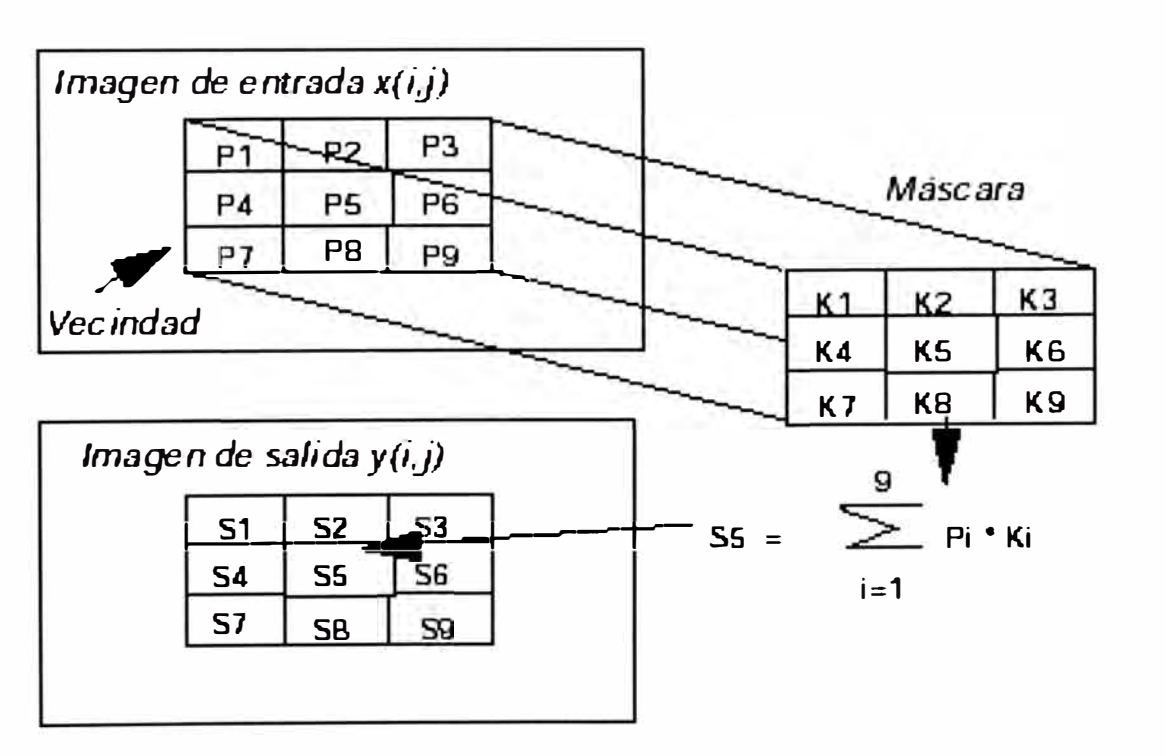


Figura 1.8 Ilustración del proceso de convolución con una máscara. Cada pixel en la imagen de salida es el resultado de la suma de los productos entre los pixeles de la máscara y los pixeles incluidos en la vecindadcorrespondiente en la imagen de entrada.

a) Filtraje espacial pasa-bajo

Los filtros espaciales pasa bajo, dejan el contenido de baja frecuencia inalterado mientras que atenúan los contenidos de alta frecuencia, este tipo de filtros resulta adecuado para atenuar ruido aditivo aleatorio presente en la imagen. En la figura 1.9 se muestran tres máscaras de convolución frecuentemente utilizadas para realizar el filtraje pasabajo, una de las propiedades de tales máscaras es que la suma de todos sus valores debe ser igual a la unidad. Uno de los efectos que es necesario tener presente cuando se aplica este tipo de filtros, es que los mismos pueden introducir apreciable borrosidad en la imagen En la figura 1.10 se ilustra este tipo de procesamiento, así en la figura 1.10a se muestra la imagen original la cual ha sido contaminada con ruido aleatorio tal como se muestra en la figura 1.10b, por su parte en la figura 1.10c se muestra el resultado luego de procesar con un filtro pasa-bajo en donde se observa que si bien el contenido de ruido disminuye, sin embargo la borrosidad de la imagen aumenta.

$$\frac{1}{9}\cdot\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix}\qquad \frac{1}{10}\cdot\begin{bmatrix}1&1&1\\1&2&1\\1&1&1\end{bmatrix}\qquad \frac{1}{16}\cdot\begin{bmatrix}1&2&1\\2&4&2\\1&2&1\end{bmatrix}$$

Figura 1.9 Tres máscaras que permiten el filtraje pasa-bajo de una imagen.

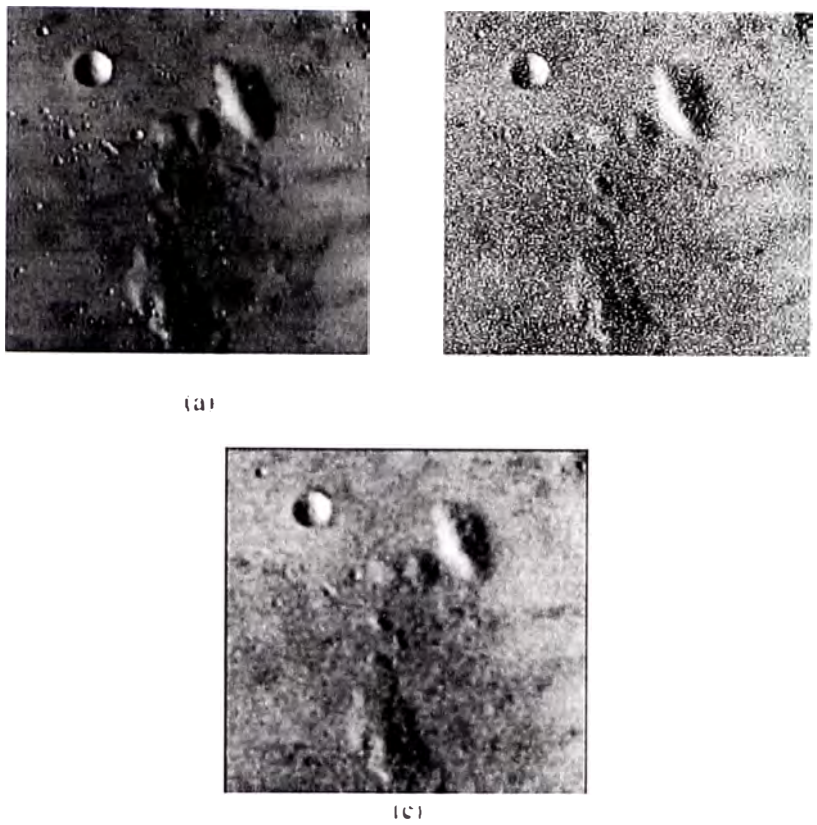


Figura 1.10 Ejemplo de filtro promediador:

(a) Imagen original. (b) Imagen contaminada con ruido. (c) Imagen procesada con un filtro promediador.

b) Filtraje pasa-alto

Los filtros pasa alto, tienen la propiedad de acentuar los detalles de alta frecuencia de una imagen, normalmente los filtros pasa alto se utilizan cuando se quiere examinar objetos con alto contenido de frecuencia espacial, como consecuencia de tal procesamiento, las porciones de una imagen que presentan componentes de alta frecuencia, serán resaltadas mediante la utilización de niveles de gris más claros, mientras que aquellas con componentes de baja frecuencia serán más oscuras, en este sentido, este tipo de filtro puede ser utilizado para reforzar los bordes presentes en la imagen. Uno de los efectos indeseados de estos filtros es que pueden

acentuar el ruido de la imagen. En la figura 1.11 se muestran tres máscaras de convolución para implantar el filtro pasa alto, por su parte en la figura 1.12 se muestra el resultado de procesar una imagen con un filtro pasa alto, así en la figura 1.12a se muestra la figura original que al ser procesada utilizando la máscara de convolución mostrada en la figura 1.12a, produce como resultado la imagen mostrada en la figura 1.12b en donde se observa que efectivamente los bordes aparecen reforzados.

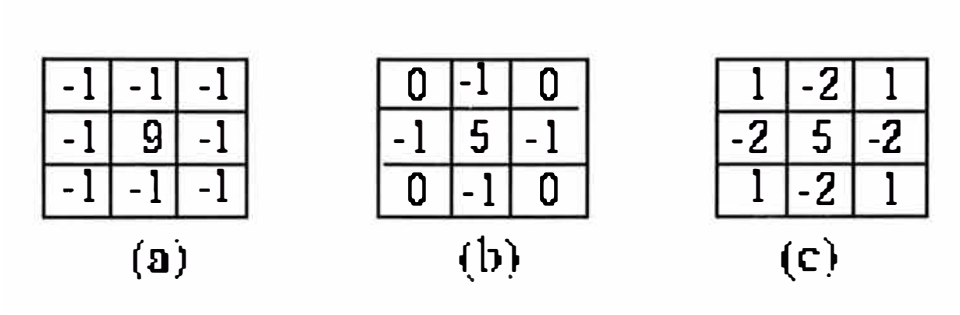


Figura 1.11 Tres máscaras que permiten implantar el filtraje pasa-alto.

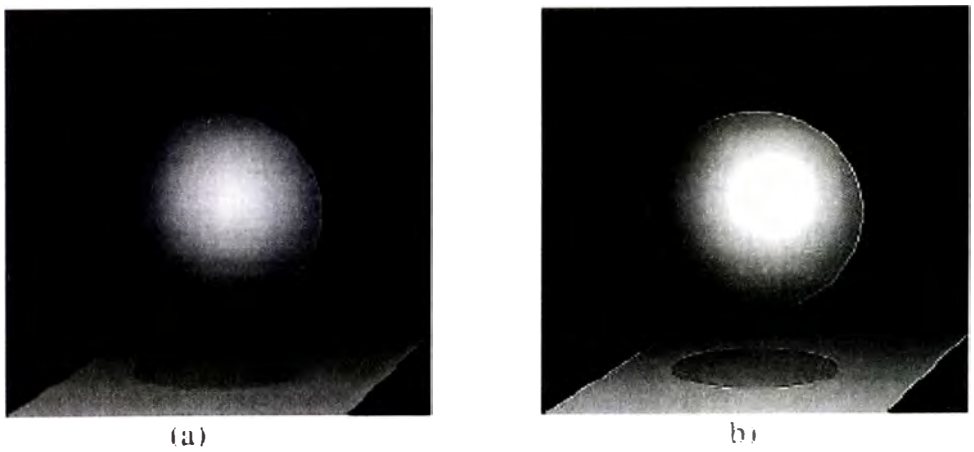


Figura 1. 12 Procesamiento de una imagen con un filtro pasa alto:

(a) imagen original

(b) imagen luego de procesar con la máscara mostrada en la figura 1.11a.

1.3.2 Filtraje no-lineal de la imagen

a) Filtro promediador

En la figura 1.13, se describe un algoritmo sencillo para realizar el suavizamiento o filtraje pasa bajo de una imagen contaminada con ruido aleatorio. Según esta técnica, se examina secuencialmente cada pixel, y si la magnitud del mismo, es mayor que el nivel de gris promedio de sus vecinos más un cierto umbral e, tal pixel se sustituye por el valor promedio, en caso contrario, se deja tal valor inalterado. El tamaño de la vecindad a considerar, debe elegirse cuidadosamente, pues en caso de ser muy grande puede introducir borrosidad apreciable. El valor del umbral pudiera determinarse a partir de información del ruido que contamina a la imagen tal como la varianza del mismo.

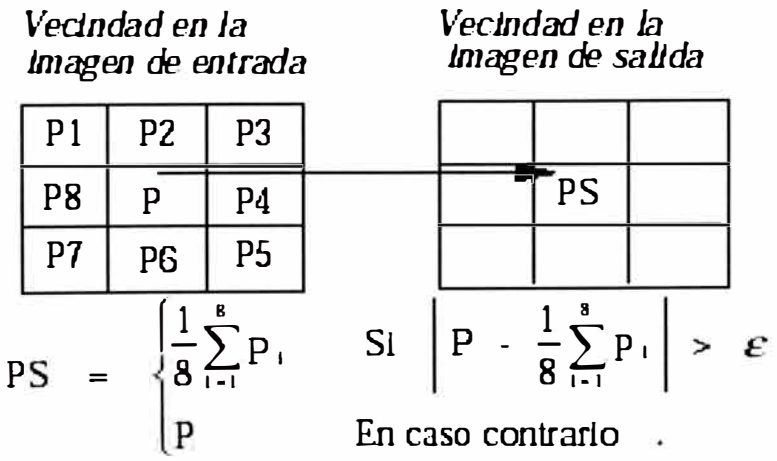


Figura 1.13 Ejemplo de un algoritmo de suavizamiento utilizando el filtro promediador.

b) Filtro de Mediana

El filtraje de mediana, es un procedimiento no-lineal, útil para reducir el ruido impulsivo y del tipo "sal y pimienta", muchas veces presente en las imágenes. El filtro de mediana utiliza los valores de los pixeles contenidos en una vecindad de tamaño impar, para determinar el nuevo valor del pixel de interés. El procedimiento para ello, consiste en clasificar todos los pixeles incluidos en la ventana en orden creciente y sustituir el pixel ubicado en el centro de la vecindad por el pixel mediano luego de la clasificación, es decir, si tenemos una secuencia discreta de tamaño N impar, entonces la mediana de tal secuencia, es aquel miembro de la secuencia, para el cual, (N-1)/2 elementos son más pequeños o a lo sumo iguales y (N-1)/2 elementos son más grandes.

En la figura 1.14 se muestra un ejemplo de la implantación del filtro de mediana. El costo computacional más importante corresponde a la clasificación del arreglo en orden creciente, por lo cual es muy importante utilizar algoritmos optimizados para realizar tal tarea. En (Press y otros, 1992) se describen varios algoritmos para realizar eficientemente tal clasificación.

Una de las propiedades del filtro de mediana, es que el mismo tiende a preservar en lo posible la información referente a los contornos de la imagen, atenuando por el contrario, variaciones impulsivas aleatorias, por lo cual se utiliza frecuentemente. En la figura 1.15 se ilustra el filtraje de mediana así en la figura 1.15a se muestra la imagen original, en la figura 1.15b la imagen contaminada con ruido impulsivo y en la figura 1.15c el resultado luego de procesar con un filtro de mediana de 5 x 5.

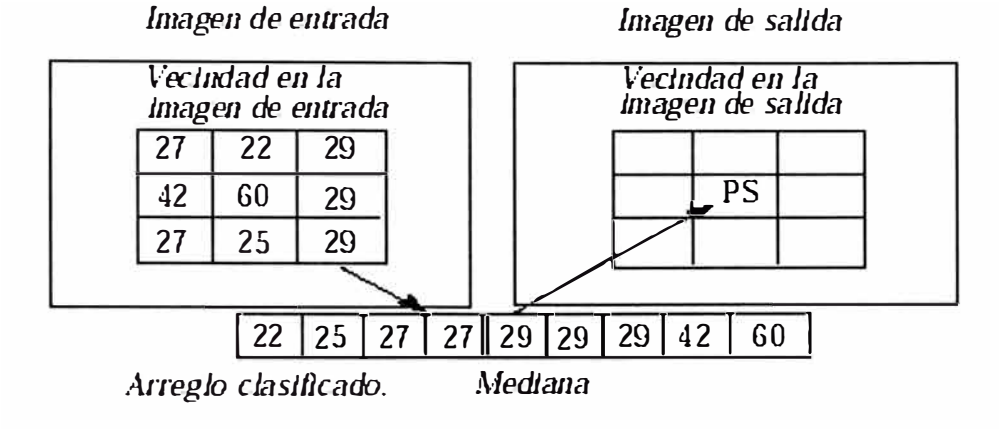


Figura 1.14 Ilustración del procedimiento para implantar el filtro de mediana.

