

BORDES Y CONTORNOS

Características en una imagen tales como bordes y contornos, los cuales son detectados a través de cambios locales de intensidad o de color, juegan un papel importante en la interpretación de imágenes. La subjetiva “claridad” de una imagen se encuentra en relación directa con las discontinuidades y nitidez de las estructuras presentes en la misma. El ojo humano da un peso importante a los bordes de los objetos, tal que sencillos trazos en imágenes son suficientes para interpretar las clases de los objetos presentes. Por esta razón son los bordes y los contornos un tema muy importante para el procesamiento de imágenes digitales y la visión artificial. En este capítulo se hace un tratamiento de los principales métodos existentes para la localización de bordes.

6.1 ¿COMO SE PRODUCEN LOS CONTORNOS?

Los Bordes tienen un predominante rol en la visión humana y probablemente también en otros sistemas biológicos de visión. Los bordes no solamente son notables, sino también es posible mediante pocas líneas de bordes reconstruir nuevamente a los objetos (Figura 6.1). Pero de todo esto, ¿como se originan estos bordes y como técnicamente es posible localizarlos en las imágenes?

Los bordes a grosso modo pueden ser considerados como puntos en una imagen en los cuales la intensidad en una determinada dirección cambia drásticamente. Dependiendo del cambio presentado en la intensidad será el valor del borde en la imagen para ese punto. El tamaño del cambio es calculado normalmente a partir de la derivada, y es utilizada como uno de los enfoques más importantes para la determinación de los bordes en una imagen.



(a)



(b)

Figura 6.1 Imagen original (a) e imagen con los bordes.

6.2 DETECCIÓN DE BORDES UTILIZANDO TÉCNICAS BASADAS EN EL GRADIENTE.

Por facilidad consideremos una sola dimensión y tomemos como ejemplo una imagen que tenga una región blanca en el centro rodeada de un fondo oscuro (Figura 6.2(a)).

El perfil en escala de grises a lo largo de una línea de la imagen podría verse como lo muestra la Figura 6.2(b). Definiremos esta señal unidimensional como $f(u)$ y calculamos a partir de ella su primera derivada

$$f'(u) = \frac{df}{du}(u), \quad (6.1)$$

así se produce una elevación positiva en todo lugar donde la intensidad aumente y una negativa donde la intensidad disminuya. Sin embargo la derivada no está definida para funciones discretas como $f(u)$ por lo que necesitamos un método para calcularla.

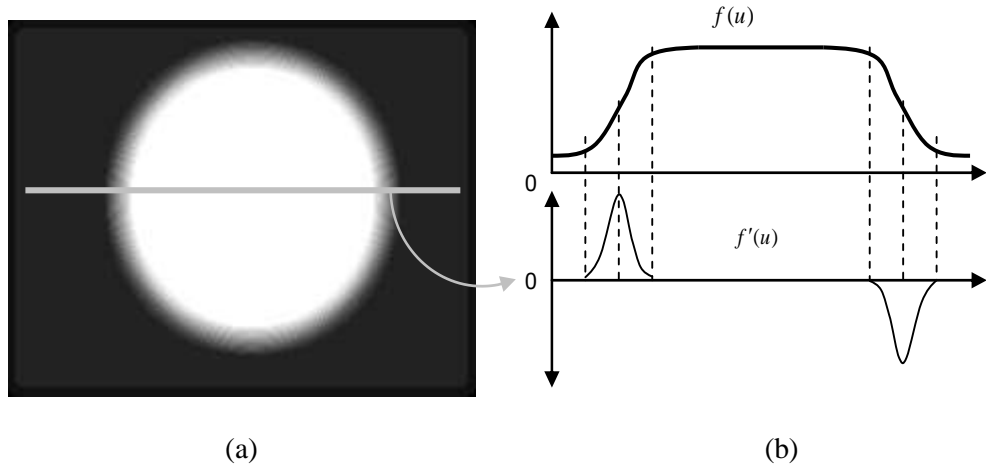


Figura 6.2 Primera derivada en el caso unidimensional obtenido a partir del perfil horizontal de la imagen. Imagen original (a) y la derivada $f'(u)$ del perfil horizontal obtenido de la imagen.

Es conocido que la derivada de una función continua en un punto x puede ser interpretada por la pendiente de la tangente en ese punto en particular. Sin embargo para una función discreta la derivada en un punto u (la pendiente de la tangente a ese punto) puede ser calculada a partir de la diferencia existente entre los puntos vecinos a u dividido por el valor de muestreo entre ambos puntos (Figura 6.3). Por lo que la derivada puede ser aproximada por

$$\begin{aligned}\frac{df}{du}(u) &\approx \frac{f(u+1) - f(u-1)}{2} \\ &= 0.5 \cdot (f(u+1) - f(u-1))\end{aligned}\tag{6.2}$$

El mismo proceso puede ser también llevado a cabo para el sentido vertical a lo largo de la columna de la imagen.

6.2.1 Derivada parcial y Gradiente

La derivada parcial puede ser considerada como la derivada de una función multidimensional a lo largo de un eje coordenado (con respecto a una de las variables de la función), por ejemplo

$$\frac{\partial I}{\partial x}(x, y) \text{ y } \frac{\partial I}{\partial y}(x, y)\tag{6.3}$$

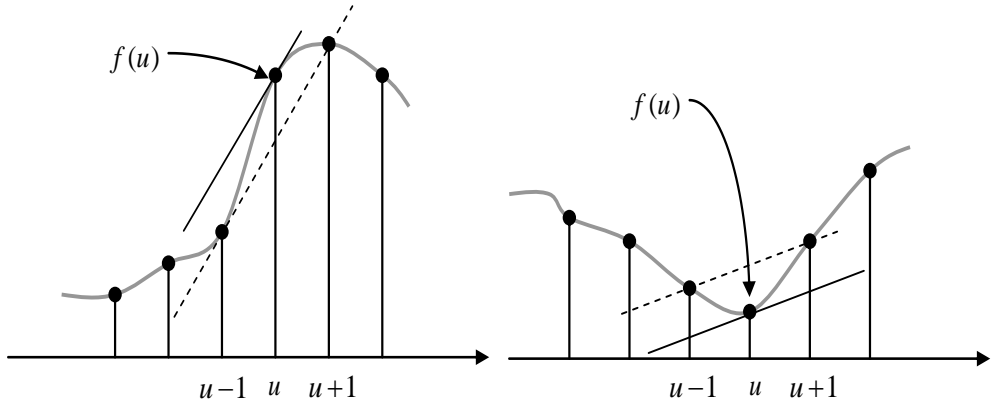


Figura 6.3 Cálculo de la primera derivada para una función discreta. La pendiente de la línea a través de los puntos vecinos $f(u-1)$ y $f(u+1)$ sirve como cálculo indirecto de la pendiente a la tangente en $f(u)$. El cálculo es lo suficientemente aproximado en la mayoría de los casos que pudieran presentarse.

Expresa la derivada parcial de la función de la imagen $I(u,v)$ con respecto a la variable u o v . El vector

$$\nabla I(x, y) = \begin{bmatrix} \frac{\partial I}{\partial x}(x, y) \\ \frac{\partial I}{\partial y}(x, y) \end{bmatrix} \quad (6.4)$$

Representa el vector del gradiente de la función I en el punto (x, y) . El valor del gradiente

$$|\nabla I| = \sqrt{\left(\frac{\partial I}{\partial x}\right)^2 + \left(\frac{\partial I}{\partial y}\right)^2} \quad (6.5)$$

Es invariante a rotaciones de la imagen y con ello también independiente de la orientación de las estructuras de contenidas en la misma. Esta propiedad es importante para la localización de los puntos bordes de la imagen con ello es el valor de $|\nabla I|$ el valor práctico utilizado en la mayoría de los algoritmos utilizados para la detección de bordes.