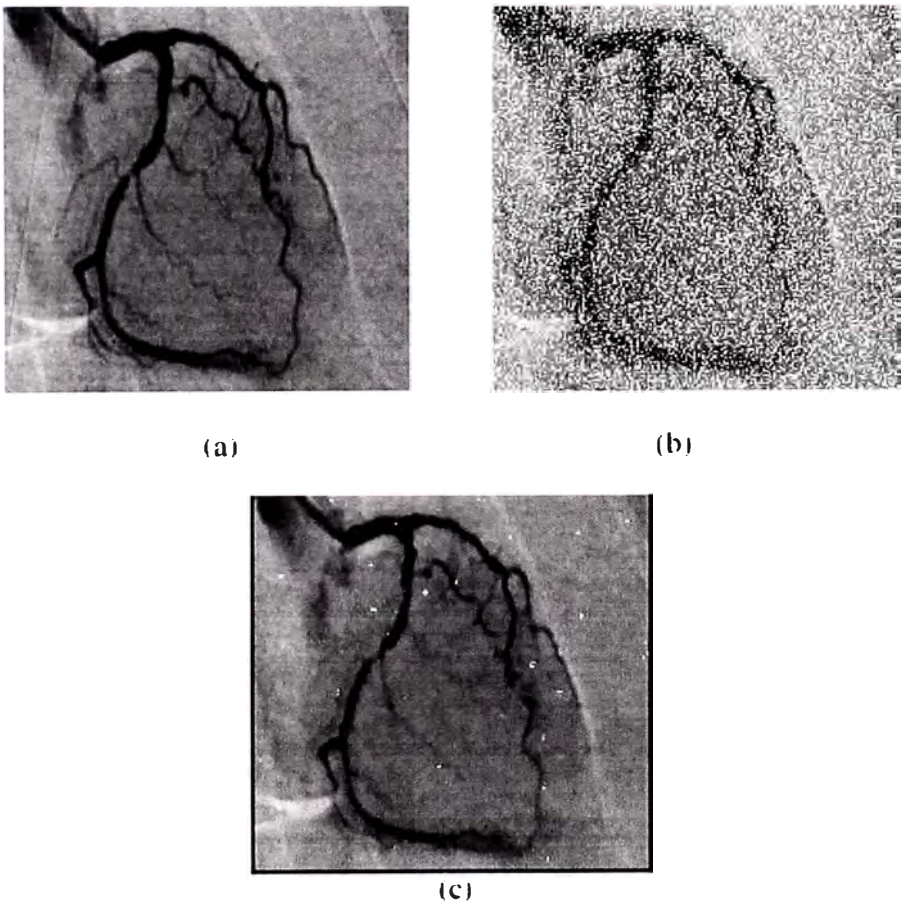


Figura 1.15 Ejemplo de filtro de mediana: (a) imagen original (b) imagen contaminada con ruido impulsivo (c) imagen procesada con filtro de mediana.

1.3.3 Detección de contorno

Las técnicas de detección de contornos son útiles en diferentes contextos, en particular la detección de contornos es una de las etapas del proceso de segmentación cuyo objeto es particionar la imagen en regiones asociadas a los diferentes elementos que componen la escena, y que puede ser utilizada posteriormente para el análisis automático de los mismos mediante algoritmos de reconocimiento de formas (González, 1987).

Un borde en una imagen, es un limite o contorno en el cual ocurren cambios significativos en algún parámetro físico de la imagen, tal como la reflectancia superficial, la iluminación o la distancia de la superficie visible al observador. Los cambios en los parámetros físicos de la imagen se manifiestan de diversas formas, incluyendo cambios en intensidad, color y textura; sin embargo, en este capitulo, nos limitaremos a estudiar únicamente los cambios en la intensidad de la imagen. En la figura 1.16, se muestran los diagramas esquemáticos de bordes unidimensionales y bidimensionales, modelados usualmente como un incremento en rampa, en los niveles de gris de la imagen, en este caso de un nivel de gris bajo a un nivel de gris alto, siendo valida la definición para el caso contrario. En el caso bidimensional, cada punto **(x,y)** define la posición del pixel y la coordenada **z** define la amplitud del nivel de gris.

Para el caso unidimensional, el borde se caracteriza por la altura, la pendiente y la coordenada x_0 del punto medio de la pendiente, entonces se puede decir que existe un borde si tanto el ángulo de la pendiente como la altura superan un cierto valor umbral. Para el caso bidimensional, también es importante la orientación con

respecto al eje x, en cualquier caso lo deseable es que el detector de contorno produzca un pixel indicador del contorno, ubicado en el punto medio de la pendiente.

En la figura 1.17 se muestra un enfoque clásico para la detección de bordes, según el cual la imagen, se somete a una acentuación de contornos, seguida por un detector de borde por umbral. Si denotamos como ***x(i,j)*** la imagen de entrada, ***G(i,j)*** la imagen luego de la acentuación de bordes, U_b el umbral para bordes de bajo a alto y U_a el umbral para bordes de alto hacia bajo, entonces podemos decir que tenemos un borde en sentido positivo si:

$$G(i,j) \geq U_b \tag{2}$$

o un borde en sentido negativo si:

$$G(i,j) < U_a \tag{3}$$

La selección del valor umbral, es uno de los aspectos importantes en detección de bordes. Un nivel de umbral muy elevado, no permitirá la detección de elementos estructurales de la imagen si estos no tienen suficiente amplitud, del mismo modo, un umbral de muy poca amplitud causará que el ruido se detecte falsamente como bordes en la imagen.

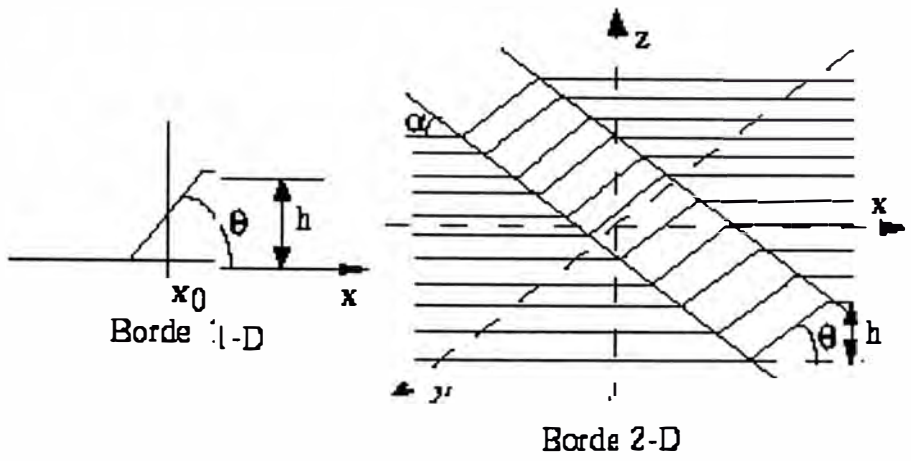


Figura 1.16 Modelo para el borde unidimensional y bidimensional.

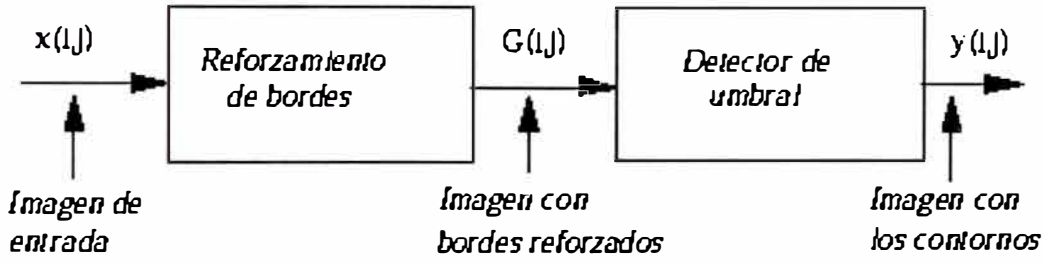


Figura 1.17 Sistema utilizado para la detección de contornos.

a) Método para el reforzamiento de bordes

Estas técnicas tienden a enfatizar los bordes de los componentes de la escena, mientras atenúan los valores de nivel de gris de las regiones casi constantes en la imagen. Generalmente, para realizar el reforzamiento de los bordes, se utilizan técnicas de gradiente.

b) Detección de contornos basados en gradientes

Esta técnica se basa en el gradiente o la derivada de la señal. En la figura 1.18 se analiza la aplicación de esta técnica para detectar un borde unidimensional, en una función ***f(x)*** (figura 1.18a). En tal caso el punto x_0 corresponde a la ubicación del borde y una manera de determinar tal valor, es mediante el uso de la primera derivada ***f'(x)*** tal como se muestra en la figura 1.18b o de la segunda derivada tal como se muestra en la figura 1.18c, en tales figuras, el valor de x_0 puede determinarse buscando los extremos (máximo o mínimo) en la primera derivada ó la posición del cruce por cero cuando la segunda derivada cambia de signo.

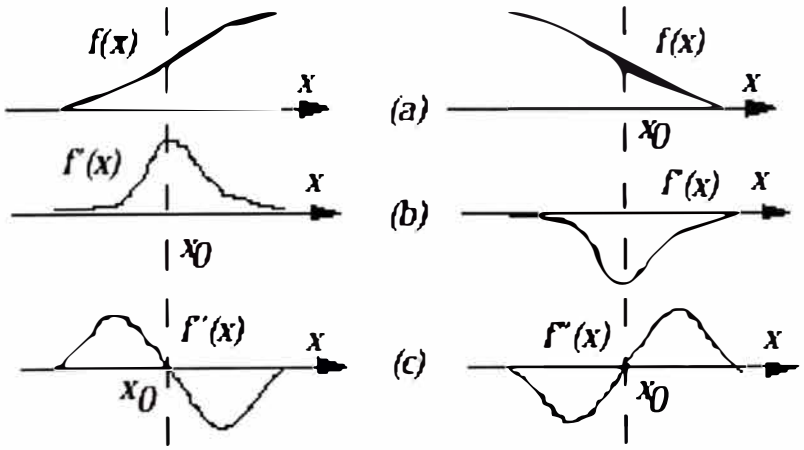


Figura 1.18 Uso de la derivada para la detección de bordes-caso unidimensional.

La generalización del esquema mostrado previamente, al caso bidimensional, conduce a sustituir la derivada por el gradiente de la función ***f(x,y)*** dado por el gradiente:

$$\nabla f(x,y) = \frac{\partial f(x,y)}{\partial x}\bar{i}_x + \frac{\partial f(x,y)}{\partial y}\bar{i}_y \qquad (4)$$

donde **i** es un vector unitario en la dirección de **x** e **i** es un vector unitario en la dirección de **y**. En la figura 1.21 se muestra un ejemplo de realce de bordes basado en el gradiente así en la figura 1.21a se muestra una imagen angiográfica a partir de la cual se obtiene la magnitud del gradiente, mostrado en la figura 1.21b, tal imagen puede servir de base para determinar los puntos del contorno mediante una selección adecuada.

En el caso de imágenes discretas, lo usual es hacer aproximaciones en el cálculo de la derivada, basadas en diferencias según diversas direcciones. En tal caso, el gradiente podrá implantarse en base a la convolución con la siguiente máscara:

$$\begin{vmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{vmatrix} \qquad (5)$$

Siguiendo este procedimiento, podemos derivar las máscaras de convolución para implantar el gradiente direccional según diferentes direcciones las cuales son a menudo denominadas según la dirección del borde que pueden detectar y son: "Norte", "Noreste", "Este", "Sudeste", "Sur", "Sudoeste", "Oeste", y "Noroeste". Las correspondientes máscaras aparecen reseñadas en (Pratt, 1978) y (Lindley, 1991).

c) Detección de contorno basado en la Laplaciana

Otra posibilidad para la acentuación de bordes, consiste en usar la segunda derivada, en tal caso se usan los puntos de cruce por cero para realizar la detección de borde. En el caso de la imagen por ser una señal bidimensional, la extensión de la segunda derivada unidimensional corresponderá en este caso la Laplaciana, cuya expresión es:

$$\nabla^2 f(x,y) = \nabla(\nabla f(x,y)) = \frac{\partial^2 f(x,y)}{\partial x^2} + \frac{\partial^2 f(x,y)}{\partial y^2} \tag{6}$$

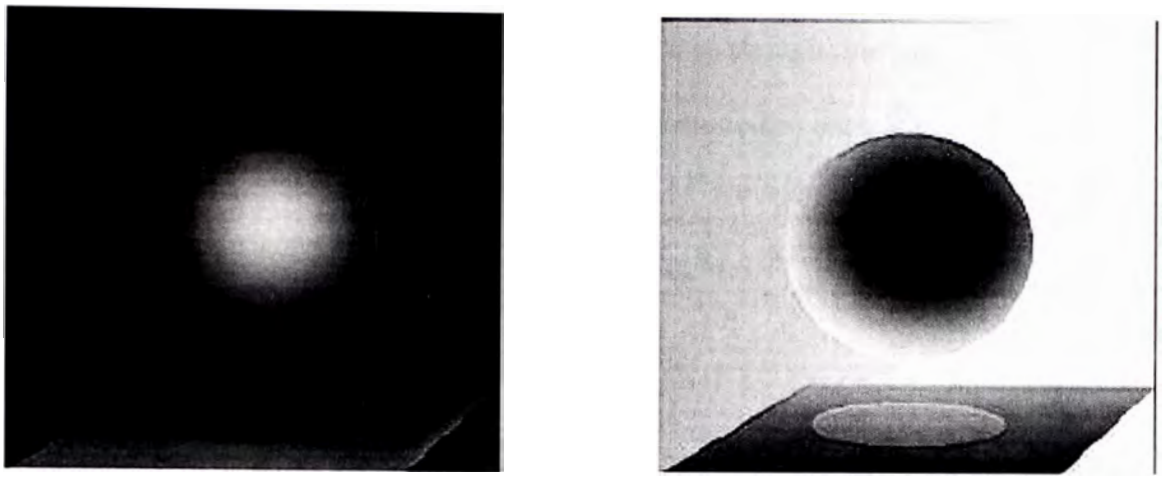
Para imágenes discretas, también se puede aproximar la Laplaciana en base a diferencias de la primera derivada. En este caso resulta la siguiente máscara:

$$\begin{vmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{vmatrix} \tag{7}$$

Otras máscaras donde intervienen todos los pixeles de la vecindad que ellas definen aparecen reseñadas en (Pratt, 1978) y (Lindley, 1991).

La laplaciana puede ser utilizado para realizar el reforzamiento de bordes o bien para detectar contornos en base a los cruces por cero. En lo referente al reforzamiento de bordes, este operador es no direccional y en consecuencia puede reforzar los bordes independientemente de su dirección y de su pendiente. En la figura 1.19b se muestra un ejemplo de su aplicación al procesar la imagen mostrada en la figura 1.19a. Se observa en este caso, que efectivamente produce un mejor realce que el obtenido con un filtro pasa alto mostrado en la figura 1.12.

En cuanto al uso del operador laplaciano como detector de contornos, una de las limitaciones de este método radica en su alta sensibilidad al ruido, pues como se hace la detección de cruces por cero, cualquier variación por pequeña que sea, puede generar un punto de contorno, de hecho, el método genera muchos contornos falsos que aparecen típicamente en aquellas regiones donde la varianza local de la imagen es pequeña.



(a) (b)

Figura 1.19 Realce de contornos basado en el uso del operador laplaciano (a) imagen original (b)imagen resultante al procesar con el operador laplaciano.

Para superar este inconveniente, se podría procesar el resultado obtenido al procesar con el operador laplaciano, mediante un umbral próximo a cero de modo que se eliminen como puntos del contorno aquellos cuya magnitud sea muy pequeña y que son producidos muy probablemente debido al ruido presente en la imagen

d) Otros detectores de contornos

a) ***Operador de Roberts***: Es un operador que aproxima el gradiente como la suma del valor absoluto del gradiente según dos direcciones ortogonales, de acuerdo a la siguiente ecuación:

$$y(i,j) = |f(i,j) - f(i+1,j+1)| + |f(i,j+1) - f(i+1,j)| \qquad (8)$$

En la figura 1.21c se muestra el reforzamiento de bordes obtenido mediante este operador al procesar la imagen de la figura 1.21a.

b) ***Operador de Sobel***: Otro de los operadores, utilizados para la acentuación de bordes, es el operador de Sobel, el mismo consiste en considerar, una vecindad de

3 x 3 pixeles, en donde cada uno de los pixeles se designa tal como se muestra en la figura 1.20. La imagen resultante de la acentuación está dada de acuerdo a la siguiente ecuación:

$$y(i,j) = \sqrt{X^2 + Y^2} \qquad (9)$$

$$X = (A2 + 2A3 + A4) - (A0 + 2A7 + A6) \qquad (10)$$

$$Y = (A0 + 2A1 + A2) - (A6 + 2A5 + A4) \qquad (11)$$

A0	A1	A2
A7	f(I,J)	A3
A6	A5	A4

Figura 1.20 Designación de los pixeles contenidos en la ventana corrediza usada en el operador de Sobel.

Lindley (1991) presenta una implantación del algoritmo de ***Sobel***, la cual no requiere extraer la raíz cuadrada, en consecuencia acelera el tiempo de ejecución del mismo. En la figura 1.21d se muestra el reforzamiento de bordes obtenido con este operador al procesar la imagen de la figura 1.21a. Normalmente, después de la aplicación del operador de ***Sobel***, resulta necesario procesar con un umbral a objeto de determinar los bordes correspondientes a los objetos que componen la escena, lo cual se muestra en la figura 1.21e.

1.3.4 Segmentación de imágenes

Para realizar la identificación de estructuras anatómicas presentes en la imagen, se utilizan las técnicas de segmentación, las cuales permiten particionar la imagen en un conjunto no solapado de regiones, cuya unión es la imagen completa. En muchas ocasiones, dependiendo de la aplicación específica, el proceso de segmentación es uno de los pasos difíciles y críticos para determinar la geometría de las diversas estructuras que componen la imagen. En general las técnicas de segmentación tienden a ajustarse a las siguientes reglas:

i) Las regiones resultantes del proceso de segmentación debieran ser uniformes y homogéneas respecto a alguna característica, tal como el nivel de gris o la textura.

ii) Las regiones interiores debieran ser simples y no incluir abundantes huecos o estructuras ruidosas.

iii) Las regiones adyacentes en una segmentación debieran tener valores diferentes con respecto a la característica según la cual son uniformes.

iv) Los limites de cada segmento debieran ser lo más simple posibles.