6.2.2 El filtro derivada

Las componentes del gradiente de la ecuación 6.4 no son otra cosa que la primera derivada tanto en el sentido de los renglones como en el de las columnas de la imagen. Tal y como fue mostrado en la ecuación 6.2 y Figura 6.3, la forma de calcular la derivada en el sentido horizontal es posible a partir de esto expresarlo monolíticamente en un filtro con la siguiente matriz de coeficientes

$$H_x^D = \begin{bmatrix} -0.5 & 0 & 0.5 \end{bmatrix} = 0.5 \cdot \begin{bmatrix} -1 & 0 & 1 \end{bmatrix} \quad (6.6)$$

Donde el coeficiente -0.5 afecta al píxel $I(x-1, y)$ y 0.5 al píxel $I(x+1, y)$. El valor del píxel de en medio $I(x, y)$ es multiplicado por cero o de igual manera ignorado (aquí se considera que el elemento bajo el cual esta el guión bajo “_” representa el punto de referencia o bien el píxel bajo el cual se calcula la propiedad en cuestión). De manera igual se puede establecer el mismo efecto del filtro pero ahora en el sentido vertical, siendo su matriz de coeficientes

$$H_y^D = \begin{bmatrix} -0.5 \\ 0 \\ 0.5 \end{bmatrix} = 0.5 \cdot \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} \quad (6.7)$$

La figura 6.4 muestra el efecto de la aplicación de los filtros definidos en las ecuaciones 6.6 y 6.7 sobre una imagen. La dependencia en la dirección que presenta en la Figura 6.4 es fácil de reconocer. El filtro del gradiente horizontal H_x^D reacciona dando como resultado una respuesta más grande en el sentido horizontal, resaltando los bordes en el sentido vertical (Figura 6.4(b)), de igual manera el filtro H_y^D actúa originando valores particularmente grandes en el sentido vertical, realzando de esa manera los bordes horizontales de una estructura (Figura 6.4(c)). En las regiones de la imagen donde la respuesta del filtro fue nula (en las Figuras 6.4(b) y 6.4(c)) los valores fueron representados por píxeles grises.

6.3 FILTROS PARA LA DETECCIÓN DE BORDES

La forma de calcular el gradiente local correspondiente a cada píxel de la imagen es lo que fundamentalmente diferencia a cada uno de los diferentes operadores para la detección de bordes. Ellos se diferencian principalmente en la forma en como se calculan el gradiente en las diferentes componentes direccionales, así como en la forma en como esos resultados parciales son unidos en uno final. En varios métodos se está interesado no únicamente en el valor del gradiente en un píxel que pertenece a un borde sino también en la dirección del borde en cuestión. Sin embargo como ambas informaciones (valor del gradiente y dirección) están implícitas en el cálculo

del gradiente es posible obtenerlas fácilmente. A continuación serán presentados algunos de los operadores de bordes más conocidos ya sea por su aplicación práctica o bien por haber sido históricamente interesantes.

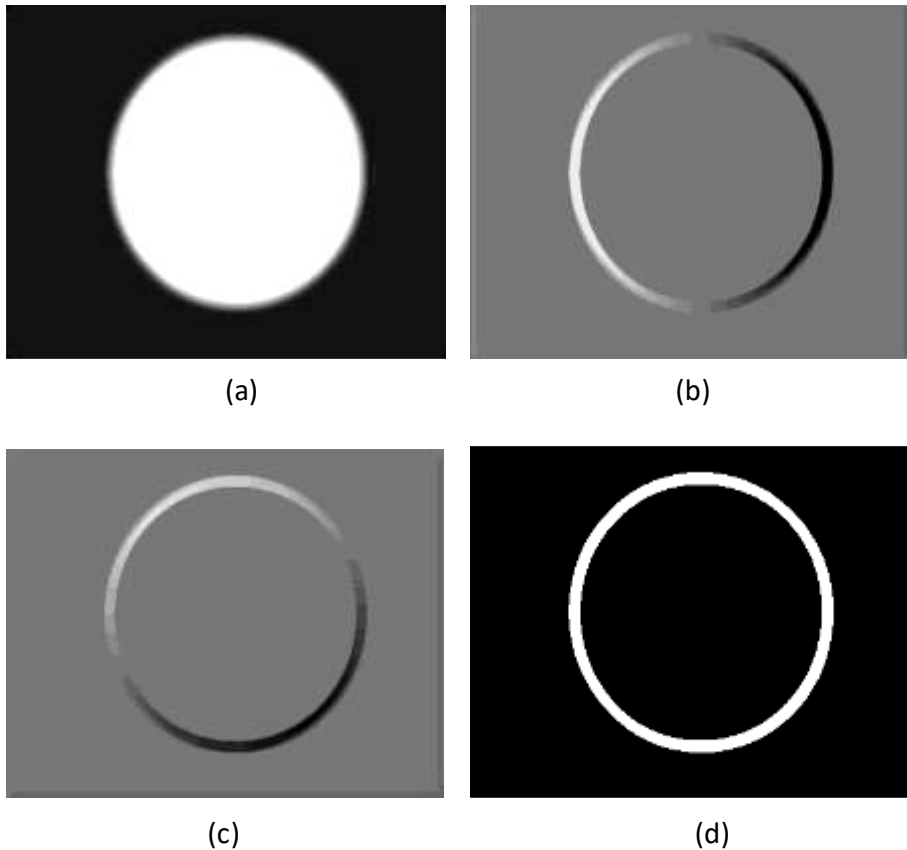


Figura 6.4 La primera derivada parcial. (a) Imagen sintética, (b) primera derivada parcial sobre el sentido horizontal $\partial I / \partial u$ y (c) sobre el sentido vertical $\partial I / \partial v$. (d) Valor del Gradiente $|\nabla I|$. En (b) y (c) son los valores oscuros pertenecientes a valores negativos, los valores más brillantes pertenecientes a valores positivos mientras que el gris corresponde al cero.

6.3.1 Los operadores Prewitt y Sobel

Los operadores Prewitt y Sobel representan dos de los métodos mas usados en la detección de bordes, los cuales son muy similares entre si diferenciándose solamente por algunas particularidades.

6.3.1.1 EL FILTRO

Ambos operadores utilizan como filtro una matriz de coeficientes de 3x3, que facilita la posibilidad de configurarlo de tal forma que el filtro no sea tan vulnerable al ruido propio de la imagen, en comparación a los filtros presentados en las ecuaciones 6.6 y 6.7. El operador Prewitt utiliza el filtro definido por

$$H_x^P = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \text{ y } H_y^P = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \quad (6.8)$$

El cual es evidentemente aplicado sobre los diferentes vecinos del píxel en cuestión. Si se analiza el filtro en su forma separada

$$H_x^P = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} * \begin{bmatrix} -1 & 0 & -1 \end{bmatrix} \text{ o bien } H_y^P = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} * \begin{bmatrix} -1 \\ 0 \\ -1 \end{bmatrix} \quad (6.9)$$

sería claro que cada uno de los vectores de la derecha ya sea para el caso de H_x^P o H_y^P produce tres columnas o renglones que constituyen los coeficientes del filtro definido en la ecuación 6.8, sin embargo como puede observarse este vector $\begin{bmatrix} -1 & 0 & -1 \end{bmatrix}$ mantiene la aproximación de la derivada definida en la sección 6.2.2. También es posible notar como el vector de la izquierda $\begin{bmatrix} 1 & 1 & 1 \end{bmatrix}$ en ambos casos implica una operación de suavizado de los datos, mostrando así como el filtro aparte de localizar o realzar los píxel pertenecientes a los bordes, realiza una operación de suavizado, que permite hacer mas robusto el filtro, al ruido presente en la imagen.

El operador de Sobel tiene un filtro prácticamente idéntico al Prewitt, con la única diferencia de que en este filtro se le da un mayor peso al renglón o columna central del filtro. La matriz de coeficientes para este operador es definida como

$$H_x^S = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \text{ y } H_y^S = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \quad (6.10)$$

Los resultados de los filtros de Prewitt y Sobel producen estimaciones del gradiente local para todos los píxeles de la imagen en sus dos diferentes direcciones, manteniendo la siguiente relación

$$\nabla I(x, y) \approx \frac{1}{6} \begin{bmatrix} H_x^P \cdot I \\ H_y^P \cdot I \end{bmatrix} \quad \text{y} \quad \nabla I(x, y) \approx \frac{1}{8} \begin{bmatrix} H_x^S \cdot I \\ H_y^S \cdot I \end{bmatrix} \quad (6.11)$$

6.3.1.2 TAMAÑO Y DIRECCIÓN DEL GRADIENTE

Independientemente de si se trata de un operador Prewitt o Sobel se caracterizarán los resultados de los filtros para cada uno de los diferentes sentidos como:

$$D_x(x, y) = H_x * I \quad \text{y} \quad D_y(x, y) = H_y * I \quad (6.12)$$

La magnitud del borde $E(u, v)$ es en ambos casos definida como la magnitud del gradiente

$$E(x, y) = \sqrt{(D_x(x, y))^2 + (D_y(x, y))^2} \quad (6.13)$$

y la dirección del gradiente en cada píxel (el ángulo) es calculado a partir de

$$\phi(x, y) = \tan^{-1} \left(\frac{D_y(x, y)}{D_x(x, y)} \right) \quad (6.14)$$

La figura 6.5 muestra la representación del tamaño y dirección del gradiente sobre una imagen. El proceso completo para la detección de bordes es de nueva cuenta resumido en la Figura 6.6. Primero la imagen original es filtrada a través de las dos matrices de coeficientes H_x y H_y y consecuentemente sus resultados son reunidos en la magnitud del gradiente $E(x, y)$ y en la dirección del mismo $\phi(x, y)$.