Lograr que se cumplan todas esas propiedades resulta a menudo difícil y por lo general lo que logran la mayor parte de métodos, son regiones en las que a menudo se observa la presencia de huecos y adicionalmente los limites o bordes de las mismas no son simples. Uno de los métodos que usualmente se sigue para implantar la segmentación consiste en primero determinar los bordes del objeto, utilizando las técnicas estudiadas en las secciones anteriores, seguidamente resulta necesario determinar el interior del objeto y clasificar los pixeles incluidos en tal borde como pertenecientes al objeto. Otras de las técnicas comúnmente utilizadas en

segmentación es la segmentación basada en el uso de un umbral y la segmentación por crecimiento de regiones.

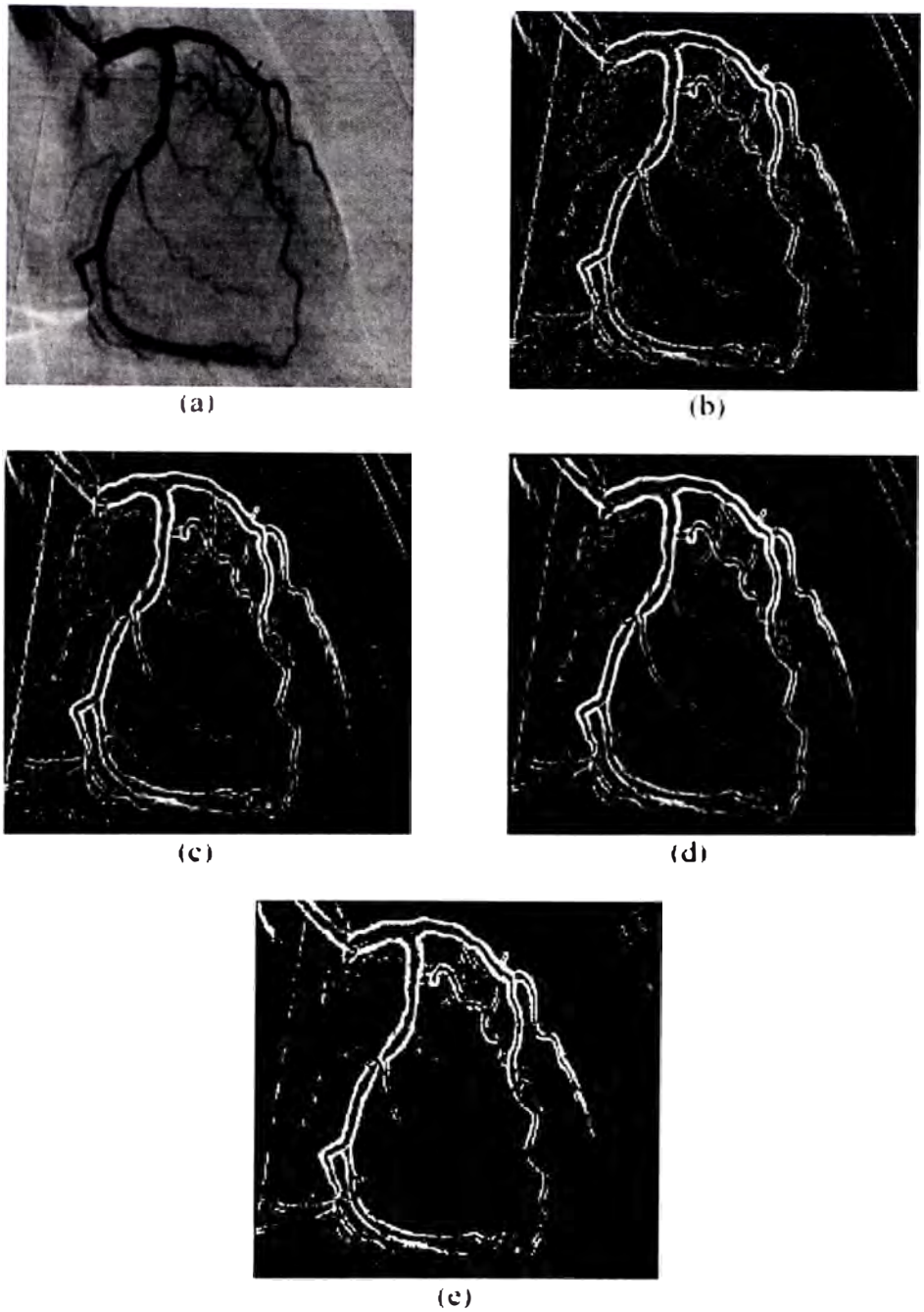


Figura 1.21 Ejemplo de realce y detección de contornos (a) imagen original(b) realce decontornos obtenido mediante el gradiente, (c) realce de contornos obtenido mediante el operador de Roberts (d) Realce obtenido mediante el operador de Sobel (e) Contorno resultante al procesar mediante un umbral la imagen obtenida en (d).

a) Segmentación basada en el uso de un umbral

Este tipo de segmentación, permite separar un objeto dentro de la imagen del fondo que lo circunda, la técnica se basa en comparar alguna propiedad de una imagen con un umbral fijo o variable, realizando tal comparación para cada uno de los pixeles que conforman la imagen, si el valor de la propiedad de un pixel supera el valor del umbral, entonces el pixel pertenece al objeto, en caso contrario, el pixel pertenece al fondo.

Cuando la segmentación se realiza basada en el nivel de gris de la imagen, el valor del nivel de gris de cada pixel debe ser comparado con el umbral, para decidir si tal pixel pertenece al objeto o al fondo. La imagen de salida, es una imagen binaria en la cual aquellos pixeles cuyo valor es 1, pertenecen al objeto y los pixeles cuyo valor es cero, pertenecen al fondo.

La selección del valor del umbral, se realiza generalmente a partir del histograma de la imagen. Así si una imagen está compuesta de un objeto que aparece en la escena sobre un fondo, entonces es de esperar que el histograma sea bimodal, es decir, si por ejemplo el objeto es más claro que el fondo, pues en el histograma aparecerán dos picos, el ubicado en los valores de gris más elevados correspondiente al objeto y otro pico para niveles de gris más bajos, correspondientes al fondo. En la figura 1.22 se muestra un histograma bimodal, en el cual el umbral se ubica entre los dos picos del histograma.

La selección automática del umbral, es un problema difícil, debido a que el histograma no siempre es bimodal, en cuyo caso resulta necesario combinar la información espacial presente en la imagen, con la información referente al nivel de

gris. En (Haralick y Shapiro, 1992) se presenta una revisión bibliográfica referente a este tema. Para el caso del histograma aproximadamente bimodal, existen técnicas de detección automática del umbral, una de las cuales fue ideada por Otsu y ha sido reseñada en (Haralick y Shapiro, 1992) y se basa en la minimización de la varianza intra-grupo.

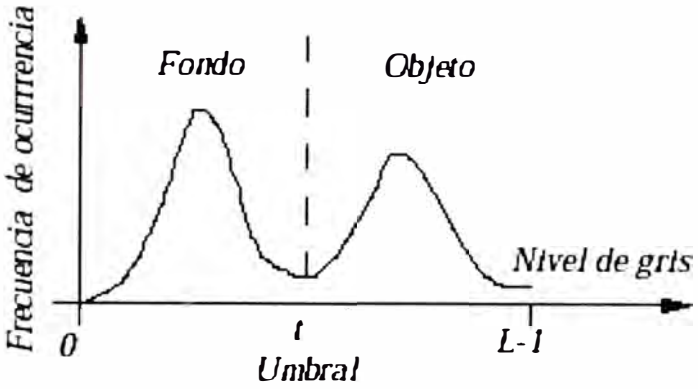


Figura 1.22 Ejemplo de un histograma bimodal, en este caso el umbral a utilizar en la segmentación debiera estar ubicado en el valle entre los dos picos del mismo.

Una implantación recursiva de este algoritmo la cual reduce apreciablemente el costo computacional, aparece reseñado en (Haralick y Shapiro, 1992). En la figura 1.23 se muestra un ejemplo de segmentación de una imagen ventriculográfica utilizando esta técnica. En este caso el histograma tiende a ser bimodal en donde el primer modo de mayor amplitud corresponde principalmente a los pixeles del fondo mientras que el segundo modo de menor amplitud incluye los pixeles correspondientes al objeto que en este caso es el ventrículo. Al binarizar esta imagen considerando un umbral con valor de 100, se obtiene la imagen que se muestra en la figura 1.23c en donde si bien se logra discriminar la forma del ventrículo se observa la existencia de pixeles aislados que no guardan ninguna relación con el objeto a determinar. Debido a ello tal imagen debe ser procesada a objeto de eliminar los

puntos no conexos tal como se muestra en la figura 1.23d, en donde el procesamiento se basa en estudiar el grado de conexidad de los pixeles dentro de una vecindad de 3 x 3 pixeles.

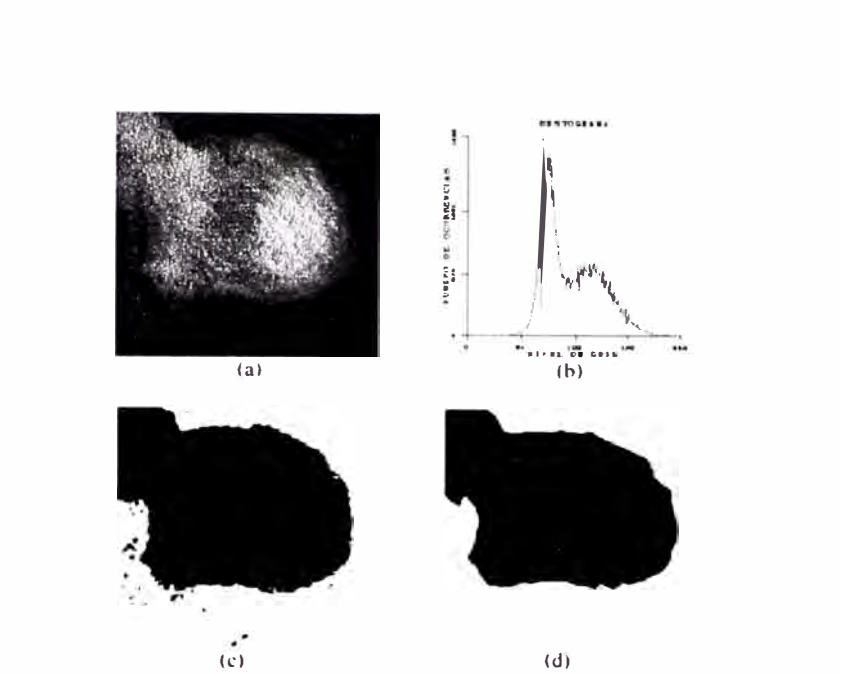


Figura 1.23 Ejemplo de la segmentación (a) imagen original correspondiente a una ventriculografia (b) histograma (c) segmentación obtenida mediante un umbral (d) forma del ventrículo obtenida luego de eliminar de manera automática los pixeles ruidosos de la imagen

b) Segmentación por crecimiento de regiones

De acuerdo a esta técnica, se buscan pixeles que tengan características similares (por ejemplo niveles de gris similares) y que adicionalmente sean vecinos. El método comienza con un pixel, el cual es seleccionado automáticamente o proporcionado por el usuario y a continuación examina los pixeles vecinos para decidir si tienen características similares. De ser así, el pixel vecino que cumpla con tal condición de similaridad, es agrupado junto con los anteriores para conformar así una región.

CAPITULO II

PROCESAMIENTO DIGITAL DE IMÁGENES EN MATLAB

2.1 Formato de señales 2d en matlab

Las imágenes son señales con dos variables independientes (espaciales) a diferencia de las señales monodimensionales con una sola variable independiente (temporal) que se ha manejado hasta ahora.

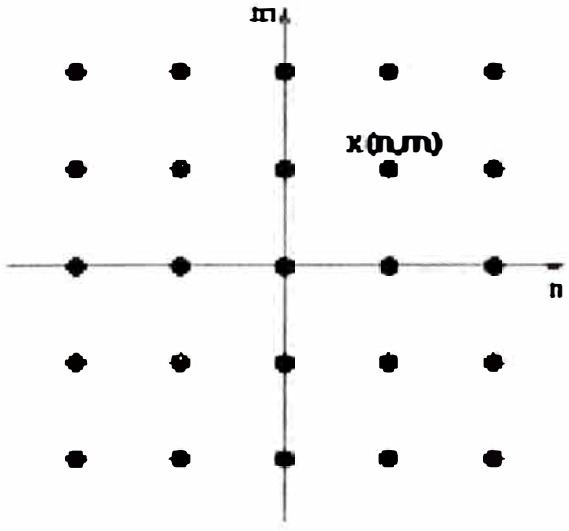


Figura 2.1 Representacion de una señal bidimensional.

En la figura 2.1 se representa una señal bidimensional **x(n,m)**. Como podemos ver, se trata de una matriz en la que cada elemento (o pixel) queda posicionado mediante las variables espaciales **n** y **m**, siendo **x(n,m)** un valor de intensidad de gris (imágenes en blanco y negro, **B**y**N**). Las imágenes en color se obtienen mediante la superposición de 3 imágenes que representan la cantidad rojo (matriz **R**), verde (matriz **G**) y azul (matriz **B**) de cada pixel. Este tipo de representacion de imágenes en color se denomina **RGB**.

Sin embargo, MATLAB usa otro tipo de representación de imágenes denominada Indexada. En este la imagen es también una matriz 2D X , pero en cada pixel encontramos un índice entero que apunta a un color de un mapa de colores o paleta map. La paleta es, en realidad, una matriz **N×3** en la que cada fila es un color representado por tres valores que indican las componentes **R**, **G** y **B** del color. De esta manera, se consigue un ahorro considerable en la cantidad de memoria requerida respecto a una imagen representada en formato **RGB**. Para leer (y mostrar) una de las imágenes disponibles, debe ejecutarse:

```
load <imagen>

imshow(X,map)
```

Las imágenes diponibles son **trees, cape, clown, kids, mri, forest, spine, canoe, shadow y autumn** entre otras. Estas imágenes pueden cambiarse a formato **RGB** mediante el comando **ind2rgb** o a una imagen **B**y**N** mediante **ind2gray**:

```
load <imagen>

I = ind2gray(X,map);

imshow(I,64)
```

En este caso, el segundo argumento del comando gráfico **imshow** representa el número de niveles de gris que se utilizará en la gráfica correspondiente.

Por razones de ahorro de memoria, será conveniente pasar las imágenes leídas a formato **B**y**N** y/o seleccionar una sección reducida de la imagen cargada. Para limpiar la memoria de MATLAB puede usarse el comando **clear**.

Tambien es posible importar imágenes disponibles en otros formatos estándar como GIF, TIFF, HDF, BMP, PCX o XWD, mediante los comandos **gifread**, **tiffread**, **hdfread**, etc. También existen los correspondientes comandos de escritura **gifwrite**, **tiffwrite**, etc.

2.2 Realce de Imágenes: Ecualización de Histogramas

Mediante el realce de imágenes se pretende resaltar o clarificar determinados aspectos de una imagen. Una de las técnicas más extendidas para el realce de imágenes es la de ecualización de histogramas. El histograma de una imagen es un gráfico en el que se muestra la distribución de la intensidad de color de una imagen. El histograma se obtiene dividiendo el intervalo de intensidades [0, 1] en un determinado numero de subintervalos. Para cada subintervalo se representa el número de pixels cuya intensidad cae dentro del mismo. Podemos realizar esto en MATLAB como:

```
load clown
I = ind2gray ( X,map );
subplot ( 2,1,1 ), imhist ( I,32 );
subplot ( 2,1,2 ), imshow ( I,32 );
```

donde el segundo argumento de **imshow** representa el número de subintervalos utilizado para realizar el histograma.

La técnica de ecualización de histogramas se basa en redistribuir las intensidades de los píxels de forma que el histograma cambie a otro predeterminado. El comando **histeq** modifica la imagen de forma que su histograma se aproxime a plano. Esta aproximación es tanto mejor cuanto menor es el número de subintervalos. La imagen resultante puede resultar poco natural, pero pueden clarificarse algunos detalles.

2.3 Restauración de Imágenes

A diferencia del realce, el objetivo de la restauración de imágenes es obtener a partir de una imagen degradada por algún tipo de distorsión otra imagen que represente lo mas fielmente posible a la original.

Por ejemplo se puede generar la señal degradada como:

```
J = imnoise ( I,'salt \& pepper' );
```

donde el segundo argumento de **imnoise** representa el tipo de ruido aplicado.