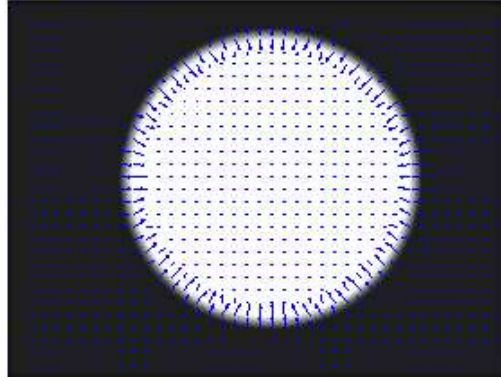


Figura 6.5 Representación de la magnitud del gradiente $E(x, y)$ y su dirección $\phi(x, y)$.

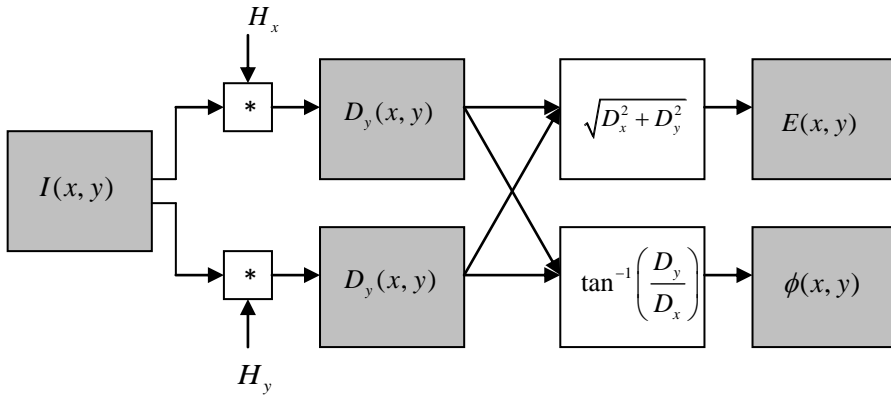


Figura 6.6 Operación típica de los filtros utilizados para la detección de bordes. Con ambas matrices de coeficientes H_x y H_y son producidas las imágenes de gradientes D_x y D_y y con ellas son calculados la magnitud del gradiente $E(x, y)$ y $\phi(x, y)$ su dirección.

El cálculo de la dirección del gradiente utilizando el operador Prewitt y la versión original del operador Sobel es relativamente imprecisa, por ello se sugiere en lugar de utilizar el filtro Sobel presentado en la ecuación 6.10 utilizar la versión que minimiza el error de ángulo, definida por

$$H_x^{s'} = \frac{1}{32} \begin{bmatrix} -3 & 0 & 3 \\ -10 & 0 & 10 \\ -3 & 0 & 3 \end{bmatrix} \quad y \quad H_y^{s'} = \frac{1}{32} \begin{bmatrix} -3 & -10 & -3 \\ 0 & 0 & 0 \\ 3 & 10 & 3 \end{bmatrix} \quad (6.15)$$

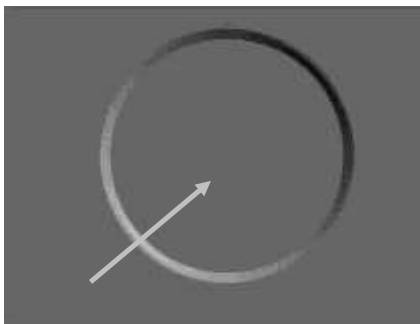
El operador Sobel es a razón de sus buenos resultados y facilidad de implementación muy utilizado e implementado en la mayoría de los paquetes de Software comerciales utilizados para el procesamiento de imágenes digitales.

6.3.2 El Operador Roberts

El operador Roberts es uno de los filtros mas viejos utilizados en la localización de bordes en una imagen y en este apartado será revisado por considerarlo históricamente interesante. El filtro es extremadamente pequeño utilizando una matriz de coeficientes de tan solo 2x2, para la determinación del gradiente en sus dos diferentes direcciones a lo largo de sus diagonales. El operador queda definido por:

$$H_1^R = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \quad \text{y} \quad H_2^R = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \quad (6.16)$$

Este filtro reacciona particularmente a los bordes de la imagen que tienen una dirección diagonal (Figura 6.7), sin embargo ello hace que el filtro sea poco selectivo en la dirección del gradiente, principalmente cuando este se calcula sobre regiones con una misma orientación. La magnitud del gradiente se calcula como fué definido en la ecuación 6.5, considerando ambas componentes H_1^R y H_2^R , sin embargo debido a la operación diagonal del filtro sobre los datos, puede considerarse que la magnitud del gradiente se forma como la resultante de dos vectores (H_1^R y H_2^R) que se encuentran a 45° . La Figura 6.8 esquematiza esta operación.



$$H_1^R = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$



Figura 6.7 Componentes diagonales del operador Roberts. De las imágenes es fácil reconocer el carácter direccional en el cálculo de este operador para encontrar la magnitud del gradiente en cada píxel de la imagen.

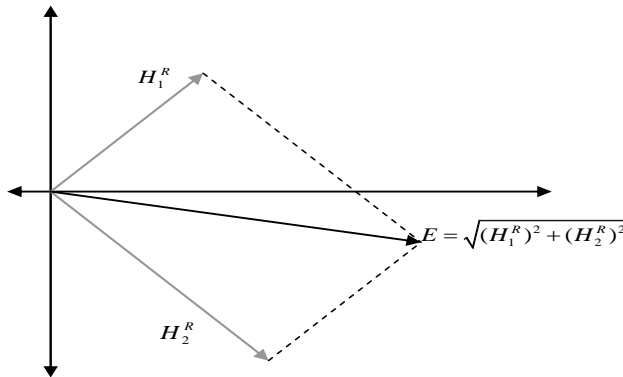


Figura 6.8 Magnitud del gradiente considerando el operador Roberts. La magnitud del gradiente $E(x, y)$ es calculado como la suma de los dos filtros ortogonales H_1^R y H_2^R que se aplican para cada dirección de la diagonal.

6.3.3 Operadores de Compás

Un problema en el diseño de filtros para la detección de bordes es que mientras más sensible se prefiere a la detección del borde de una estructura mas dependiente se vuelve a la dirección del mismo, por lo que para diseñar un buen operador es necesario hacer un compromiso entre magnitud de respuesta y sensibilidad a la dirección del gradiente.

Una solución para ello es no utilizar solamente dos filtros que descompongan el accionar del filtro en dos direcciones, ya sea horizontal y vertical en el caso de los operadores Prewitt y Sobel o en diagonal hacia arriba o abajo como lo es para el operador Roberts, sino utilizar filtros para un mayor número de direcciones. Un ejemplo clásico es el operador de Kirsch, el cual cuenta con ocho diferentes filtros,

cada uno separado del otro por 45°. Las matrices de coeficientes de este operador son las siguientes:

$$\begin{aligned}
 H_0^K &= \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, H_1^K = \begin{bmatrix} -2 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 2 \end{bmatrix}, H_2^K = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}, H_3^K = \begin{bmatrix} 0 & -1 & -2 \\ 1 & 0 & -1 \\ 2 & 1 & 0 \end{bmatrix} \\
 H_4^K &= \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}, H_5^K = \begin{bmatrix} 2 & 1 & 0 \\ 1 & 0 & -1 \\ 0 & -1 & -2 \end{bmatrix}, H_6^K = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}, H_7^K = \begin{bmatrix} 0 & 1 & 2 \\ -1 & 0 & 1 \\ -2 & -1 & 0 \end{bmatrix}
 \end{aligned} \tag{6.17}$$