2.3.1 Filtrado Lineal

Una primera técnica de restauración puede ser la aplicación de un filtrado lineal pasabaja. El filtro pasabaja elimina las variaciones bruscas en la señal introducidas por el ruido aditivo. En la figura 2.2 se representa la respuesta en frecuencia de un filtro pasabaja 2D, donde se tienen en cuenta dos frecuencias, una vertical y otra horizontal. Aunque el filtro representado tiene una banda pasante cuadrada, es posible utilizar otras formas como circular o intermedia entre cuadrada

y circular. Podemos diseñar el filtro pasabaja mediante el método de la ventana de la siguiente forma:

```
Hd = zeros ( 11,11 );

Hd ( 4:8,4:8 ) = ones ( 5,5 );

h = fwind1 ( Hd,hamming (11) );
```

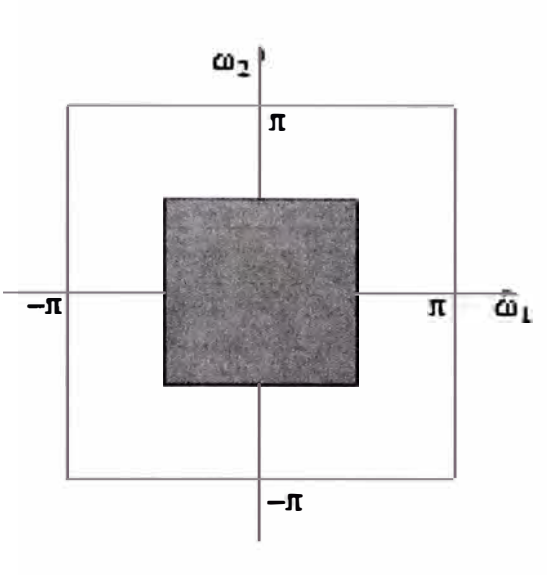


Figura 2.2 Respuesta en frecuencia de un filtro 2D pasabaja.

Es posible visualizar la respuesta en frecuencia del filtro diseñado mediante los siguientes comandos:

```
[ f1,f2 ] = freqspace ( [ 11 11 ] );

[ x,y ] = meshgrid ( f1,f2 );

colormap ( jet ( 64 ) )

subplot ( 1,2,1 ), surf ( x,y,Hd ), axis ( [ -1  1  -1  1  0  1.2 ] )

subplot ( 1,2,2 ), freqz2 (  h,[32 32] ), axis ( [ -1  1  -1  1  0  1.2 ] )
```

donde se ha utilizado el comando **freqz2** para obtener y dibujar 32×32 puntos de la respuesta en frecuencia del filtro. **freqspace** crea dos vectores con las frecuencias horizontal y vertical de muestreo, para construir la malla de representación [x,y].

2.3.2 Filtrado No Lineal

El filtrado lineal de imágenes presenta dos inconvenientes. El primero es que genera una imagen cuyos contornos aparecen desdibujados debido a la característica pasabaja del filtro. Además, no es capaz de eliminar muestras de tipo impulsivo claramente erróneas. Esto podemos observarlo en la figura 2.3, donde se representan las muestras de una señal monodimensional contaminada y el resultado de aplicar un filtro lineal pasabaja en línea discontinua.

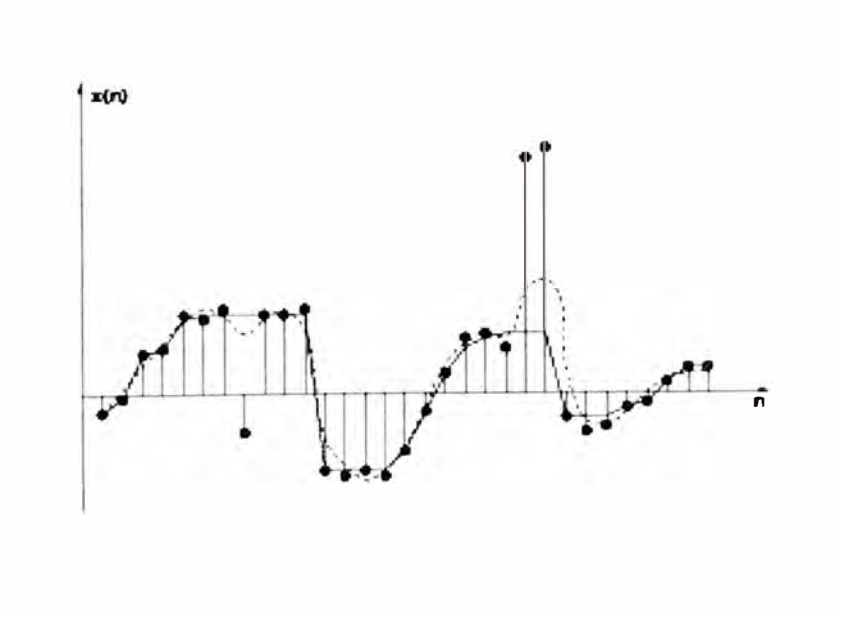


Figura 2.3 Efectos del filtrado de mediana.

Un filtro de mediana es un filtro no lineal que para cada muestra de señal de entrada **x(n)** obtiene una salida que es la mediana de las muestras contenidas en un entorno de radio **N** con centro en la muestra **n** actual,

$$y(n) = \text{median}(x(n-N), x(n+N))$$

La figura 2.3 muestra en línea continua el resultado de aplicar el filtro de mediana. Se observa que este filtro es capaz de eliminar muestras ruidosas de tipo impulsivo a la vez que mantiene los bordes nítidos, cualidades muy interesantes para señales de video. Para la aplicación de los filtros de mediana a imágenes en necesario considerar entornos bidimensionales de tamaño **N×M**.

2.4 Transformadas y Codificación

También se puede definir la **Transformada de Fourier** (FT) para señales bidimensionales. **MATLAB** dispone de un comando de transformada discreta 2D rápida (**fft2**). La FT transforma los 2 ejes espaciales en dos ejes de frecuencia (horizontal y vertical). Podemos observar el resultado de aplicar la FT a una imagen mediante el siguiente programa:

```
% Transformada de Fourier
load mandrill
Imshow ( X,map )
pause
I = ind2gray ( X,map );
Imshow ( I,64 )
pause
F = fftshift ( fft2 (I) );
Colormap ( jet ( 64 ) )
Imshow (log ( abs ( F ) ) )
pause
imagesc ( log (abs (F) ) )
```

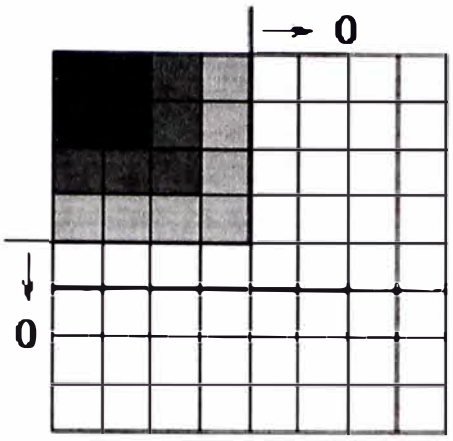


Figura 2.4 DCT y muestreo zonal.

La **Transformada Discreta del Coseno (DCT)** es otra transformación que presenta una importante aplicación en la codificación de imágenes (estándares JPEG, MPEG). La **DCT** tiene la propiedad de que es apropiada para realizar un muestreo zonal de la misma. El muestreo zonal consiste en suponer que los coeficientes transformados de orden superior son nulos. En la figura 2.4 se muestra la **DCT** de un bloque 8 × 8 de imagen. Conforme nos alejamos del origen, los coeficientes de la **DCT** tienden a ser nulos, por lo que puede suponerse que solo los primeros 4 × 4 coeficientes portan la información relevante, mientras que el resto pueden considerarse nulos sin cometer mucho error. De esta forma se ha reducido el número de coeficientes necesario para representar el bloque de imagen en un factor cuatro.

Los comandos **MATLAB** para la **DCT** y **DCT inversa** de señales 2D son **dct2** y **idct2**.

CAPITULO III

PROCESAMIENTO DE UNA REGIÒN DE INTERÈS EN UNA IMAGEN USANDO EL PROCEDIMIENTO ROIDEMO DEL MATLAB

Lo que se describe en esta sección tiene por objeto mostrar la implementación del procedimiento ROIDEMO escrito en MATLAB, que procesa una región determinada dentro de una imagen.

En primer lugar se describe la operación que realiza el usuario del programa para seleccionar una región de interés en la imagen y luego como escoger y aplicar determinado proceso a la zona seleccionada, por ejemplo incrementar el contraste, agregar ruido gaussiano, aplicarle un filtro pasabajo, etc.

Luego se explica la función de los módulos que conforman la estructura del programa, destacando aquellos más importantes como por ejemplo aquel módulo que escoge una imagen, el módulo que selecciona al área a ser procesada y aquel módulo que aplica el tipo de procesamiento escogido por el usuario.

En el ANEXO A se encontrará la descripción de los comandos más importantes usados en el programa mencionado, como son: **roipoly, roifill, roifill2 y filter2.**

3.1 ¿Cómo funciona el procedimiento?

Se podrá apreciar una ventana con diversas opciones que permitirán seleccionar una imagen en el menú de la izquierda (ver Figura 3.1). Para seleccionar el área de interés, se debe mover el puntero del mouse sobre la imagen de la izquierda notándose que el cursor cambia a una cruz. Hacer clic en puntos de la imagen de tal forma que nos indiquen los vértices seleccionados de la región deseada. El punto final debe ser marcado con un doble clic, shift-clic o clic-derecho.(También se puede finalizar la selección sin añadir otro vértice y presionando <enter> o <return> desde el teclado). Cuando se completa la selección, la región será mostrada en el centro como una imagen binaria.

Luego seleccionar la operación a realizar desde el menú **"Region of Interest Operations"** en el lado derecho y presionar el botón "**Apply**". La imagen original es filtrada usando la imagen binaria como una máscara, la operación es aplicada solamente en el área seleccionada de la imagen original la cual corresponde al área de color blanco en la máscara.

Cuando se selecciona una nueva imagen, se podrá notar que la operación de **ROI** ha cambiado también. Todas las imágenes tienen operaciones asociadas por defecto a cada una, pero se puede experimentar con diferentes operaciones para cada imagen.

Por ejemplo, la imagen llamada **"Aluminum and Noise"** es una imagen de granos de aluminio salpicadas con ruidos de "sal y pimienta". La operación por defecto es **"Filtro Medio" (Median Filter)**, debido a que el filtrado medio es especialmente efectivo para remover este tipo de ruido.Otro ejemplo es la imagen **"Blood",** cuya operación por defecto es **"Interpolación Límitrofe" (Boundary Interpolation)**. Esta operación llama a la función **roifill,** la cual interpola gradualmente dentro de una región arbitraria resolviendo la ecuación de Laplace dentro de la región. Lo hace seleccionando un polígono que rodea totalmente una de las celdas de **"Blood"**; es decir realmente elimina la celda de la imagen.

Para efectuar el proceso antes mencionado se tiene un programa en **MATLAB**, el cual realiza las transformaciones y procesamientos básicamente usando las siguientes instrucciones:

- **roipoly**
- **roifill**
- **roifilt2**
- **filter2**

las cuales se describen ampliamente en el ANEXO A

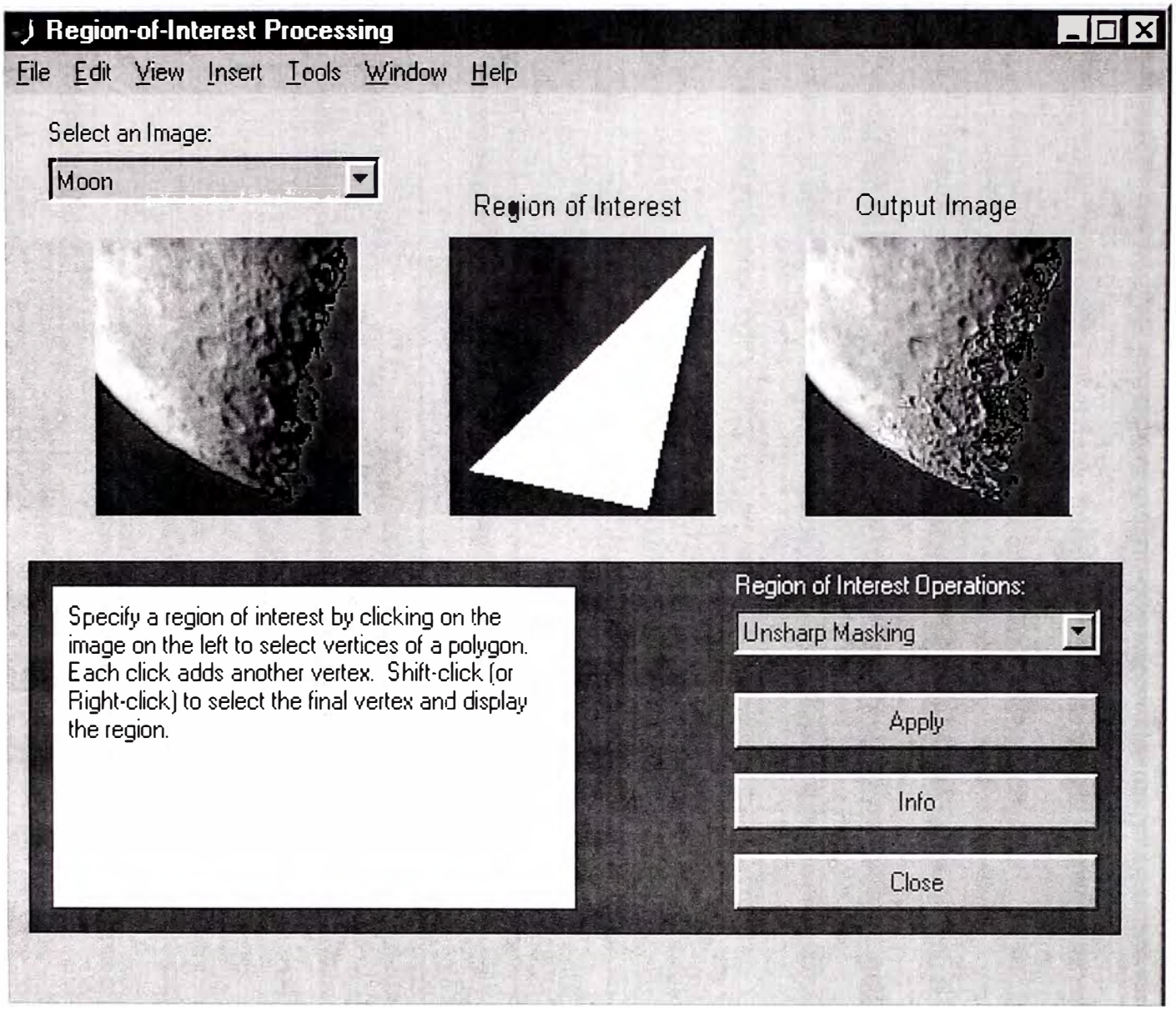

Region-of-Interest Processing
File Edit View Insert Tools Window Help
Select an Image:
Moon
Region of Interest
Output Image
Specify a region of interest by clicking on the image on the left to select vertices of a polygon. Each click adds another vertex. Shift-click (or Right-click) to select the final vertex and display the region.
Region of Interest Operations:
Unsharp Masking
Apply
Info
Close

Figura 3.1

3.2 Estructura del programa

Como se mencionó, el programa permite operar en una región de interés dentro de una imagen, teniendo el usuario que seleccionar una imagen inicialmente, para luego determinar una zona de interés dentro de la misma y finalmente escoger un determinado procesamiento para ser aplicado a la zona escogida.

El programa hace uso de subrutinas para la inicialización de pantallas y parámetros por defecto, así como para escoger las diferentes opciones de imágenes, zonas y tipos de procesamiento. En la Tabla 3.1 se describen las subrutinas que utiliza el programa:

Subrutina	Descripción
InitializaROIDEMO	Configura controles e imágenes.
Load NewImage	Carga una imagen nueva desde un archivo tipo **mat** y llama a **Apply Operation** para realizar la operación **ROI** dentro de ella
ApplyOperation	Revisa el menú **popup** para saber que operación se ha escogido y luego la realizar en la imagen original solamente dentro de la región de interés especificada por la máscara

Subrutina	Descripción
MyGetline	Almacena información en la estructura userdata de la figuras en lugar de hacerlo en variables globales. Almacena el primer punto y luego configura las funciones **button down** y **motion** para el resto del proceso de selección de la región de interés (**ROI**).
ButtonDown	Agrega nuevos vértices a la máscara poligonal que conforma la región de interés.
ButtonMotion	Realiza una animación a la selección del polígono.
KeyPress	Subrutina que controla que hacer cuando se presione una tecla durante el proceso de selección del **ROI**.
CompleteGetline	Completa la selección del **ROI** y retorna la figura a la ventana más o menos en el estado que se encontraba antes que fuera invocada la rutina **MyGetline**. Resetea las funciones **ButtonDown**, **ButtonMotion**, y **KeyPress**. También calcula la máscara usando el comando **roipoly** y luego la muestra.

Subrutina	Descripción
UpdateOperation	Actualiza la operación **ROI** del valor por defecto a la operación escogida en el menú **popup**.
NormalMotionFcn	Esta es la función por defecto de la figura en la ventana. Coloca el puntero como una flecha cuando no está sobre la imagen original y coloca el puntero como una cruz cuando está sobre la imagen original

3.2.1 Inicialización del programa. Subrutina InitializeROIDEMO

Es una de las rutinas más extensas del programa. Se encarga de inicializar los entornos y parámetros que conforman la pantalla del procedimiento. Entre otras cosas realiza lo siguiente:

- Presenta los rótulos de los menús y textos explicativos e indicativos.
- Configura tamaño y formato de la pantalla inicial.
- Presenta mensaje de error cuando el tamaño o resolución de la pantalla no es adecuada para mostrar las figuras.
- Configura espaciamientos horizontal, vertical y de márgenes internos.
- Configura los parámetros para todos los menús y botones, así como para la imagen original, la máscara de la imagen, la imagen de salida, el área de trabajo,etc, entre las que se encuentran las que corresponden a los parámetros

del menu **popup** y el menú de las operaciones que se pueden procesar en la imagen (lineas 281 a 316 del listado del programa en el Anexo B).

- Configura el **status bar** y los botones **Apply, Info** y **Close**

3.2.2 Selección de la imagen. Subrutina LoadNewImage

Comienza en la línea 674 del programa y luego de obtener el identificador de la imagen sobre la cual se va aplicar el proceso hace uso de una instrucción **switch (case)** para cargar la imagen seleccionada en la variable **img** y además asignar el valor por defecto de la operación a realizar sobre dicha imagen. Por ejemplo si se ha escogido la figura "Aluminum and Noise" esta es una imagen de granulos de aluminio corrompida con ruido tipo "sal y pimienta" y el proceso por defecto asociado es "Filtro Medio", debido a que es especialmente efectivo para remover este tipo de ruido. Sin embargo se le puede aplicar cualesquiera de los procesos que han sido definidos en el menú respectivo.

Para el caso de este programa se cuenta con las siguientes imágenes: Blood, Saturn, Pout, Quarter, Circuit, Aluminum and Noise, Trees , Flower, Rice, Moon y Bone Marrow.

En la figura 3.2 se muestra el listado de esta subrutina

```
%%%  Sub-Function - LoadNewImage
%%%

function LoadNewImage(DemoFig)

callb = 0;
if nargin<1
   DemoFig = gcbf;
   callb = 1;   % We are in a callback
end

set(DemoFig,'Pointer','watch');
ud=get(DemoFig,'Userdata');
v = get(ud.h.ImgPop,{'value','String'}).
name = deblank(v{2}(v{1},:));
drawnow

switch name
case 'Blood'
   blood = [];
   load imdemos blood
   img = blood;
   set(ud.h.OperationsPop, 'Value', 9);
case 'Saturn',
   saturn = [];
   load imdemos saturn
   img = saturn;
   set(ud.h.OperationsPop, 'Value', 1);
case 'Pout',
   pout = [];
   load imdemos pout
   img = pout;
   set(ud.h.OperationsPop, 'Value', 2);
case 'Quarter',
   quarter = [];
   load imdemos quarter
   img = quarter;
   set(ud.h.OperationsPop, 'Value', 1);
case 'Circuit',
   circuit2 = [];
   load imdemos circuit2
   img = circuit2;
   set(ud.h.OperationsPop, 'Value', 1);
case 'Aluminum and Noise',
   alumgrns = [];
   load imdemos alumgrns
   img = imnoise(alumgrns,'Salt & Pepper', .05);
   set(ud.h.OperationsPop, 'Value', 4);
case 'Trees',
   trees = [];
   load imdemos trees
   img = trees;
   set(ud.h.OperationsPop, 'Value', 10);
case 'Flower',
   flower = [];
   load imdemos flower
   img = flower;
   set(ud.h.OperationsPop, 'Value', 7);
case 'Rice',
   rice = [];
   load imdemos rice
   img = rice;
   set(ud.h.OperationsPop, 'Value', 3);
case 'Moon',
   moon = [];
   load imdemos moon
   img = moon;
   set(ud.h.OperationsPop, 'Value', 1);
case 'Bone Marrow',
   bonemarr = [];
   load imdemos bonemarr
   img = bonemarr;
   set(ud.h.OperationsPop, 'Value', 5);
otherwise
   error('roidemo: Unknown Image Option!');
end

UpdateOperation(DemoFig);
img = double(img)/255;
set(ud.h.OriginalImage, 'Cdata', img);
if callb
   set(ud.h.Apply, 'Enable', 'on');
end
set(DemoFig, 'Pointer', 'arrow');
setstatus(DemoFig, 'Press "Apply" to perform
 the operation on the region of interest.');
```

Figura 3.2

3.2.3 Selección del área de interés ROI

Para lograr este propósito el programa cuenta con diversas subrutinas:

La rutina **MyGetline** almacena el primer punto seleccionado y deriva el control a las rutinas **ButtonDown** y **Motion** para el resto del proceso de selección de vértices del polígono que conforma el área seleccionada.

ButtonDown se encarga de manejar el proceso de agregar nuevos vértices, mientras que **ButonMotion** realiza una animación de la selección del polígono para poder apreciar el área escogida.

La rutina **Keypress** se encarga de manejar el programa cuando se presiona una tecla durante el proceso de selección del **ROI.**

Finalmente **CompleteGetline** es la rutina que completa la selección del polígono considerando dicha finalización con el uso de clic derecho, shift clic o doble clic. Luego entrega la figura casi como estaba antes de que se invoque a **MyGetline**. Esta rutina también resetea las funciones **ButtonDown, Motion** y **Keypress.**

3.2.4 Selección del proceso a realizar y su aplicación

Al ingresar al menú **popup** para escoger el proceso a aplicarse a la región de interés la subrutina **NormalMotion** maneja la asignación por defecto a cada imagen.

Una vez que se ha escogido desde el menú **popup** el proceso a realizar, la subrutina **UpdateOperation** se encarga de actualizar dicha opción.

Es ahora entonces que la rutina **ApplyOperation** lee la operación escogida y la aplica a la imagen original solamente en la región de interés y finalmente dicho resultado es mostrado en la ventana correspondiente.

Es en esta rutina **ApplyOperation** en donde se puede apreciar los diferentes tipos de procesos que se aplican a una imagen. Observemos en el listado total del programa que a partir de la línea 607 usando un comando **switch (case)** se selecciona las diferentes opciones del case mostrándonos operaciones como:

- Ecualización por Histograma
- Filtro Pasabajo
- Máscara Unsharp
- Filtro Mediano
- Aumento del Brillo
- Reducción del Brillo
- Incremento de Contraste
- Reducción de Contraste
- Interpolación Limítrofe
- Agregar Ruido Gaussiano

En la figura 3.3 se quiere mostrar como ejemplo los comandos de la rutina que manejan el caso de haber escogido aplicar un "Filtro Pasa Bajo" a una región determinada, se puede apreciar que hace uso de una ventana de Hanning de tamaño 15 y del comando **roifilt2** para lograr dicho filtrado.

```
case 'Lowpass Filter'
   order = 15;
   cutoff = 0.3;
   [f1,f2] = freqspace(order,'meshgrid');
   d = find(f1.^2+f2.^2 < cutoff^2);
   Hd = zeros(order);
   Hd(d) = 1;
   % Use hanning(15) as the window.  The window coefficients
   % are inlined to remove dependency on the Signal Processing
   % Toolbox.
   h = fwind1(Hd, ...
           [0.0381 0.1464 0.3087 0.5000 0.6913 0.8536 ...
            0.9619 1.0000 0.9619 0.8536 0.6913 0.5000 ...
            0.3087 0.1464 0.0381]);
   result = roifilt2(h, orig, mask);
```

Figura 3.3

En la figura 3.4 se muestra el caso de haber seleccionado la imagen **"Aluminum and Noise"**, y donde observamos que el programa carga mediante el archivo ejecutable **imdemos** dicha imagen en el WorkSpace y la guarda en la variable **img** y por medio del comando **imnoise** le asigna por defecto un ruido **"Salt & Pepper"**.

```
case 'Aluminum and Noise',
   alumgrns = [];
   load imdemos alumgrns
   img = imnoise(alumgrns,'Salt & Pepper', .05);
   set(ud.h.OperationsPop, 'Value', 4);
```

Figura 3.4

CAPITULO IV

APLICACIONES USANDO ARCHIVOS EXTERNOS AL PROCEDIMIENTO

En este capítulo se mostrarán algunos ejemplos de aplicación del procedimiento **roidemo** con imágenes escaneadas para dicho propósito, almacenadas en el disco duro de la computadora con un formato TIF , tamaño de 128 x 128 pixels y en escala de grises de 8 bits (TIF 8 bit gray-scale), tal como son las imágenes originales del procedimiento.

4.1 Captura de imágenes externas al procedimiento

Las imágenes han sido introducidas al procedimiento **roidemo** para su visualización y proceso, en primer lugar ejecutando el archivo **imdemos.mat** quien carga los archivos de imágenes en el **WorkSpace** del programa; a continuación importando las imágenes escaneadas a dicho **WorkSpace** y finalmente grabando los cambios hechos. Es importante destacar que el procedimiento **roidemo** invoca al archivo **imdemos** cuando se ejecuta la subrutina **LoadNewImage** como se puede observar en el listado del programa (líneas 670 al 755 en ANEXO A):

4.2 Modificaciones al procedimiento original

Para el caso de la presente aplicación el procedimiento fué ligeramente modificado removiendo líneas de comandos en dos áreas específicas del programa:

- Primero, en la subrutina que configura el menú **popup** que se encarga de escribir los nombres de las imágenes, reemplazando en esta sección los nombres de las primeras 5 imágenes originales por los nombres de las 5 imágenes escaneadas que se mencionaron al inicio del capítulo. Concretamente esto se realizó modificando la línea 286 del programa y que es parte de la subrutina **ImagePopUp**.

  En la figura 4.1 se muestra la modificación de los 5 primeros nombres de archivos de imágenes usados en el menú **popup**, observamos que los nombres Edificio1, Edificio2, Texto1, Texto2 y Texto3 reemplazan a los nombres **Moon**, **Flower**, **Blood**, **Circuit** y **Rice** del menu original.

```
280    %================================
281    % Image popup menu
282 -  ipleft = fleft+ifs;
283 -  ipbot = imageBot+128+vs*.5;
284 -  ud.h.ImgPop = uicontrol(Menu, ...
285       'Position',[ipleft ipbot (fwid-4*ifs)/3 menuHt], ...
286       'String',['Edificio1|Edificio2|Textol|Texto2|Texto3|Pout' ...
287          '|Trees|Saturn|Quarter|Bone Marrow|Aluminum and Noise'], ...
288       'Callback','roidemo(''LoadNewImage'')');
289    % Text label for Image Menu Popup
290 -  uicontrol( Text, ...
291       'Position',[ipleft ipbot+menuHt menuWid txtHt], ...
292       'Background', wdcolor, ...
293       'Foreground', 'black', ...
294       'String','Seleccione una imagen:');
295
```

Figura 4.1 Subrutina Image popup menu

- Segundo, en la subrutina **LoadNewImage** que se encarga de cargar la imagen escogida en el menú **popup**, se modifican algunos nombres de archivos de imágenes. Observamos en la fig 4.2 que en la sentencia **switch-case**, se han utilizado los nombres de archivos: **edificio1**, **edificio2**, **texto1**, **texto2**, **texto3** en reemplazo de **moon**, **flower**, **blood**, **circuit** y **rice** respectivamente, similar al caso anterior del menú **popup** .

```
688 -   name = deblank(v{2}(v{1},:));
689 -   drawnow
690
691 -   switch name
692 -   case 'Textol'
693 -      textol = [];
694 -      load imdemos textol
695 -      img = textol;
696 -      set(ud.h.OperationsPop, 'Value', 9);
697 -   case 'Saturno',
698 -      saturn = [];
699 -      load imdemos saturn
700 -      img = saturn;
701 -      set(ud.h.OperationsPop, 'Value', 1);
702 -   case 'Nino',
703 -      pout = [];
704 -      load imdemos pout
705 -      img = pout;
706 -      set(ud.h.OperationsPop, 'Value', 2);
707 -   case 'Quarter',
708 -      quarter = [];
709 -      load imdemos quarter
710 -      img = quarter;
711 -      set(ud.h.OperationsPop, 'Value', 1);
712 -   case 'Texto2',
713 -      texto2 = [];
714 -      load imdemos texto2
715 -      img = texto2;
716 -      set(ud.h.OperationsPop, 'Value', 1);
717 -   case 'Aluminio y Ruido',
718 -      alumgrns = [];
719 -      load imdemos alumgrns
720 -      img = imnoise(alumgrns,'Salt & Pepper', .05);
721 -      set(ud.h.OperationsPop, 'Value', 4);
722 -   case 'Arboles',
723 -      trees = [];
724 -      load imdemos trees
725 -      img = trees;
726 -      set(ud.h.OperationsPop, 'Value', 10);
727 -   case 'Edificio2',
728 -      edificio2 = [];
729 -      load imdemos edificio2
730 -      img = edificio2;
731 -      set(ud.h.OperationsPop, 'Value', 7);
732 -   case 'Texto3',
733 -      texto3 = [];
734 -      load imdemos texto3
735 -      img = texto3;
736 -      set(ud.h.OperationsPop, 'Value', 3);
737 -   case 'Edificio1',
738 -      edificio1 = [];
739 -      load imdemos edificio1
740 -      img = edificio1;
741 -      set(ud.h.OperationsPop, 'Value', 1);
742 -   case 'Bone Marrow',
743 -      bonemarr = [];
744 -      load imdemos bonemarr
745 -      img = bonemarr;
746 -      set(ud.h.OperationsPop, 'Value', 5);
747 -   otherwise
748 -      error('roidemo: Unknown Image Option');
749 -   end
750
```

Figura 4.2

También se modificaron algunas líneas del programa que muestran el texto en la ventana del procedimiento, de tal manera que la información en inglés fue traducida al español (títulos de ventana, botones, texto explicativo, etc.), permitiendo una mejor interface de usuario para las personas de habla hispana.

4.3 Nueva apariencia del procedimiento

Para finalizar en la figuras que siguen se muestran varias vistas de la nueva apariencia de la ventana del procedimiento **roidemo** y los resultados de la aplicacion de algunas de las opciones escogidas.