De estos 8 filtros $H_0^K, H_1^K, \dots, H_7^K$ se deben calcular sin embargo solo cuatro, debido a que los últimos cuatro son iguales a los cuatro primeros excepto en el signo. Por ejemplo se tiene que $H_4^K = -H_0^K$. Debido a la propiedad de linealidad de la convolución es evidente que

$$I * H_4^K = I * -H_0^K = -(I * H_0^K) \tag{6.18}$$

Las imágenes producidas por la operación de los filtros de Kirsch D_0, D_1, \dots, D_7 son producidas de la siguiente manera:

$$\begin{aligned}
 D_0 &= I * H_0^K & D_1 &= I * H_1^K & D_2 &= I * H_2^K & D_3 &= I * H_3^K \\
 D_4 &= -D_0 & D_5 &= -D_1 & D_6 &= -D_2 & D_7 &= -D_3
 \end{aligned} \tag{6.19}$$

La magnitud del gradiente, la cual puede concebirse como una composición de todas las imágenes producto de los filtros de Kirsch, se calcula en el píxel (x, y) como el máximo de los valores de las imágenes obtenidas por la operación de cada uno de los 8 filtros. Por lo que el valor de la magnitud del gradiente para el píxel (x, y) queda definido como:

$$E^K(x, y) = \max(D_0(x, y), D_1(x, y), \dots, D_7(x, y))$$

$$= \max(|D_0(x, y)|, |D_1(x, y)|, \dots, |D_3(x, y)|)$$
(6.20)

La dirección del gradiente queda determinada por el filtro que aporta el máximo en el cálculo de la magnitud del gradiente. Por lo que la dirección del gradiente queda especificada por:

$$\phi^K(x, y) = \frac{\pi}{4} l \quad \text{donde} \quad l = \arg \max_{0 \leq i \leq 7} (D_i(x, y))$$
(6.21)

En términos prácticos las ventajas de los resultados ofrecidos por los operadores compás (tal como el operador Kirsch) apenas y son percibidos en comparación a operadores mas simples como lo es el Sobel. Quizás una ventaja interesante de este operador resulta el hecho de no necesitar el cálculo de la raíz cuadrada para la determinación de la magnitud del gradiente (al emplear en este caso el máximo), lo cual puede representar una ventaja cuando se disponen de bajos recursos de cómputo.

6.4 DETECCIÓN DE BORDES CON MATLAB®

Teniendo como fundamento teórico las secciones anteriores, presentaremos en esta sección la manera en la cual podemos utilizar las herramientas de MatLAB para el cálculo de los bordes de una imagen. Estas herramientas pueden ir desde la simple utilización de MatLAB como lenguaje de programación (archivo de extensión .m) hasta la utilización de los módulos de Simulink de procesamiento de imagen y visión artificial.

6.4.1 Utilización de MatLAB como lenguaje de programación para encontrar bordes

MatLAB es conocido como un poderoso lenguaje de programación de alto nivel que permite manipular de forma matricial expresiones matriciales, lo cual lo hace altamente atractivo para la programación de aplicaciones de procesamiento digital de imágenes. Para generar programas que permitan encontrar bordes de una imagen utilizando alguno de los operadores Sobel, Prewitt, Roberts o Kirsch, es necesario dividir el programa en 3 partes.

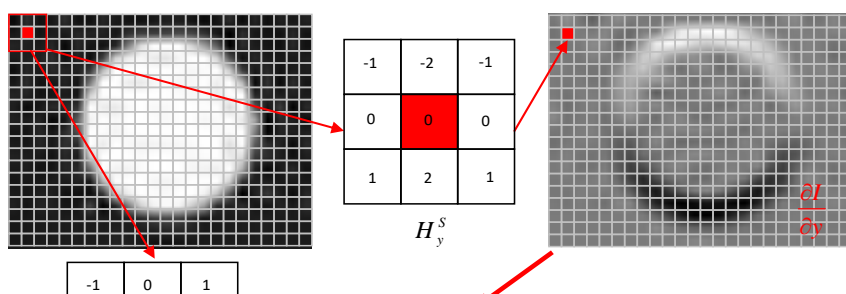


Figura 6.9 Proceso de codificación para la determinación de la magnitud del gradiente utilizando el operador Sobel.

En la Primera parte es necesario generar una imagen por filtro, normalmente estos son dos (por lo cual habrá dos imágenes), los cuales corresponden a las diferentes direcciones sobre las cuales se define el operador (véase la ecuación 6.4). En esta parte el filtro el cual es una matriz de coeficientes, se convoluciona en forma espacial con la imagen original, dando como resultado la

La magnitud del gradiente en la dirección definida para ese filtro en particular. La figura 6.9 muestra este proceso considerando como operador el Sobel.

En la segunda parte se obtiene la magnitud del gradiente a partir de las imágenes resultantes del proceso de convolución entre el filtro (ya sea horizontal o vertical) y la imagen (véase ecuación 6.5).

En la tercera parte se fija un umbral U que caracteriza el valor a partir del cual se considera que el píxel en cuestión forma parte de un borde, que de manera natural define las propiedades estructurales de los objetos.

El programa 6.1 muestra la codificación en MatLAB para la determinación de los bordes en una imagen considerando como operador el de Sobel.

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Determinación de los Bordes de una imagen
% utilizando el operador Sobel
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Erik Cuevas, Daniel Zaldivar, Marco Pérez
% Versión: 25/10/07
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%Se obtienen los valores de las dimensiones de la
%Imagen
[m n]=size(Im);
%Se convierte la imagen a double para evitar problemas
%en la conversión del Tipo de dato
Im=double(Im);
%Se crean las Matrices con ceros
Gx=zeros(size(Im));
Gy=zeros(size(Im));

%PRIMERA PARTE
%Se aplican los Filtros Sobel (véase ecuación 6.10)
% a la Imagen Gx, en dirección x
%Gy en la dirección y
for r=2:m-1
    for c=2:n-1
        Gx(r,c)=-1*Im(r-1,c-1)-2*Im(r-1,c)-Im(r-
1,c+1)...
            +Im(r+1,c-1)+2*Im(r+1,c)+Im(r+1,c+1);
        Gy(r,c)=-1*Im(r-1,c-1)+Im(r-1,c+1)-2*Im(r,c-
1)...
            +2*Im(r,c+1)-Im(r+1,c-1)+Im(r+1,c+1);
    end
end

%SEGUNDA PARTE
%Se calcula el Valor total del Gradiente
% (véase ecuación 6.5 o 6.13)
Gt=sqrt(Gx.^2+Gy.^2);
%Se obtiene el Valor máximo del Gradiente
VmaxGt=max(max(Gt));
%Se Normaliza el gradiente a 255
GtN=(Gt/VmaxGt)*255;

%Se convierte la imagen para el despliegue
GtN=uint8(GtN);

%TERCERA PARTE
%Se binariza la imagen considerando un umbral del 100
B=GtN>100;

%Se obtienen los valores mínimos para los Gradientes x
%e y.
VminGx=min(min(Gx));

```

```

VminGy=min(min(Gy));

%Utilizando los mínimos se desplaza para evitar
%negativos
GradOffx=Gx-VminGx;
GradOffy=Gy-VminGy;

%Se obtienen los valores máximos para Gx y Gy.
VmaxGx=max(max(GradOffx));
VmaxGy=max(max(GradOffy));

%Se normalizan los gradientes a 255
GxN=(GradOffx/VmaxGx)*255;
GyN=(GradOffy/VmaxGy)*255;

%Se convierte la imagen para el despliegue
GxN=uint8(GxN);
GyN=uint8(GyN);

```

Programa 6.1 Determinación de los bordes de una Imagen con MatLAB®.

6.4.2 Funciones de MatLAB para la detección de bordes.

El toolbox de procesamiento de imágenes provee la función `edge`, la cual implementa los diferentes operadores tratados en las secciones anteriores (Sobel, Prewitt y Roberts). Para algunos de estos operadores es posible especificar la dirección de filtro a calcular, lo cual significaría indicar la sensibilidad bajo la cual se realizaría el cálculo del gradiente, horizontal, vertical o ambos.

La sintaxis de la función en forma generalizada sería

```
[g t]=edge(f, 'método', parámetros)
```

Donde `f` es la imagen a la cual se le pretenden extraer los bordes, `método` es uno de los operadores listados en la Tabla 6.1, y `parámetros` son las especificaciones que deben configurarse dependiendo del tipo de método utilizado y los cuales serán explicados en forma individual para cada operador. La salida `g` es una imagen binaria donde los píxeles pertenecientes a los bordes detectados de `f` tienen el valor de 1, mientras los que no tendrán el valor de cero. El parámetro `t` es opcional y entrega el umbral utilizado por el algoritmo para determinar cual valor del gradiente puede ser considerado como borde.

Operador	'método'	Filtros
Prewitt	'prewitt'	$H_x^P = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \quad H_y^P = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$
Sobel	'sobel'	$H_x^S = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad H_y^S = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$
Roberts	'roberts'	$H_1^R = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \quad H_2^R = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$

Tabla 6.1 Operadores de detección de bordes utilizados por la función `edge` del toolbox de procesamiento de imagen.

6.4.2.1 OPERADOR SOBEL

El operador sobel utiliza los filtros descritos en la Tabla 6.1 para aproximar las derivadas parciales $\partial I / \partial u$ y $\partial I / \partial v$. De la combinación de esas derivadas parciales según la ecuación 6.5 obtenemos para cada píxel (u,v) un valor del gradiente. Entonces, se dice que un píxel (u,v) corresponde a un borde de la imagen si el valor de su gradiente es mayor a un umbral **U** preestablecido como criterio.

La llamada general a la función de detección de bordes que implementa el toolbox de procesamiento de imágenes bajo el método Sobel es:

```
[g t]=edge(f, 'sobel', U, dir)
```

donde **f** es la imagen a la cual se le pretenden extraer los bordes, **T** es un umbral que se especifica como criterio de magnitud del gradiente para clasificar a los bordes y **dir** selecciona el filtro a utilizar el cual puede ser `'horizontal'` si se utiliza H_x^S o `'vertical'` si se utiliza H_y^S , la opción normal (default) es utilizar como criterio de gradiente el cálculo que se genera a partir del cálculo de ambos filtros lo cual en todo caso quedará especificado por `'both'`. Como respuesta a la llamada de la función obtenemos la imagen **g** que como se dijo anteriormente contiene los bordes detectados, al ser la imagen binaria esto se ve reflejado por aquellos píxeles que tienen el valor de 1 (bordes) mientras que los que son ceros representan aquellos que no son bordes. Si el umbral **U** es especificado **t**=**U**. Si **U** no se especifica el algoritmo determina uno automáticamente y lo utiliza para la detección de bordes y a su vez devuelve su valor en **t**.

6.4.2.2 OPERADOR PREWITT

El operador Prewitt utiliza para la detección de bordes las matrices de coeficientes especificadas en la Tabla 6.1. La sintaxis general de la función `edge` utilizando este método es:

```
[g t]=edge(f, 'prewitt', U, dir)
```

Los parámetros de esta función son idénticos a los indicados en el caso Sobel. El operador Prewitt es un poco mas simple (computacionalmente hablando) de ser implementado en relación al caso Sobel, sin embargo los resultados obtenidos son un poco mas ruidosos, ya que en el caso Sobel de acuerdo a los coeficientes implementados en sus filtros realiza un suavizado sobre los datos, lo cual puede ser observado en los coeficientes con valor 2 en el renglón o columna del píxel sobre el cual se desea calcular el gradiente.

6.4.2.3 OPERADOR ROBERTS

El operador Roberts utiliza los filtros definidos en la Tabla 6.1 para aproximar el cálculo del gradiente en el píxel (u,v) . La llamada general a la función `edge` con este método es:

```
[g t]=edge(f, 'roberts', U, dir)
```

Los parámetros de esta función son idénticos a los descritos en el caso del operador Sobel. El operador Roberts representa uno de los métodos más antiguos para el cálculo del gradiente en imágenes. Aunque este método es el mas sencillo de implementar tiene una limitada funcionalidad, ya que al ser no simétrico no puede detectar bordes que se encuentren direccionalmente en múltiplos de 45° . Como comparación la Figura 6.10 muestra el resultado de aplicar las anteriores funciones para cada uno de los diferentes operadores para detectar los bordes en una imagen, utilizando el mismo umbral para todos los casos.

6.4.3 Utilización de los bloques de procesamiento de imagen y video de Simulink[®]

Los bloques de procesamiento de imagen y video de Simulink implementan la función de detección de bordes en una imagen o bien en una señal de video proveniente de un archivo o una cámara conectada a la computadora.



(a)

(b)

(c)

(d)

Figura 6.10 Comparación de los diferentes operadores utilizados para la detección de bordes, utilizando la función edge de MatLAB. (a) Imagen original, (b) bordes detectados por el operador Sobel, (c) Prewitt y (d) Roberts. En todos los casos se utilizó un umbral=0.02.

Los bloques implementados de procesamiento de imágenes y video para trabajar en el ambiente de Simulink permiten realizar operaciones directamente con imágenes o video, interconectando bloques como es la filosofía de Simulink para generar el procesamiento indicado sobre las imágenes. En el caso de detección de bordes la librería de Simulink implementa el bloque ‘Edge Detection’, el cual permite encontrar los bordes de una imagen o de Frames de Video, utilizando para ello los operadores Sobel, Prewitt y Roberts anteriormente discutidos en este capítulo. La Figura 6.11 muestra el bloque ‘Edge Detection’ de Simulink.

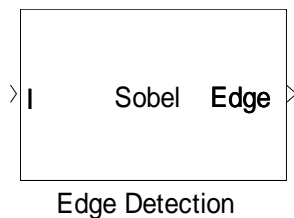


Figura 6.11 Bloque para la detección de bordes en imágenes o frames de video, como parte de la librería de procesamiento de imagen y video de Simulink. El operador por defecto de este bloque es el Sobel.

La mascarilla principal de este bloque la cual aparece en la Figura 6.12 permite configurar cuatro aspectos importantes.

1. **El método**, el cual se refiere al operador utilizado para aproximar el valor del gradiente de la imagen. Dicho operador puede ser el Sobel, Prewitt o Roberts, cuyas generalidades ya fueron discutidas anteriormente en este capítulo.
2. **La imagen de salida**, donde se especifican 3 diferentes opciones. a) Binary image, especificando esta opción estamos estableciendo que, como salida de este bloque, tendremos la imagen binaria donde el píxel en uno corresponde al borde, mientras que los píxeles en cero corresponden a puntos no-bordes. b) Gradient components, con esta opción aparecerán realmente dos salidas (G_h y G_v) en el bloque las cuales corresponden a las salidas provocadas por cada uno de los filtros de los que se componen cada uno de los operadores que son las imágenes de los gradientes en cada una de las dos direcciones. c) Binary image and Gradient components, con esta opción aparecerán tres diferentes salidas en el bloque las cuales corresponderán a la imagen que contiene los bordes (la imagen binaria) y las imágenes correspondiente a cada una de las direcciones del gradiente (G_h y G_v).
3. **El valor del Umbral**, el cual especifica el valor a partir del cual el gradiente de un píxel es considerado como un borde en la imagen, para poder definir este valor habrá que señalar la opción “User-defined threshold”.
4. **Ancho del borde**, en esta opción el usuario puede definir que bordes que son muy anchos reducirlos a un solo píxel, esta opción es particularmente atractiva para aumentar la legibilidad de la imagen obtenida.

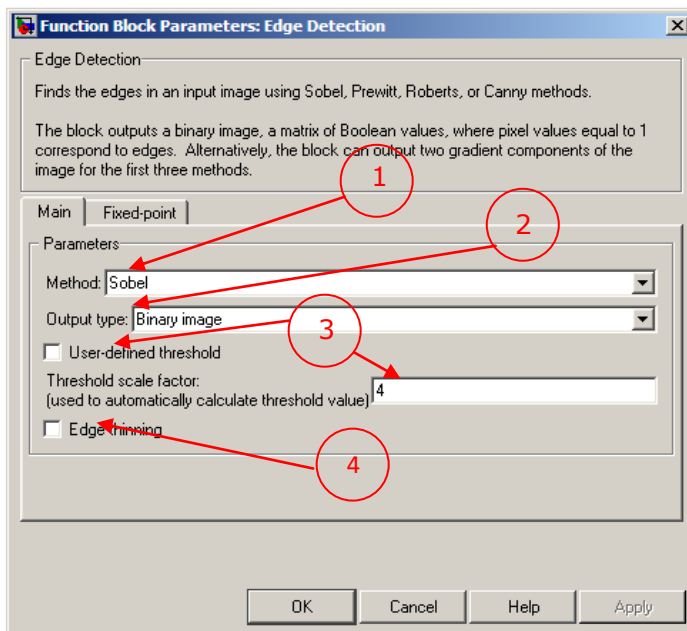


Figura 6.12 Mascarilla principal del bloque “Edge Detection”, desglosada por los campos de configuración.

6.4.3.1 UTILIZACIÓN DE LOS BLOQUES DE PROCESAMIENTO DE IMAGEN Y VIDEO DE SIMULINK®

PARA LA DETERMINACIÓN DE BORDES DE UNA IMAGEN.

Para encontrar los bordes de una imagen utilizando la librería de bloques para el procesamiento de imagen y video de Simulink, lo único que hay que hacer es colocar en el modelo los bloques necesarios para la lectura de la imagen, para la conversión de de RGB (en caso de que la imagen tenga ese formato de color) a escala de grises, para la detección de bordes y para el despliegue de la imagen resultado. La Figura 6.13 muestra un ejemplo de modelo Simulink para la determinación de bordes en una imagen, utilizando el operador Prewitt.

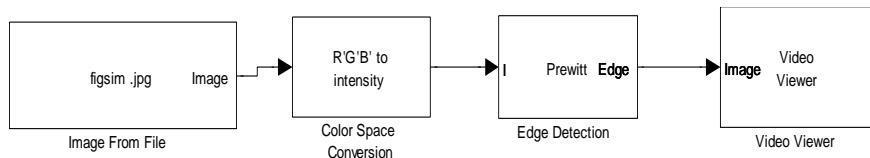


Figura 6.13 Proceso para encontrar los bordes de una imagen, utilizando las librerías de bloques para el procesamiento de imagen y video de Simulink. En el ejemplo se utiliza como operador que aproxima el valor del gradiente el Prewitt.

Como muestra la Figura 6.13, el sistema consta de 4 diferentes bloques, el primero “Image From File”, permite tomar la imagen de un archivo, como dicha imagen se encuentra en el formato RGB habrá que convertirla a una imagen de intensidad (a escala de grises). La operación de conversión de una imagen RGB a escala de grises u otros modelos de color es realizada por el bloque “Color Space Conversión”. Una vez disponible la imagen en escala de grises se utiliza el bloque “Edge Detection”, el cual se configura tal y como fué tratado en el apartado anterior. Específicamente en la Figura 6.12 fué utilizado como operador el Prewitt. Por último el resultado de la imagen binaria obtenida del bloque “Edge Detection” se despliega en la consola con el bloque “Video Viewer”.

Un aspecto importante además de la configuración individual de cada uno de los bloques, es la configuración del ambiente Simulink para que el sistema sea capaz de realizar el algoritmo de procesamiento de la imagen. Como es conocido Simulink es un ambiente que originalmente fue concebido para la simulación de sistemas dinámicos, por lo que para que pueda funcionar adecuadamente para el procesamiento de imágenes es necesario realizar lo siguiente en la opción **Simulation>Configuration Parameters**:

1. El tiempo inicial (Start time) y final (Stop time) debe de ser configurado a cero. Esto es como se trata de realizar el procesamiento completo de una imagen una sola vez, el tiempo no tiene significación.
2. En las opciones del método numérico, se configura el tipo (Type) a “Fixed-step” mientras que en la configuración del método numérico (Solver) se elige “Discrete no continuous status”. La Figura 6.14 muestra la forma final de la configuración de los parámetros de simulación.

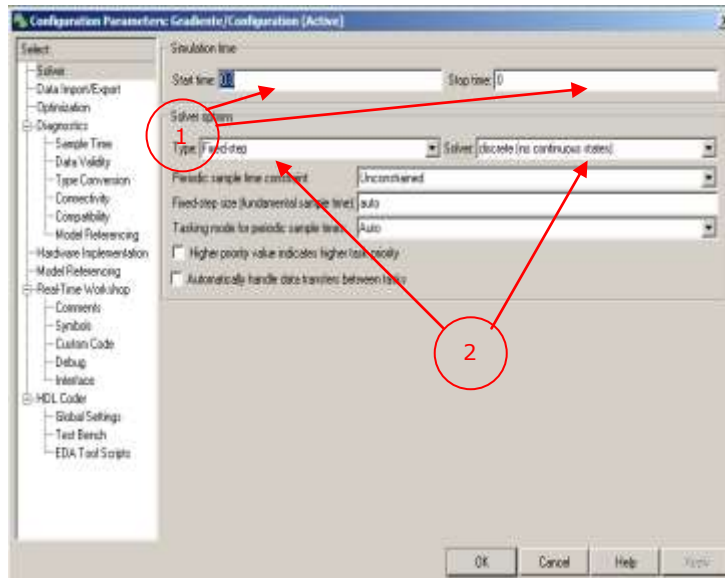



Figura 6.14 muestra la forma final de la configuración de los parámetros de simulación.

Una vez establecido el sistema de procesamiento tal como lo muestra la Figura 6.13 y configurado el ambiente se procede a establecer el procesamiento de la imagen, para lo cual sólo es necesario pulsar el botón  a partir del cual iniciará la operación del algoritmo programado, la Figura 6.15 muestra el resultado de la utilización del bloque “Edge Detection” sobre una imagen utilizando el operador Prewitt y un umbral de 0.2.

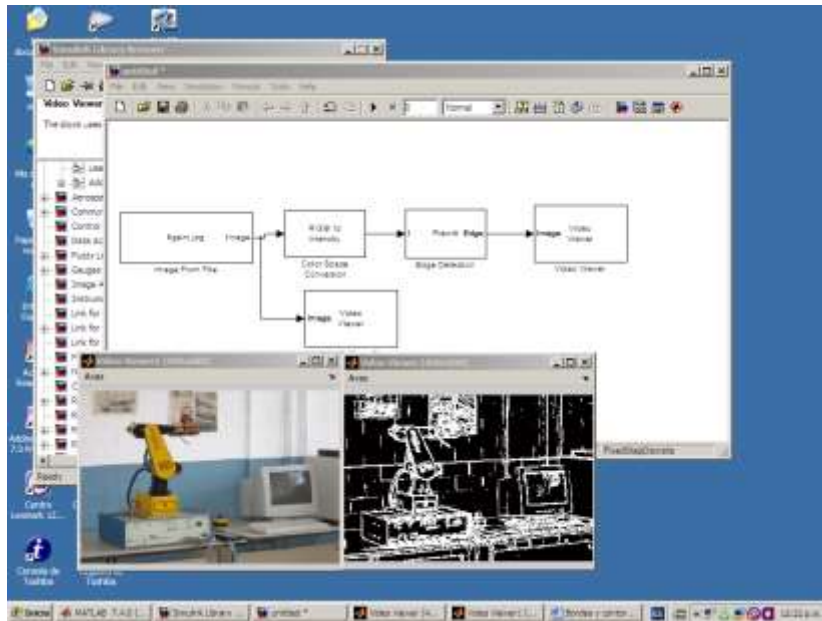


Figura 6.15 Resultado de utilizar el bloque de detección de bordes sobre una imagen, utilizando el operador Prewitt y un umbral de 0.2.

Otra opción, con la que puede encontrarse los bordes de una imagen mediante la determinación del gradiente en un píxel, es la de programar directamente en Simulink los filtros correspondientes al operador en cuestión tal y como son descritos en la Tabla 6-1. Obviamente ello conllevaría a un trabajo adicional puesto que la idea es programar lo que ya viene implementado internamente en el bloque “Edge Detection”. Aunque esto podría parecer ocioso el hecho de programar directamente las estructuras aparte de significar un ejercicio interesante de programación en Simulink permitiría ganar flexibilidad en la utilización de operadores pudiendo de esta manera programar otros filtros utilizados para la detección de bordes que no vengan implementados en el bloque “Edge Detection” tal y como lo es el filtro Kirsch (véase subsección 6.3.3). Esta sección es importante por significar el desarrollo de una estructura de programa de Simulink estándar utilizada en este libro no solo para la detección de bordes sino para otras aplicaciones donde se utiliza el filtrado espacial.