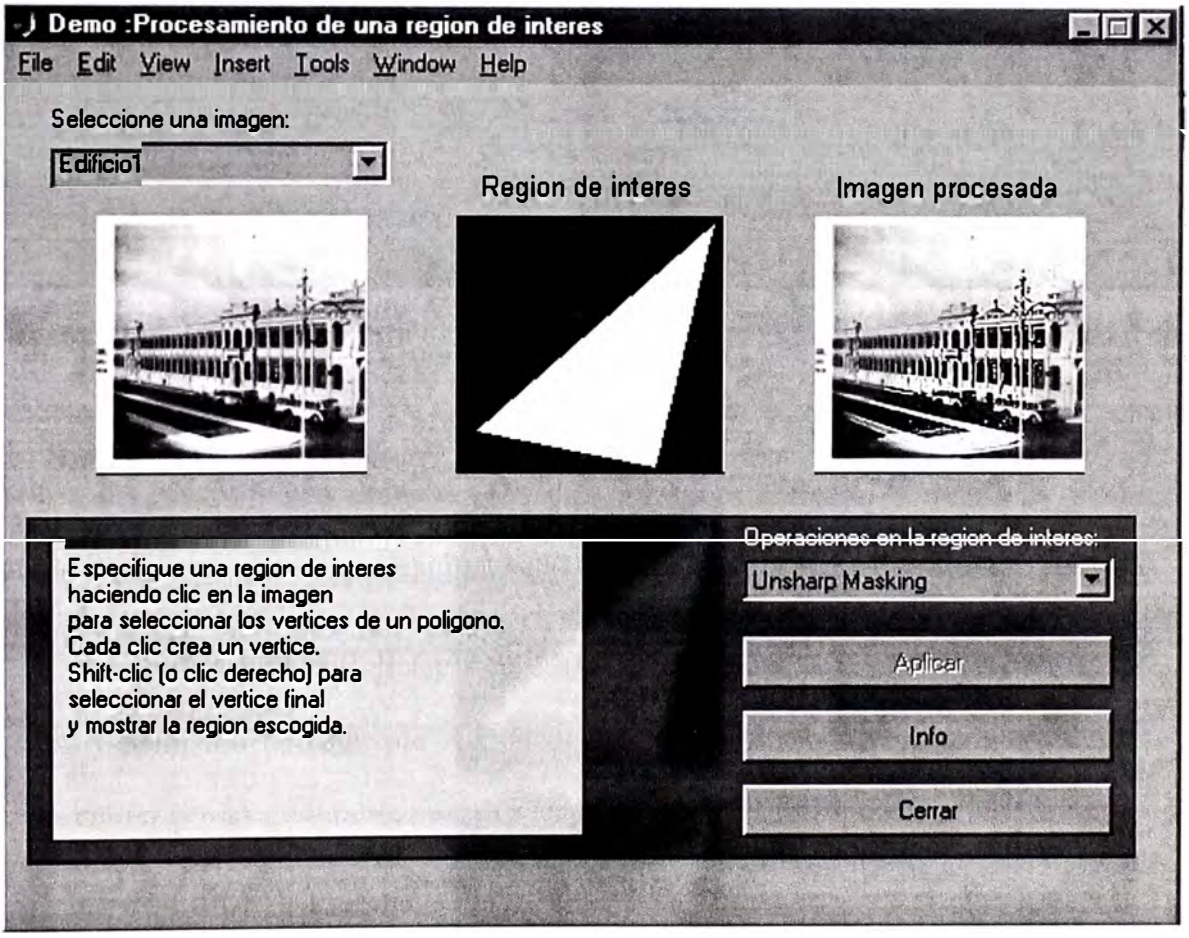


Figura 4.4 Ventana principal del procedimiento roidemo

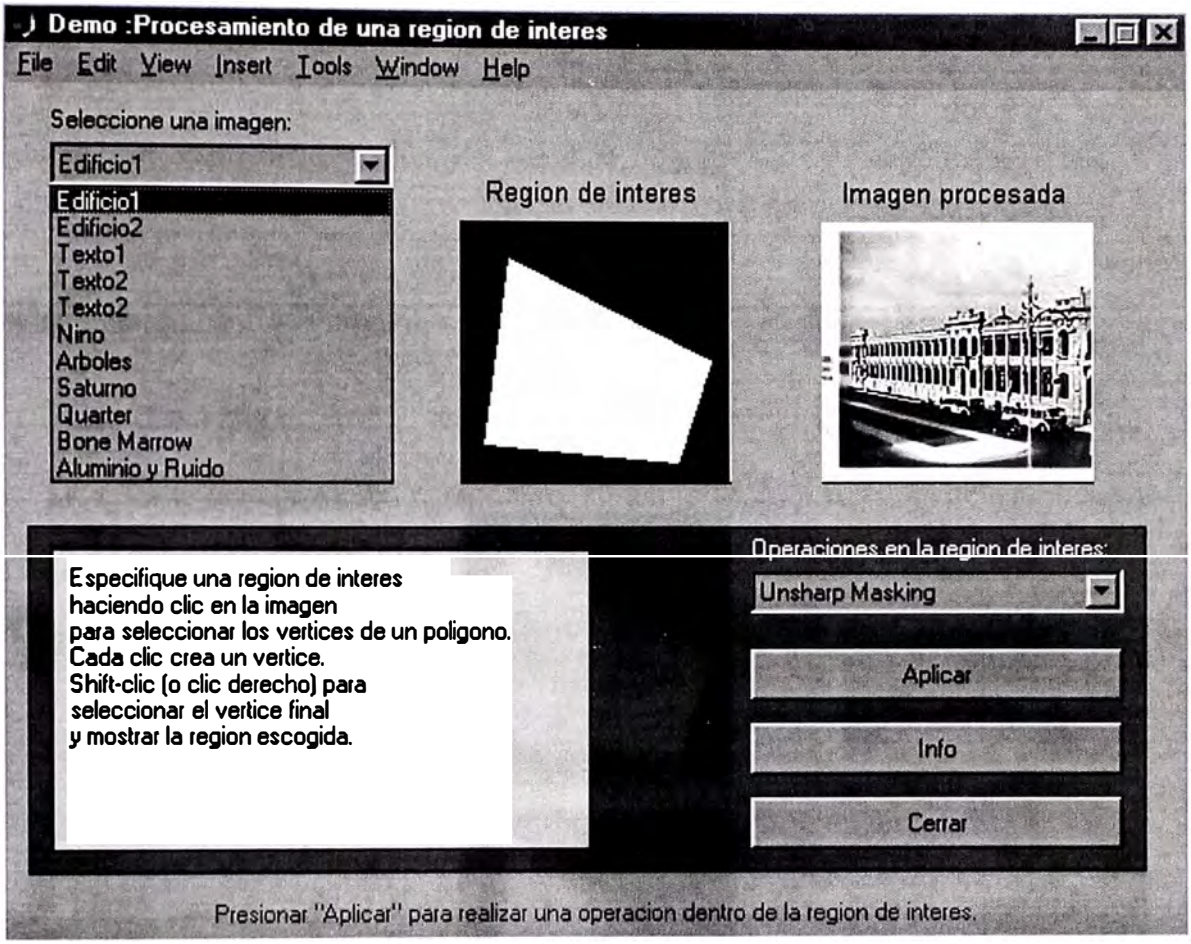


Figura 4.5

La figura 4.5 nos muestra el menu **popup** modificado y donde se puede encontrar los nombres de archivos : Edificio1, Edificio2, Texto1, Texto2 y Texto3, y además los textos explicativos y titulos de ventana están en español (a excepción de la barra de menús que es controlada por otro programa).

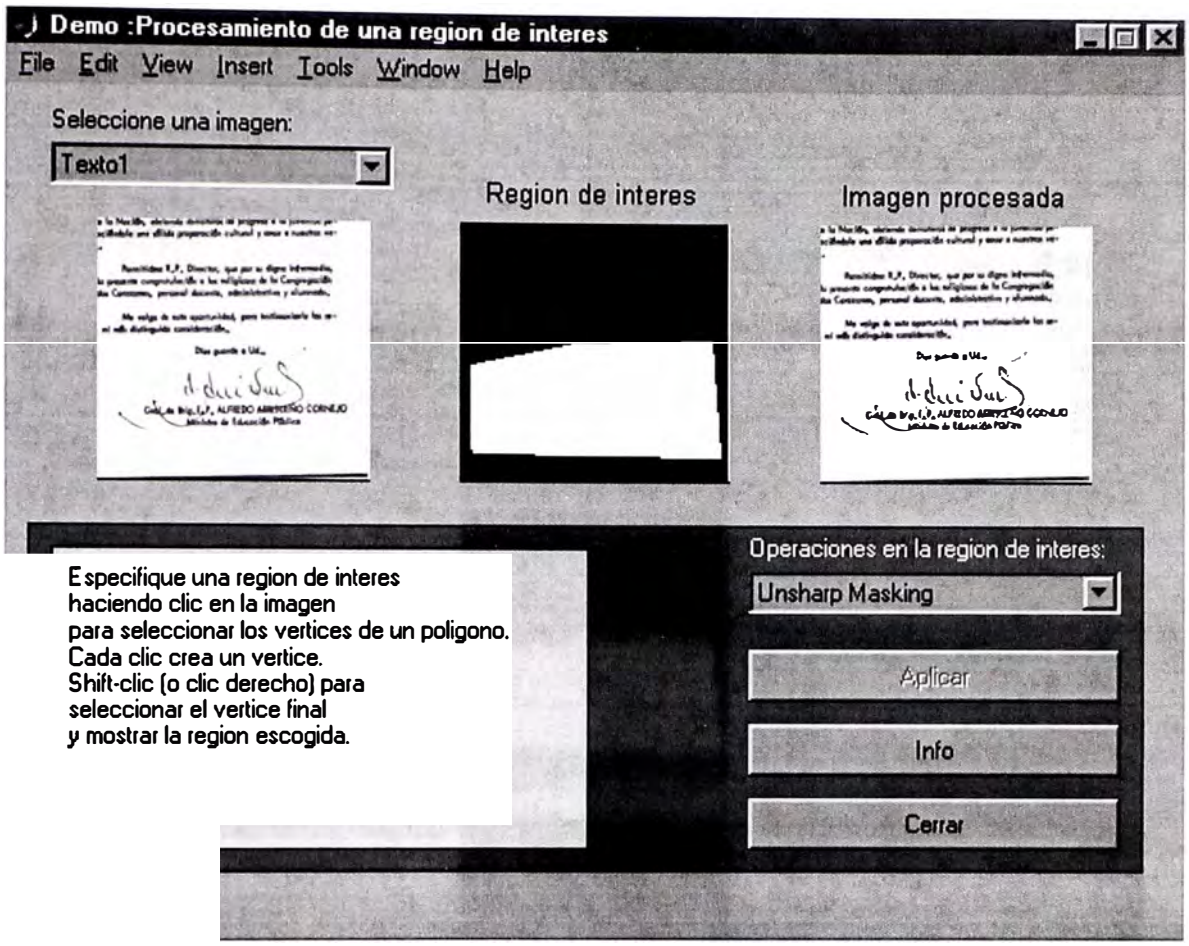


Figura 4.6

En la figura 4.6 observamos a manera de ejemplo como se ha aplicado la operación **Unsharp Masking** a la imagen **Texto1** a la cual se le ha seleccionado una región que incluye la firma del documento, obteniendo como resultado final una mejor legibilidad en dicha área.

CONCLUSIONES Y RECOMENDACIONES

1. El procesamiento digital de imágenes ha progresado enormemente en estos últimos años gracias al uso de la computadora como herramienta de cálculo, y se ha podido observar que el tratamiento de las mismas que hasta hace poco demandaba mucho esfuerzo, hoy en día es mucho más rápido y sencillo usando programas especializados.

2. El uso de MATLAB como programa especializado en este tipo de procesamiento es bastante satisfactorio porque permite la aplicación de una gama enorme de comandos que simplifican el tedioso cálculo matemático y permiten una mejor observación de los fenómenos que se presentan en el tratamiento de las señales.

3. En el caso de este tratado, se ha podido comprobar que mediante un programa hecho en MATLAB se puede presentar fácilmente un procedimiento que permite escoger una imagen de una lista, para luego delimitar un área determinada de la misma y aplicarle un procesamiento también seleeccionado por el usuario; y todo esto teniendo una interfaz amigable y fácil de utilizar.

4. MATLAB puede importar o exportar imágenes a otros formatos usuales como gif, tiff, hdf,bmp, pcx o xwd. Para ello dispone de los correspondientes comandos gifread, gifwrite, tiffread, tiffwrite, etc...

5. El uso de los comandos como **roipoly**, **roifill**, **roifilter**, **filt2** permite simplificar el proceso del tratamiento de una señal en una región seleccionada dentro de la imagen.

6. Se ha comprobado que una buena técnica para el realce de imágenes es la ecualización de histogramas.

7. La estructura de un programa en MATLAB no difiere sustancialmente de la foma como se lleva a cabo la programación con otros lenguajes, esto le da un carácter de versatilidad y potencia de procesamiento sobre todo cuando se involucra ecuaciones y fórmulas matemáticas avanzadas.

8. Se ha comprobado que el procedimiento descrito en este informe, con algunas modificaciones, puede ser aplicado al tratamiento de imágenes de textos incluídos en libros importantes que se encontraban en mal estado por el paso del tiempo, mal uso u otros motivos; y que es el reflejo de situaciones que se presentan en las más importantes bibliotecas o centros de manejo de documentos, permitiendo la posibilidad de generar copias legibles y su posterior almacenamiento en medios más confiables.

ANEXO A

DESCRIPCIÓN DE COMANDOS

En esta sección se describen los comandos más importantes que utiliza el programa en MATLAB que genera el procedimiento mencionado en el capìtulo III. Se muestra la sintaxis de ellos y se complementa la información con un ejemplo sencillo.

A1. roipoly

Selecciona una región poligonal de interés.

Sintaxis

BW = roipoly (I,c,r)

BW = roipoly (I)

BW = roipoly (x,y,I,xi,yi)

[BW,xi,yi] = BW = roipoly (I,c,r)

[x,y,BW,xi,yi] = roipoly (...)

Descripción

Se usa **roipoly** para seleccionar una región poligonal de interés dentro de una imagen, **roipoly** proporciona una imagen binaria que puede usarse como una máscara para **filtrado enmascarado.**

BW = roipoly (I,c,r) retorna la región de interés seleccionada por el polígono descrito por los vectores **c** y **r**. **BW** es una imagen binaria similar a **I,** con **0´s** fuera de la región de interés y **1's** dentro de la misma.

BW = roipoly (I) presenta la imagen de **I** en la pantalla y permite al usuario especificar el polígono usando el mouse:

- Si se omite **I**, **roipoly** opera sobre la imagen en los ejes predeterminados.
- Se usa los botones normales del clic para agregar los vértices al polígono.
- Para eliminar el vértice previamente seleccionado se presiona **Backspace** o **Delete.**

- Para agregar un vértice final a la selección y comenzar el relleno, hacer clic izquierdo, clic derecho o doble clic.
- Al presionar **Return,** se finaliza la selección sin agregar vértices.

BW = roipoly (x,y,I,xi,yi) Usa los vectores **x** e **y** para establecer un sistema coordenado espacial (nondefault), **xi** e **yi** son vectores del mismo tamaño que especifican vértices y posiciones del polígono en este sistema de coordenadas.

[BW,xi,yi] = roipoly (...) Proporciona las coordenadas del polígono en **xi** e **yi**. Notar que **roipoly** siempre produce un polígono cerrado. Si los puntos especificados describen un polígono cerrado (es decir, si el último par de coordenadas es idéntico al primer par), el tamaño de **xi** e **yi** es igual al número de puntos especificado. Si los puntos especificados no describen un polígono cerrado, **roipoly** agrega un punto final que tiene las mismas coordenadas que el primer punto. (En este caso el tamaño de **xi** e **yi** es más grande en uno que el número de puntos especificado).

[x,y,BW,xi,yi] = roipoly (...) Proporciona el **Xdata** e **Ydata** en **x** e **y**, la máscara de la imagen en **BW**, y las coordenadas del polígono en **xi** e **yi.**

Si **roipoly** es llamado sin argumentos de salida, la imagen resultante es presentada en una nueva figura.

La imagen de entrada **I** puede ser de clase **uint8**, **uint16** o **double.**

En cualquiera de la sintaxis usadas con **roipoly**, se puede reemplazar la imagen de entrada **I** con dos argumentos, **m** y **n**, que especifican las dimensiones de

la fila y la columna de una imagen arbitraria. Por ejemplo, los siguientes comando crean una máscara binaria de 100 x 200.:

```
c = [ 112 112 79 79 ];
r = [ 37 66 66 37 ];
BW =  roipoly [ 100, 200, c ,r ];
```

Si se especifica **m** y **n** con una forma interactiva de **roipoly**, entonces de presenta en la pantalla una imagen de color negro de **m** por **n**.

Ejemplo

```
i = imread ( 'eight.tif' );
c = [ 222 272 300 270 221 194 ];
r = [ 21 21 75 121 75 ];
BW = roipoly [ I,c,r ];
imshow (I);figure, imshow ( BW )
```

A2. roifill

Interpola gradualmente dentro de una región arbitraria de una imagen.

Sintaxis

J = roifill (I,c,r)

J = roifill (I)

J = roifill (I,BW)

[J, BW] = roifill (...)

J = roifill (x, y,I,xi, yi)

[x,y,J,BW,xi,yi] = roifill (...)

Descripción

roifill Rellena un polígono específico dentro de una imagen. Mediante resolución de la ecuación de Laplace interpola gradualmente desde los valores mas internos de pixeles hasta los límites del polígono. **roifill** puede ser usado, por ejemplo, para "borrar" pequeños objetos en una imagen.

J = roifill (I,c,r,) Rellena el polígono especificado por **c** y **r**, los cuales son vectores del mismo tamaño que contienen las coordenadas de fila-columna de los pixeles en los vértices del polígono. El k-ésimo vértice es [r(k), c(k)].

J = roifill (I) Presenta la imagen de **I** en la pantalla y permite al usuario especificar el polígono usando el mouse:

- Si se omite **I**, **roifill** opera sobre la imagen en los ejes predeterminados.

- Se usa los botones normales del clic para agregar los vértices al polígono.
- Para eliminar el vértice previamente seleccionado se presiona **Backspace** o **Delete.**
- Para agregar un vértice final a la selección y comenzar el relleno, hacer clic izquierdo, clic derecho o doble clic.
- Al presionar **Return,** se finaliza la selección sin agregar vértices.

J = roifill (I, BW) usa **BW** (una imagen binaria del mismo tamaño que **I**) como una máscara. **roifill** rellena en las regiones dentro de **I** correspondientes a los pixeles non-zero en **BW**. Si hay múltiples regiones, **roifill** realiza la interpolación de cada región independientemente.

[J, BW] = roifill (...) Proporciona la máscara binaria utilizada para determinar cuales pixeles en **I** son rellenados. **BW** es una imagen binaria del mismo tamaño que **I** con **1's** para pixeles que corresponden a la región interpolada de **I** y **0's** de otra forma.

J = roifill (x,y,I,xi,yi) Usa los vectores **x** e **y** para establecer un sistema coordenado espacial (nondefault), **xi** e **yi** son vectores del mismo tamaño que especifican vértices y posiciones del polígono en este sistema de coordenadas.

[x,y,J,BW,xi,yi] = roifill (...) Proporciona el **Xdata** e **Ydata** en **x** e **y**, la salida de imagen en **J**, la máscara de la imagen en **BW**, y las coordenadas del polígono en **xi** e **yi.** Si se usa la forma **roifill (I,BW)** entonces **xi** e **yi** estarán vacíos.

Si **roifill** es llamado sin argumentos de salida, la imagen resultante es presentada en una nueva figura.

La imagen de entrada **I** puede ser de clase **uint8**, **uint16** o **double.** La máscara binaria de entrada, **BW**, puede ser de clase **numeric** o **logic**. La máscara de salida binaria, **BW**, es siempre de clase **logic**. La imagen de salida **J** es de la misma clase que **I**. Todas las otras entradas y salidas son de clase **double.**

Ejemplo

```
i = imread ( 'eight.tif' );

c = [ 222 272 300 270 221 194 ];

r = [ 21 21 75 121 75 ];

J = roifill [ I,c,r ];

imshow (I);

figure, imshow ( J )
```

A3. roifilt2

Filtra una región de interés.

Sintaxis

J = roifilt2 (h,I,BW)

J = roifilt2 (I,BW,fun)

J = roifilt2 (I,BW,fun,P1,p2,...)

Descripción

J = roifilt2 (h,I,BW) Filtra los datos en **I** con un filtro **h** lineal en dos dimensiones.

BW es una imagen binaria del mismo tamaño que **I** que es usada como una máscara para el filtrado. **roifilt2** proporciona una imagen que consiste de valores filtrados de pixeles en las posiciones donde **BW** contiene **1's**, y valores no filtrados de pixeles en las posiciones donde **BW** contiene **0's**. Para esta sintaxis, **roifilt2** llama a **filter2** para implementar el filtro.

J = roifilt2 (I,BW,fun) Procesa los datos en **I** usando la función **fun**. El **J** resultante contiene valores computados para pixels en posiciones donde **BW** contiene **1's** y valores actuales en **I** para pixeles en posiciones donde **BW** contiene **0's**.

fun puede ser una **function handle** creada usando @, o un objeto **inline. fun** considerará a una matriz como un argumento simple y entregará una matriz del mismo tamaño.

y = fun (x)

J = roifilt2 (I,BW,fun,P1,P2, ...) Pasa los parámetros adicionales P1,P2,...a **fun**

Para la sintaxis que incluye al filtro **h**, la imagen de entrada puede ser de clase **uint8**, **uint16** o **double,** y la matriz de salida, **J**, es de la misma clase que la imagen de entrada. Para la sintaxis que incluye la función, **I** puede ser de cualquier clase soportada por **fun**, y la clase de **J** depende de la clase de la salida de **fun.**

Ejemplo

```
i = imread ( 'eight.tif' );

c = [ 222 272 300 270 221 194 ];

r = [ 21 21 75 121 75 ];

BW = roipoly ( I,c,r );

H = fespecial( 'unsharp' );

J = roifilt2 ( h,I,BW );

imshow ( J ), figure , imshow ( J )
```

A4. filter2

Filtrado digital de dos dimensiones.

Sintaxis

Y = filter2 (h,X)

Y = filter2 (h,X,shape)

Descripción

Y = filter2 (h,X) Filtra los datos en **X** con un filtro **FIR** de dos dimensiones en la matriz **h**. Computa los resultados, **Y**, usando correlación bidimensional y proporciona la parte central de la correlación que es del mismo tamaño de **X.**

Y = filter2 (h,X,shape) Proporciona la parte de Y especificada por el parámetro **shape. shape** es una cadena con alguno de los siguientes valores:

'full' Proporciona la correlación totalmente bidimensional. En este caso, **Y** es más grande que **X**.

'**same'** (por defecto) Proporciona la parte central de la correlación. En este caso, **Y** es del mismo tamaño que **X.**

'valid' Proporciona solamente aquellas partes de la correlación que están computadas sin codos zero-paddes. En este caso, **Y** es más pequeño que **X.**

La correlación de dos dimensiones es equivalente a la convolución bidimensional con la matriz del filtro girada 180 grados.

Dada una matriz **X** y un filtro **FIR h** bidimensional, **filter2** gira la matriz del filtro 180 grados para crear una convolución. Luego llama a **conv2**, la función convolución bidimensional para implementar la operación de filtrado.

filter2 usa **conv2** para computar la convolución total en dos dimensiones del filtro FIR con la matriz de entrada. Por defecto, **filter2** extrae la parte central de la convolución que es del mismo tamaño que la matriz de entrada, y entrega este resultado. Si el parámetro **shape** especifica una parte alterna de la convolución para el resultado, entonces **filter2** entregará dicha parte.

ANEXO B

LISTADO DEL PROGRAMA

A continuación se presenta el listado del programa **roidemo.m**, el cual genera el procedimiento que se estudia en este tratado.

Dicho programa se encuentra dentro del disco duro de una PC que tiene instalado el MATLAB versión 6.5 y en el siguiente path:

C:\MATLAB6P5\toolbox\images\roidemo

Para efectos de la ejecución del programa se ha utilizado una PC Pentium IV de 2.4 GHZ con 256 Mbytes de RAM , un almacenamiento en disco de 80 GB y una tarjeta de video TNT Vidia de 32 MB., sin embargo también hubo oportunidad de ejecutarlo en otras PC de menores características, notándose que de no ser por el tiempo de respuesta más bajo en estas últimas, los resultados del proceso no difieren.

C:\MATLAB6p5\toolbox\images\imdemos\roidemo.m

```
 1 function ud = roidemo(action, varargin)
 2 %ROIDEMO Region-of-interest processing demo.
 3 %
 4 %  This demo illustrates the use of region-of-interest (ROI)
 5 %  operations for processing selected portions of images.
 6 %
 7 %  Select an image from the menu on the left.  Move the pointer
 8 %  over the image on the left; notice that the cursor changes to
 9 %  a crosshair.  Click on points in the image to select vertices
10 %  of the region of interest. Select the final point with a
11 %  double-click, shift-click, or right-click. (You can also end
12 %  the selection without adding a vertex by pressing <enter> or
13 %  <return> on the keyboard.)  When you complete your selection,
14 %  the region will be displayed in the center as a binary image.
15 %
16 %  Select the operation to perform from the "Region of Interest
17 %  Operations" menu on the right, and press the "Apply" button.
18 %  The original image is filtered using the binary image as a
19 %  mask; the operation is applied only to the portion of the
20 %  original image which corresponds to the white area of the
21 %  mask.
22 %
23 %  When you select a new image, you may notice that the ROI
24 %  operation has changed also.  The images all have appropriate
25 %  default operations associated with them, but you can
26 %  experiment with different operations for each image.
27 %
28 %  For example, the image called "Aluminum and Noise" is an
29 %  image of aluminum grains corrupted with salt & pepper noise.
30 %  The default operation is "Median Filter", because median
31 %  filtering is especially effective at removing this type of
32 %  noise.
33 %
34 %  Another example is the "Blood" image, whose default operation
35 %  is "Boundary Interpolation".  This operation calls the
36 %  function ROIFILL, which smoothly interpolates within an
37 %  arbitrary region by solving Laplace's equation on the
38 %  interior of the region. Try selecting a polygon that entirely
39 %  surrounds one of the blood cells; this will effectively
40 %  remove the cell from the image.
```

```
41 %
42 %  The following images are courtesy of J. C. Russ, The Image Processing
43 %  Handbook, Third Edition, 1998, CRC Press: Flower, Blood, Circuit, Rice,
44 %  Saturn, Bone Marrow, and Aluminum.
45 %
46 %  See also ROIPOLY, ROIFILT2, ROIFILL.
47
48 %  Copyright 1993-2002 The MathWorks, Inc.
49 %  $Revision: 1.17 $  $Date: 2002/04/04 18:03:31 $
50
51 %  Function Subroutines:
52 %
53 %  InitializeROIDEMO - Set up controls and Images.
54 %
55 %  LoadNewImage  -  Load a new image from a mat-file and call
56 %                   ApplyOperation to do the ROI operation on it.
57 %
58 %  ApplyOperation - Look at the popup to see what operation we
59 %                   are doing and then do it to the original image
60 %                   only in the Region-of-Interest specified by
61 %                   the mask.
62 %
63 %  MyGetline -      A pirated version of Getline for this demo only.
64 %                   This version stores information in the figure's
65 %                   userdata structure instead of in global variables.
66 %                   This routine stores the first point and then sets
67 %                   the button down function and motion function for
68 %                   the remainder of the ROI selection process.
69 %
70 %  ButtonDown -     Button down function for adding new vertices of
71 %                   the polygonal ROI mask.
72 %
73 %  ButtonMotion -   Motion function for animating the polygon selection.
74 %
75 %  KeyPress -       What to do in case of a Key press on the keyboard
76 %                   during the ROI selection process.
77 %
78 %  CompleteGetline - Completes selection of ROI and returns the Figure
79 %                   window more or less to the state in which it was
80 %                   before MyGetline was called.   Resets the button
81 %                   down function, motion function, and key press fcn. It
82 %                   also calculates the mask using roipoly and displays it.
83 %
```

```
 84 %   UpdateOperation - Updates the ROI operation from the current value of
 85 %                    the operation popup menu.
 86 %
 87 %   NormalMotionFcn - This is the default motion function of the figure
 88 %                     window, it sets the pointer to an arrow when not over
 89 %                     the original image and sets it to a cross-hair when it
 90 %                     IS over the original image.
 91
 92
 93 if nargin<1,
 94   action='InitializeROIDEMO';
 95   if nargout>0  % If user wants the UserData, give it to them.
 96     ud = feval(action,varargin{:});
 97     return
 98   end
 99 end
100
101 feval(action,varargin{:})
102 return
103
104 %%%
105 %%%  Sub-function - InitializeROIDEMO
106 %%%
107
108 function udout = InitializeROIDEMO()
109
110 % If roidemo is already running, bring it to the foreground
111 h = findobj(allchild(0), 'tag', 'Region-of-Interest Processing Demo');
112 if ~isempty(h)
113   figure(h(1))
114   if nargout>0
115     udout = get(h(1), 'UserData');
116   end
117   return
118 end
119
120 screenD = get(0, 'ScreenDepth');
121 if screenD>8
122   grayres=256;
123 else
124   grayres=128;
125 end
126
```

```
127
128 RoiDemoFig = figure( ...
129    'Name','Region-of-Interest Processing Demo', ...
130    'NumberTitle','off', 'HandleVisibility', 'on', ...
131    'tag', 'Region-of-Interest Processing Demo', ...
132    'Visible','off', 'Resize', 'off',...
133    'BusyAction','Queue','Interruptible','off', ...
134    'Color', [.8 .8 .8], 'IntegerHandle', 'off', ......
135    'WindowButtonMotionFcn', 'roidemo(''NormalMotionFcn'');', ...
136    'DoubleBuffer', 'on', 'Colormap', gray(grayres));
137
138 ud.FigureHandle = RoiDemoFig;
139
140 figpos = get(RoiDemoFig, 'position');
141 % Adjust the size of the figure window
142 figpos(3:4) = [560 420];
143 horizDecorations = 10;  % resize controls, etc.
144 vertDecorations = 45;   % title bar, etc.
145 screenSize = get(0,'ScreenSize');
146 if (screenSize(3) <= 1)
147    % No display connected (apparently)
148    screenSize(3:4) = [100000 100000]; % don't use Inf because of vms
149 end
150 if (((figpos(3) + horizDecorations) > screenSize(3)) | ...
151          ((figpos(4) + vertDecorations) > screenSize(4)))
152    % Screen size is too small for this demo!
153    delete(fig);
154    error(['Screen resolution is too low ', ...
155             '(or text fonts are too big) to run this demo']);
156 end
157 dx = screenSize(3) - figpos(1) - figpos(3) - horizDecorations;
158 dy = screenSize(4) - figpos(2) - figpos(4) - vertDecorations;
159 if (dx < 0)
160    figpos(1) = max(5,figpos(1) + dx);
161 end
162 if (dy < 0)
163    figpos(2) = max(5,figpos(2) + dy);
164 end
165 set(RoiDemoFig, 'position', figpos);
166
167 rows = figpos(4); cols = figpos(3);
168
169 % Colors
```

```
170 bgcolor = [0.45 0.45 0.45];  % Background color for frames
171 wdcolor = [.8 .8 .8];  % Window color
172 fgcolor = [1 1 1];  % For text
173
174 hs = (cols-(3*128)) / 4;       % Horizantal Spacing
175 vs = hs/2;                     % Vertical Spacing
176 ifs = hs/4;                    % Intraframe spacing
177
178
179 %===================================
180 % Parameters for all buttons and menus
181
182 Std.Interruptible = 'off';
183 Std.BusyAction = 'queue';
184
185 % Defaults for image axes
186 Ax = Std;
187 Ax.Units = 'Pixels';
188 Ax.Parent = RoiDemoFig;
189 Ax.ydir = 'reverse';
190 Ax.XLim = [.5 128.5];
191 Ax.YLim = [.5 128.5];
192 Ax.CLim = [0 1];
193 Ax.XTick = [];
194 Ax.YTick = [];
195
196 Img = Std;
197 Img.CData = [];
198 Img.Xdata = [1 128];
199 Img.Ydata = [1 128];
200 Img.CDataMapping = 'Scaled';
201  Img.Erasemode = 'none';
202
203 Ctl = Std;
204 Ctl.Units = 'Pixels';
205 Ctl.Parent = RoiDemoFig;
206
207 Frame = Ctl;
208 Frame.Style = 'Frame';
209
210 Btn = Ctl;
211 Btn.Style = 'pushbutton';
212 Btn.Enable = 'off';
```

```
213
214 Edit = Ctl;
215 Edit.Style = 'edit';
216 Edit.HorizontalAlignment = 'right';
217 Edit.BackgroundColor = 'white';
218 Edit.ForegroundColor = 'black';
219
220 Menu = Ctl;
221 Menu.Style = 'Popupmenu';
222
223 Text = Ctl;
224 Text.Style = 'text';
225 Text.HorizontalAlignment = 'left';
226 Text.BackgroundColor = bgcolor;
227 Text.ForegroundColor = fgcolor;
228
229 btnHt = 26;
230 txtHt = 18;
231 menuHt = 26;
232 editHt = 21;
233
234 imageBot = rows-hs-128-editHt;
235 %================================
236 % Original Image
237 ud.h.OriginalAxes = axes(Ax, ...
238    'Position', [hs imageBot 128 128]);
239 % title('Original Image');
240 ud.h.OriginalImage = image(Img, ...
241    'Parent', ud.h.OriginalAxes, ...
242    'ButtonDownFcn', 'roidemo(''MyGetline'')');
243
244 %================================
245 % Mask Image
246 ud.h.MaskAxes = axes(Ax, ...
247    'Position', [2*hs+128 imageBot 128 128]);
248 title('Region of Interest');
249 ud.h.MaskImage = image(Img, ...
250    'Parent', ud.h.MaskAxes);
251 def_x =  [ 9  124   96    9];
252 def_y =  [ 108  3  126  108];
253 ud.DefaultMask = roipoly(128, 128, def_x, def_y);
254 set(ud.h.MaskImage, 'CData', ud.DefaultMask);
255
```

```
256 %---------=========================
257 % Output Image
258 ud.h.OutputAxes = axes(Ax, ...
259    'Position', [3*hs+256 imageBot 128 128]);
260 title('Output Image');
261 ud.h.OutputImage = image(Img, ...
262    'Parent', ud.h.OutputAxes);
263
264 %================================--
265 %  The frame
266 fleft = ifs;
267 fbot = vs*1.5;
268 fwid = cols - 2*hs/3;
269 fht = imageBot-vs-fbot;
270 ud.h.ControlFrame = uicontrol(Frame, ...
271    'Position', [fleft fbot fwid fht], ...
272    'BackgroundColor', bgcolor);
273
274 menuWid = (fwid)/3;
275 menuBot = fbot+fht-vs-menuHt;
276 labelBot = fbot+fht-22;    % For the labels above the three top menus
277 labelWid = menuWid/2;      % For the labels in front of edit boxes
278 btnWid = 128;
279
280 %================================
281 % Image popup menu
282 ipleft = fleft+ifs;
283 ipbot = imageBot+128+vs*.5;
284 ud.h.ImgPop = uicontrol(Menu, ...
285    'Position',[ipleft ipbot (fwid-4*ifs)/3 menuHt], ...
286    'String',['Moon|Flower|Blood|Circuit|Rice|Pout' ...
287       '|Trees|Saturn|Quarter|Bone Marrow|Aluminum and Noise'], ...
288    'Callback','roidemo(''LoadNewImage'')');
289 % Text label for Image Menu Popup
290 uicontrol( Text, ...
291    'Position',[ipleft ipbot+menuHt menuWid txtHt], ...
292    'Background', wdcolor, ...
293    'Foreground', 'black', ...
294    'String','Select an Image:');
295
296 %================================
297 % Operations Menu:
298 opleft = fleft+fwid-menuWid-ifs;
```

```
299 ud.h.OperationsPop = uicontrol(Menu, ...
300    'Position',[opleft menuBot menuWid menuHt], ...
301    'String',...
302    {  'Unsharp Masking', ...
303       'Histogram Equalization', ...
304       'Lowpass Filter', ...
305       'Median Filter', ...
306       'Brighten', 'Darken', ...
307       'Increase Contrast', ...
308       'Decrease Contrast', ...
309       'Boundary Interpolation', ...
310       'Add Gaussian Noise' }, ...
311    'Callback','roidemo(''UpdateOperation'')');
312 % Text label for Image Menu Popup
313 uicontrol( Text, ...
314    'Position',[opleft labelBot menuWid txtHt], ...
315    'String','Region of Interest Operations:');
316
317 %====================================
318 % Buttons - Apply, Info and Close
319 ud.h.Close=uicontrol(Btn, ...
320    'Position',[opleft fbot+ifs menuWid btnHt], ...
321    'String','Close', ...
322    'Callback','close(gcbf)');
323
324 ud.h.Info=uicontrol(Btn, ...
325    'Position',[opleft fbot+2*ifs+btnHt menuWid btnHt], ...
326    'String','Info', ...
327    'Callback','helpwin roidemo');
328
329 ud.h.Apply=uicontrol(Btn, ...
330    'Position',[opleft fbot+3*ifs+2*btnHt menuWid btnHt], ...
331    'String','Apply', ...
332    'Callback','roidemo(''ApplyOperation'')');
333
334
335 %====================================
336 % Frame for the text
337 ftleft = ipleft;
338 ftbot = fbot+ifs;
339 ftwid = 256;
340 ftht = fht - 2*ifs;
341 ud.h.TextFrame = uicontrol(Frame, ...
```

```
342    'Position', [ftleft ftbot ftwid ftht], ...
343    'BackgroundColor', [.9 .9 .9]);
344  ud.h.ExplanatoryText = uicontrol( Text, ...
345    'Position',[ftleft+8 ftbot+8 ftwid-16 ftht-16], ...
346    'BackgroundColor', [.9 .9 .9], ...
347    'ForegroundColor', [0 0 0], ...
348    'HorizontalAlignment', 'Left', ...
349    'String',...
350    [  'Specify a region of interest by clicking on the ' ...
351       'image on the left to select vertices of a polygon.  ' ...
352       'Each click adds another vertex.  Shift-click (or Right-click) to ' ...
353       'select the final vertex and display the region.']);
354
355  %==================================
356  % Status bar
357  ud.h.Status = uicontrol( Text, ...
358    'Background', wdcolor, ...
359    'Foreground', [.8 0 0], ...
360    'Position',[0 2 cols txtHt], ...
361    'Horiz','center', ...
362    'Tag', 'Status', ...
363    'String','Initializing ROIDEMO...');
364
365  ud.GetlineX = [];
366  ud.GetlineY = [];
367
368  set(RoiDemoFig, 'Userdata', ud);
369  set(RoiDemoFig, 'visible','on','HandleVisibility','callback');
370  LoadNewImage(RoiDemoFig);
371  ApplyOperation(RoiDemoFig);
372  set([ud.h.Info ud.h.Close ud.h.Apply], 'Enable', 'on');
373  if nargout>0
374     udout = ud;
375  end
376  return
377
378
379
380
381  %%%
382  %%%  Sub-Function - MyGetline
383  %%%
384
```

```
385 function MyGetline
386 % Pirated from the Image Processing Toolbox function getline.m,
387 % this version is specific to this demo.  First button down function.
388
389 DemoFig = gcbf;
390 ud = get(DemoFig, 'UserData');
391
392 pt = get(gca, 'CurrentPoint');
393 startingPoint = pt(1,1:2);
394 ud.GetlineX = [pt(1,1) pt(1,1)];
395 ud.GetlineY = [pt(1,2) pt(1,2)];
396
397 set(DemoFig, 'Pointer', 'crosshair');
398 set(DemoFig, 'WindowButtonDownFcn', 'roidemo(''ButtonDown'');');
399 set(DemoFig, 'KeyPressFcn', 'roidemo(''KeyPress'');');
400
401 % Initialize the lines to be used for the drag
402 ud.GetlineH1 = line('XData', ud.GetlineX, ...
403    'YData', ud.GetlineY, ...
404    'Clipping', 'on', ...
405    'Color', 'k', ...
406    'LineStyle', '-', ...
407    'EraseMode', 'xor');
408
409 ud.GetlineH2 = line('XData', ud.GetlineX, ...
410    'YData', ud.GetlineY, ...
411    'Clipping', 'on', ...
412    'Color', 'w', ...
413    'LineStyle', '--', ...
414    'EraseMode', 'xor');
415
416 set(DemoFig, 'UserData', ud);
417 if ~strcmp(get(DemoFig, 'SelectionType'), 'normal')
418    % We're done!
419    CompleteGetline(DemoFig);
420 else
421    % Let the motion functions take over:
422    set(DemoFig, 'WindowButtonMotionFcn', 'roidemo(''ButtonMotion'');', ...
423       'WindowButtonDownFcn', 'roidemo(''ButtonDown'');');
424    % And so that we don't start MyGetline again:
425    set(ud.h.OriginalImage, 'ButtonDownFcn', '');
426 end
427
```

```
428
429
430 %%%
431 %%%  Sub-Function - ButtonDown
432 %%%
433
434 function ButtonDown
435 DemoFig = gcbf;
436 ud = get(DemoFig, 'UserData');
437
438 selectionType = get(DemoFig, 'SelectionType');
439 if ~strcmp(selectionType, 'open')
440    % We don't want to add a point on the second click
441    % of a double-click
442
443    pt2 = get(ud.h.OriginalAxes, 'CurrentPoint');
444    if ~isempty(ud.GetlineX)
445       ud.GetlineX = [ud.GetlineX(1:end-1) pt2(1,1) ud.GetlineX(end)];
446       ud.GetlineY = [ud.GetlineY(1:end-1) pt2(1,2) ud.GetlineY(end)];
447    else
448       ud.GetlineX = pt2(1,1);
449       ud.GetlineY = pt2(1,2);
450    end
451
452    set([ud.GetlineH1 ud.GetlineH2], 'XData', ud.GetlineX, ...
453       'YData', ud.GetlineY);
454
455 end
456 set(DemoFig, 'UserData', ud);
457
458 if (~strcmp(get(DemoFig, 'SelectionType'), 'normal'))
459    % We're done!
460    CompleteGetline(DemoFig);
461 end
462
463
464
465 %%%
466 %%%  Sub-Function - ButtonMotion
467 %%%
468
469 function ButtonMotion
470 DemoFig = gcbf;
```

```
471 ud = get(DemoFig, 'UserData');
472
473 pt2 = get(ud.h.OriginalAxes, 'CurrentPoint');
474 if length(ud.GetlineX) >= 3
475    x = [ud.GetlineX(1:end-1) pt2(1,1) ud.GetlineX(end)];
476    y = [ud.GetlineY(1:end-1) pt2(1,2) ud.GetlineY(end)];
477 else
478    x = [ud.GetlineX pt2(1,1)];
479    y = [ud.GetlineY pt2(1,2)];
480 end
481
482 set([ud.GetlineH1 ud.GetlineH2], 'XData', x, 'YData', y);
483
484
485
486 %%%
487 %%%  Sub-Function - KeyPress
488 %%%
489
490 function KeyPress
491 DemoFig = gcbf;
492 ud = get(DemoFig, 'UserData');
493
494 key = real(get(DemoFig, 'CurrentCharacter'));
495
496 if isempty(key), % switch statement was triggering error in Unix, the
497    return;       % "Shift" key was causing the variable key to be []
498 end
499
500 switch key
501 case {8, 127}  % delete and backspace keys
502    % remove the previously selected point
503    switch length(ud.GetlineX)
504    case 0
505       % nothing to do
506    case 1
507       ud.GetlineX = [];
508       ud.GetlineY = [];
509       % remove point and start over
510       set([ud.GetlineH1 ud.GetlineH2], ...
511          'XData', ud.GetlineX, ...
512          'YData', ud.GetlineY);
513       set(ud.h.OriginalImage, 'ButtonDownFcn', ...
```

```
514         'roidemo(''MyGetline'');');
515       set(DemoFig, 'WindowButtonMotionFcn', 'roidemo(''NormalMotionFcn'');',...
516         'WindowButtonDownFcn', '');
517
518     otherwise
519       % remove last point
520       ud.GetlineX(end-1) = [];
521       ud.GetlineY(end-1) = [];
522       set([ud.GetlineH1 ud.GetlineH2], ...
523         'XData', ud.GetlineX, ...
524         'YData', ud.GetlineY);
525     end
526
527 case {13, 3}   % enter and return keys
528     % We're done
529     CompleteGetline(DemoFig);
530 end
531
532 set(DemoFig, 'UserData', ud);
533
534
535
536 %%%
537 %%%  Sub-Function - CompleteGetline
538 %%%
539
540 function CompleteGetline(DemoFig)
541
542 ud = get(DemoFig, 'UserData');
543
544 % Get the rect info and call roipoly to create the mask
545 mask = roipoly(128,128,ud.GetlineX,ud.GetlineY);
546 set(ud.h.MaskImage,'Cdata', mask);
547 set(ud.h.Apply, 'Enable', 'on');
548
549 % Clean up after getline and prepare it for the next MyGetline Callback
550 set(DemoFig, 'Pointer', 'arrow');
551 ud.GetlineX = [];
552 ud.GetlineY = [];
553
554 % Delete the animation objects
555 if (ishandle(ud.GetlineH1))
556     delete(ud.GetlineH1);
```

```
557 end
558 if (ishandle(ud.GetlineH2))
559     delete(ud.GetlineH2);
560 end
561
562 % Reset the Figure's Motion and Button Down functions and Key Press Fcn
563 set(DemoFig, 'WindowButtonMotionFcn', 'roidemo(''NormalMotionFcn'');', ...
564    'WindowButtonDownFcn', '', 'UserData', ud, 'KeyPressFcn', '');
565
566 % Reset the ButtonDownFunction of the Image
567 set(ud.h.OriginalImage, 'ButtonDownFcn', 'roidemo(''MyGetline'')');
568 setstatus(DemoFig, 'Press "Apply" to perform the operation on the region of interest.');
569
570
571
572 %%%
573 %%%  Sub-Function - UpdateOperation
574 %%%
575
576 function UpdateOperation(DemoFig)
577
578 if nargin<1
579    DemoFig = gcbf;
580 end
581
582 ud = get(DemoFig, 'UserData');
583 opval = get(ud.h.OperationsPop, 'value');
584 opstr = get(ud.h.OperationsPop, 'string');
585 ud.Operation = opstr{opval};
586 set(DemoFig, 'UserData', ud);
587 set(ud.h.Apply, 'Enable', 'on');
588 setstatus(DemoFig, 'Press "Apply" to perform the operation on the region of interest.');
589
590 %%%
591 %%%  Sub-Function - ApplyOperation
592 %%%
593
594 function ApplyOperation(DemoFig)
595
596 if nargin<1
597    DemoFig = gcbf;
598 end
599
```

```
600 setptr(DemoFig, 'watch');
601 ud = get(DemoFig, 'UserData');
602 orig = get(ud.h.OriginalImage, 'Cdata');
603 mask = logical(get(ud.h.MaskImage, 'Cdata'));
604 result = orig;
605 setstatus(DemoFig, ['Applying ' lower(ud.Operation) ' operation...']);
606
607 switch ud.Operation
608 case 'Histogram Equalization'
609    J = histeq(orig(mask));
610    result(mask) = J;
611
612 case 'Lowpass Filter'
613    order = 15;
614    cutoff = 0.3;
615    [f1,f2] = freqspace(order,'meshgrid');
616    d = find(f1.^2+f2.^2 < cutoff^2);
617    Hd = zeros(order);
618    Hd(d) = 1;
619    % Use hanning(15) as the window.  The window coefficients
620    % are inlined to remove dependency on the Signal Processing
621    % Toolbox.
622    h = fwind1(Hd, ...
623           [0.0381 0.1464 0.3087 0.5000 0.6913 0.8536 ...
624            0.9619 1.0000 0.9619 0.8536 0.6913 0.5000 ...
625            0.3087 0.1464 0.0381]);
626    result = roifilt2(h, orig, mask);
627
628 case 'Unsharp Masking'
629    h = fspecial('unsharp');
630    result = roifilt2(h, orig, mask);
631
632 case 'Median Filter'
633    med_orig = medfilt2(orig,[5 5]);
634    result(mask) = med_orig(mask);
635
636 case 'Brighten'
637    bright_orig = imadjust(orig, [], [.25 1]);
638    result(mask) = bright_orig(mask);
639
640 case 'Darken'
641    dark_orig = imadjust(orig, [], [0 .75]);
642    result(mask) = dark_orig(mask);
```

```
643
644 case 'Increase Contrast'
645    ic_orig = imadjust(orig, [.25 .75], []);
646    result(mask) = ic_orig(mask);
647
648 case 'Decrease Contrast'
649    dc_orig = imadjust(orig, [], [.25 .75]);
650    result(mask) = dc_orig(mask);
651
652 case 'Boundary Interpolation'
653    result = roifill(orig, mask);
654
655 case 'Add Gaussian Noise'
656    noisy_orig = imnoise(orig, 'Gaussian', 0, .01);
657    result(mask) = noisy_orig(mask);
658
659 otherwise
660    warning('Invalid ROIDEMO operation');
661    result = repmat(uint8(0),128,128);
662 end
663
664 set(ud.h.OutputImage, 'Cdata', result);
665 set(ud.h.Apply, 'Enable', 'off');
666 setstatus(DemoFig, '');
667 setptr(DemoFig, 'arrow');
668 return
669
670 %%%
671 %%%  Sub-Function - LoadNewImage
672 %%%
673
674 function LoadNewImage(DemoFig)
675
676 callb = 0;
677 if nargin<1
678    DemoFig = gcbf;
679    callb = 1;   % We are in a callback
680 end
681
682 set(DemoFig,'Pointer','watch');
683 ud=get(DemoFig,'Userdata');
684 v = get(ud.h.ImgPop,{'value','String'});
685 name = deblank(v{2}(v{1},:));
```

```
686 drawnow
687
688 switch name
689 case 'Blood'
690    blood = [];
691    load imdemos blood
692    img = blood;
693    set(ud.h.OperationsPop, 'Value', 9);
694 case 'Saturn',
695    saturn = [];
696    load imdemos saturn
697    img = saturn;
698    set(ud.h.OperationsPop, 'Value', 1);
699 case 'Pout',
700    pout = [];
701    load imdemos pout
702    img = pout;
703    set(ud.h.OperationsPop, 'Value', 2);
704 case 'Quarter',
705    quarter = [];
706    load imdemos quarter
707    img = quarter;
708    set(ud.h.OperationsPop, 'Value', 1);
709 case 'Circuit',
710    circuit2 = [];
711    load imdemos circuit2
712    img = circuit2;
713    set(ud.h.OperationsPop, 'Value', 1);
714 case 'Aluminum and Noise',
715    alumgrns = [];
716    load imdemos alumgrns
717    img = imnoise(alumgrns,'Salt & Pepper', .05);
718    set(ud.h.OperationsPop, 'Value', 4);
719 case 'Trees',
720    trees = [];
721    load imdemos trees
722    img = trees;
723    set(ud.h.OperationsPop, 'Value', 10);
724 case 'Flower',
725    flower = [];
726    load imdemos flower
727    img = flower;
728    set(ud.h.OperationsPop, 'Value', 7);
```

```
729 case 'Rice',
730    rice = [];
731    load imdemos rice
732    img = rice;
733    set(ud.h.OperationsPop, 'Value', 3);
734 case 'Moon',
735    moon = [];
736    load imdemos moon
737    img = moon;
738    set(ud.h.OperationsPop, 'Value', 1);
739 case 'Bone Marrow',
740    bonemarr = [];
741    load imdemos bonemarr
742    img = bonemarr;
743    set(ud.h.OperationsPop, 'Value', 5);
744 otherwise
745    error('roidemo: Unknown Image Option!');
746 end
747
748 UpdateOperation(DemoFig);
749 img = double(img)/255;
750 set(ud.h.OriginalImage, 'Cdata', img);
751 if callb
752    set(ud.h.Apply, 'Enable', 'on');
753 end
754 set(DemoFig, 'Pointer', 'arrow');
755 setstatus(DemoFig, 'Press "Apply" to perform the operation on the region of interest.');
756
757
758
759 %%%
760 %%%  Sub-Function - NormalMotionFcn
761 %%%
762
763 function NormalMotionFcn
764 % Set the cursor to a Cross-Hair when above the Original image, and back
765 % to an arrow when not.
766 % This is the normal motion function for the window when we are not in
767 % a MyGetline selection state.
768
769 DemoFig = gcbf;
770 ud = get(DemoFig, 'Userdata');
771 pos = get(ud.h.OriginalAxes, 'Position');
```

```
772
773 pt = get(DemoFig, 'CurrentPoint');
774 x = pt(1,1);
775 y = pt(1,2);
776
777 if (x>pos(1) & x<pos(1)+pos(3) & y>pos(2) & y<pos(2)+pos(4))
778    set(DemoFig, 'Pointer', 'cross');
779 else
780    set(DemoFig, 'Pointer', 'arrow');
781 end
```

BIBLIOGRAFÍA

1. "Base del Procesamiento de Imágenes" Rubén Medina - Jesús Bellera Universidad de los Andes Facultad de Ingeniería – Venezuela
2. Laboratorios de Procesamiento Digital de Imágenes Grupo de Investigación en Señales, Telemática y Comunicaciones Universidad de Granada – España
3. ToolBoxes – Image Processing – Enhancement – Region of interest Processing - MATLAB The Language of Thecnical Computing
4. Aprenda MATLAB 6.0 como si estuviera en primero - Escuela Superior de Ingenieros Industriales Universidad de Navarra - España
5. "Fundamentals of Image Processing" Collins S y Skorton D.
6. The Image Processing Handbook - Third Edition 1998 - J.C. Russell