Para poder llevar a cabo el programa en Simulink que determine los bordes de una imagen a través de la programación directa del operador es necesario describir la utilización de algunos bloques funcionales, parte de la librería de bloques para el procesamiento de imagen y video.

6.4.3.2 IMAGE FROM WORKSPACE

Este bloque permite añadir al algoritmo de procesamiento una matriz o imagen constante definida previamente desde la consola de comandos de MatLAB. La Figura 6-16 muestra el bloque como es representado en Simulink. La configuración de este bloque permite definir el nombre de la variable tal y como fué declarada en línea de comandos.

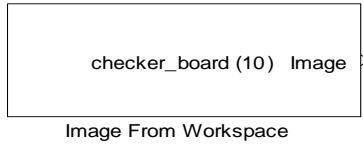


Figura 6.16 Bloque de Simulink para añadir una matriz o imagen al algoritmo de procesamiento de imagen. El bloque por defecto inicializa con una imagen predefinida, tal y como se muestra.

6.4.3.3 2-D CONVOLUCION

El bloque de “2-D Convolucion” realiza la convolución de dos matrices. Asumiendo que la matriz A tiene las dimensiones (M_a, N_a) y la matriz B tiene las dimensiones (M_b, N_b) , cuando el bloque calcula la convolución, el cálculo realizado se efectúa en base a la siguiente ecuación

$$C(i, j) = \sum_{m=0}^{(M_a-1)} \sum_{n=0}^{(N_a-1)} A(m, n) * B(i-m, j-n) \quad (6.22)$$

donde $0 \leq i < M_a + M_b - 1$ y $0 \leq j < N_a + N_b - 1$. La Figura 6.17 muestra el esquema del bloque en Simulink. Así mismo en la Tabla 6.2 se describe la funcionalidad entrada/salida de sus conexiones. Tal y como fué visto en la sección 5.5.1 la convolución puede ser considerada como una operación de filtro normal (correlación) con la excepción de que la matriz $H(i, j)$ de coeficientes debe de ser invertida en sus ejes coordenados $H(-i, -j)$, lo cual puede ser llevado a cabo por una rotación de $H(i, j)$ en 180° .

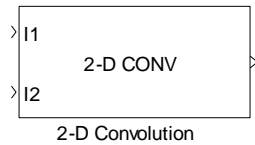


Figura 6.17 Bloque de la librería para el procesamiento de imagen y video de Simulink para realizar la convolución espacial entre dos matrices, normalmente una imagen y una ventana donde se encuentra contenido la funcionalidad del filtro.

Conexión	Descripción
I1	Matriz o una imagen a escala de grises, o bien un plano de la imagen RGB.
I2	Matriz, o ventana de convolución, normalmente no simétrica.
Salida	Una imagen o matriz resultado de la convolución de I1 y I2.

Tabla 6.2 Funcionalidad de las conexiones entrada/salida del bloque 2-D Convolution.

El bloque tiene la funcionalidad de detectar el tipo de dato utilizado a sus entradas generando de esta manera una salida coherente con el tipo de datos de entrada.

Las dimensiones de la matriz o imagen de salida se encuentran determinadas por el parámetro de configuración “Output Size”. Asumiendo que las dimensiones de la conexión I1 son (Ma, Na) y para la la conexión I2 son (Mb, Nb). Si se elige el parámetro de configuración “Output Size” a `Full`, la salida es la imagen producida por la convolución de I1 e I2, con dimensiones (Ma+Mb-1, Na+Nb-1). Si por el contrario se elige el parámetro de configuración “Output Size” a `Same as input port I1`, la salida es la parte central de la convolución verificada entre I1 e I2, omitiendo las partes laterales o bordes de la imagen o matriz resultado de la convolución completa, por consiguiente las dimensiones de la matriz o imagen resultado son los mismos de los de I1. Por último el parámetro de configuración “Output Size” puede también tener el valor de `valid`, con lo que la imagen o matriz de salida será el resultado de la convolución entre I1 e I2 sin considerar añadir ceros en los bordes de la matriz I1, de tal forma que pueda realizarse la multiplicación de los coeficientes de I2.

En la convolución, el valor de un elemento de la salida es computado como la suma de las multiplicaciones de los elementos vecinos. Por ejemplo, supóngase que la primera matriz del bloque I1 es definida como:

$$I1 = \begin{bmatrix} 2 & 3 & 1 & 3 \\ 6 & 7 & 2 & 1 \\ 5 & 6 & 1 & 0 \\ 5 & 4 & 8 & 1 \end{bmatrix} \quad (6.23)$$

La segunda matriz de entrada al bloque I2 la cual normalmente representa la ventana que define la matriz de coeficientes y las vecindades tomadas en cuenta para el cálculo de los elementos de salida de la convolución, es definida como:

$$I2 = \begin{bmatrix} 2 & 1 & 2 \\ 3 & 2 & 4 \\ 4 & 1 & 2 \end{bmatrix} \quad (6.24)$$

Considerando estas dos matrices, el proceso para calcular el elemento (2,2) de la salida resultado de la convolución entre I1 e I2 es el siguiente:

1. Tal como se observa en la ecuación 6.21 se rota la matriz I2 180° con respecto a su punto central.

$$\begin{bmatrix} 2 & 1 & 2 \\ 3 & 2 & 4 \\ 4 & 1 & 2 \end{bmatrix} \xrightarrow{180^\circ} \begin{bmatrix} 2 & 1 & 4 \\ 4 & 2 & 3 \\ 2 & 1 & 2 \end{bmatrix}$$

2. Se desplaza el elemento central de la matriz I2 rotada de tal forma que su elemento central coincida con el elemento (2,2) de la matriz I1.
3. Se multiplica cada elemento de la matriz I2 rotada por los elementos con los que coinciden de la matriz I2, que se encuentra debajo de ella.
4. Se suman los productos individuales del paso 3, para calcular el valor total de (2,2).

La Figura 6.18 muestra un esquema de la forma en como se desarrolla el procesamiento de la convolución para el elemento (2,2), considerando como I1 e I2 las matrices anteriormente descritas.

6.4.3.4 PROGRAMACIÓN DIRECTA DEL ALGORITMO PARA LA LOCALIZACIÓN DE BORDES EN SIMULINK®.

Para realizar la programación directa del gradiente prescindiendo de utilizar el bloque “Edge Detection”, es necesario solamente considerar la aplicación directa de un operador de la Tabla 6.1, sobre la imagen generando por la operación de cada filtro (dependiendo de la dirección sobre la cual se encuentre el gradiente) una matriz que representa al gradiente en esa dirección (G_x, G_y).

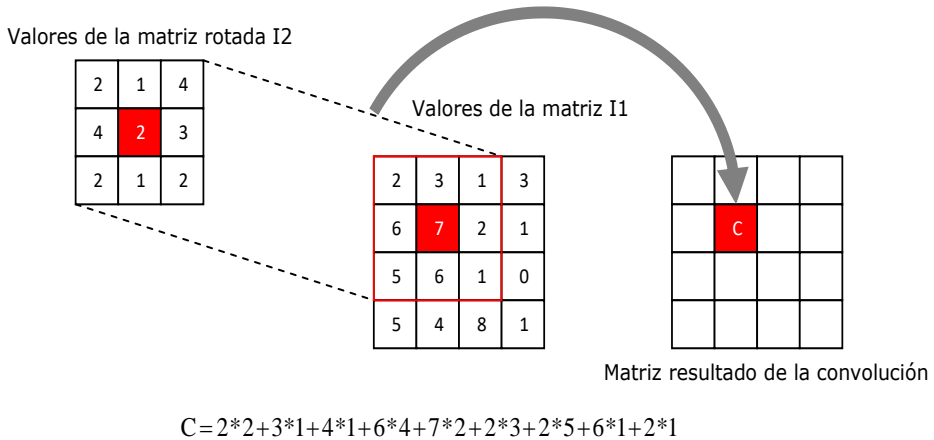


Figura 6.18 Representación de la convolución realizada entre las matrices I1 e I2, considerando que se calcula el elemento (2,2), según el proceso anteriormente tratado. Del proceso puede notarse como la ventana de convolución, la cual normalmente es utilizada para realizar un procesamiento espacial en la imagen, debe ser rotada 180° antes de realizar la multiplicación.

Teniendo ambas matrices, obtenemos el gradiente total el cual puede ser calculado a partir de la ecuación 6.5, sin embargo otra opción que inclusive tiene un costo computacional menor sería:

$$G_{total} = |G_x| + |G_y| \quad (6.25)$$

La Figura 6.19 muestra el programa en Simulink realizado para la detección de bordes en una imagen.

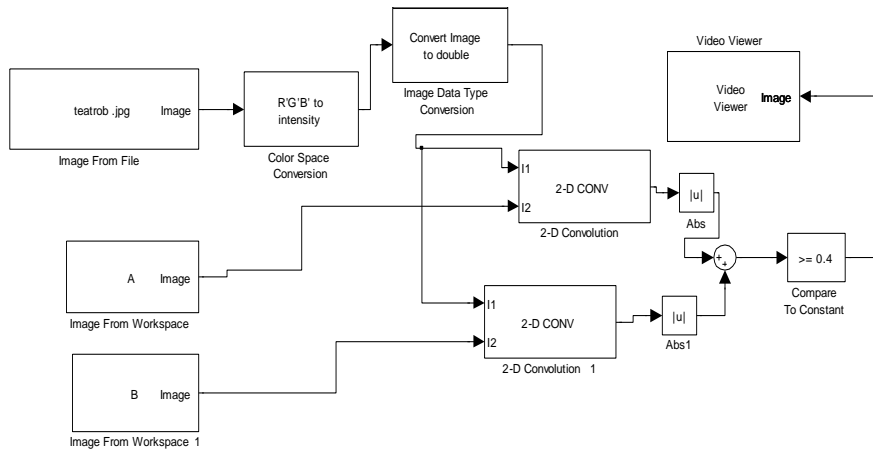


Figura 6.19 Programa en Simulink para detectar los bordes en una imagen sin utilizar el bloque “Edge Detection”. Las matrices A y B, representan los filtros que se aplican dependiendo del operador utilizado para encontrar el gradiente.

Tal y como se muestra en la figura 6.19 las matrices A y B representan los filtros del operador utilizado como forma de aproximar el gradiente. En el caso del ejemplo presentado aquí se escogió el operador Sobel. Considerando esto las matrices de coeficientes que definen a sus filtros están indicadas en la Tabla 6.1. Es de notar que las matrices A y B, deben de ser definidas desde la consola de MatLAB, para que de ahí puedan ser tomadas por Simulink. Sin embargo como fué ya explicado, para el calculo de los elementos de la convolución entre dos matrices I1 e I2, utilizando el bloque “2-D Convolution”, la matriz I2, la cual corresponde normalmente a la rejilla que determina el procesamiento espacial de la imagen I1, es rotada 180° (como se ve en la ecuación 6.22) antes de realizar su procesamiento. Por ello debemos de considerar este efecto al momento de definirla por línea de comandos en MatLAB, para que cuando sea rotada esta operación no afecte la definición de la matriz. Para el ejemplo aquí mostrado al utilizar el operador Sobel se tiene:

```
>>A=[-1 0 1; -2 0 2; -1 0 1]; %Este Filtro corresponde a  $H_x^S$ 
>>B=[-1 -2 -1; 0 0 0; 1 2 1]; %Este Filtro corresponde  $H_y^S$ 
>>A=-A; %Se cambia A para que cuando sea rotada corresponda a  $H_x^S$ 
>>B=-B; %Se cambia B para que cuando sea rotada corresponda a  $H_y^S$ 
```

Por último se obtiene el valor del gradiente total según la ecuación 6.24 y se le aplica un umbral de 0.4, con lo cual se eligen como bordes los píxeles de la imagen cuyo gradiente sea igual o mayor a ese valor. La Figura 6.20 muestra el resultado de haber ejecutado el programa de la figura 6.19.

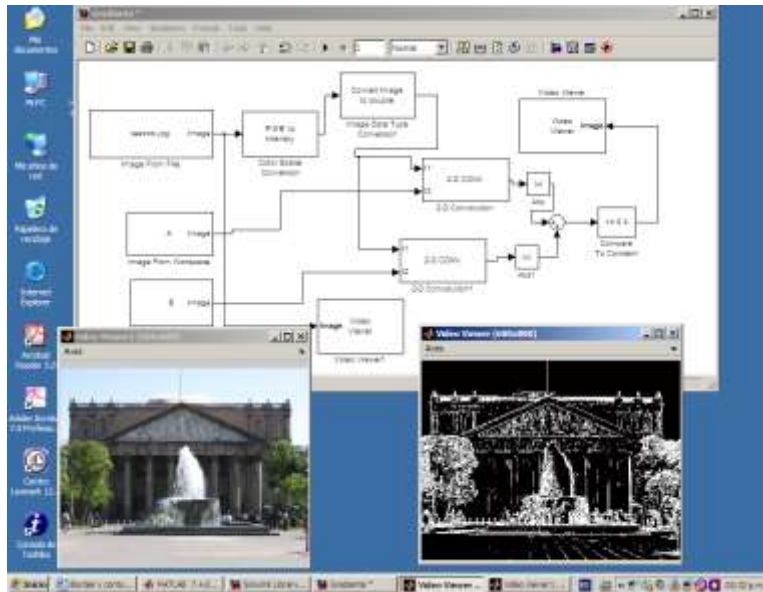


Figura 6.20 Imagen original y resultado, como consecuencia de haber ejecutado el programa definido en la Figura 6.18.

Es importante también indicar que ambos bloques que desempeñan la convolución con los filtros Sobel son configurados a **Same as input port I1** en su parámetro “Output Size”. Con ello hacemos que el resultado de la convolución coincida con las dimensiones de la imagen original definida en I1.

6.4.3.5 UTILIZACIÓN DE LOS BLOQUES DE PROCESAMIENTO DE IMAGEN Y VIDEO DE SIMULINK® PARA LA DETERMINACIÓN DE BORDES EN FRAMES DE VIDEO.

Para utilizar la librería de bloques de procesamiento de imágenes y video de Simulink para procesar video, se aplican todas las observaciones realizadas para el caso del procesamiento de imagen, es decir la funcionalidad de los bloques no cambia. Sin embargo lo que si es necesario considerar es que el procesamiento de video es un proceso iterativo, en el cual un procesamiento o algoritmo se ejecuta sobre el frame (una imagen de varias) de video entrante, continuando este proceso indefinidamente. A consecuencia de ello es necesario configurar el ambiente de simulación, de tal forma que el algoritmo se ejecute continuamente conforme existan frames de video disponibles.

Dicha configuración es llevada a cabo con solo colocar el tiempo final o “Stop time” de la opción **Simulation>Configuration Parameters** a inf. La figura 6.21 muestra la pantalla de los parámetros de configuración de la simulación y el valor ya configurado.

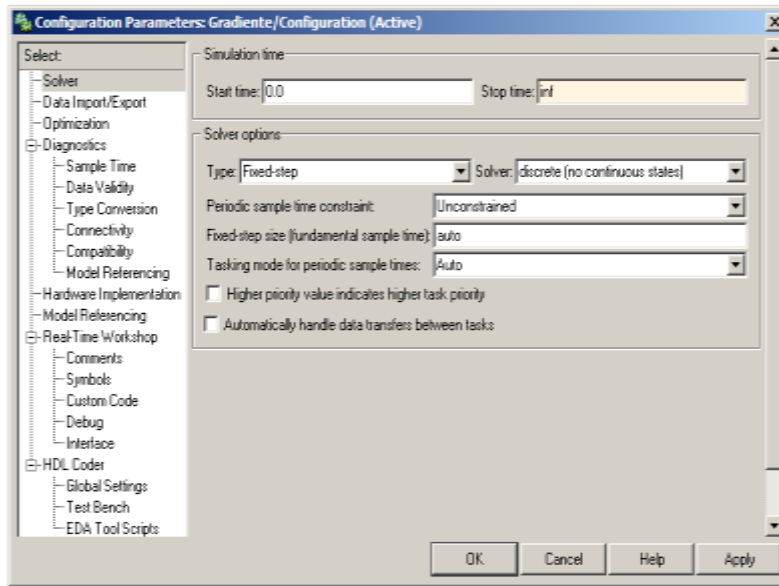


Figura 6.21 Pantalla de configuración de parámetros de la simulación con los parámetros ajustados para el procesamiento de video utilizando la librería de bloques para el procesamiento de imágenes y video de Simulink.

Para poder procesar frames de video lo primero que hay que tener es como entrada una fuente de video que permita obtener imágenes y a partir de ellas procesarlas. Una opción interesante sería poder captar imágenes a partir de una fuente de video de bajo costo como lo sería una Webcam.

Para poder captar imágenes a partir de una Webcam MatLAB provee un bloque parte de las librerías contenidas en el **Toolbox de Adquisición de Imágenes**. El bloque es accesible desde Simulink y permite capturar frames de una Webcam, con el solo hecho de estar instalado su driver.

El bloque se encuentra representado en la Figura 6.22, en ella pude apreciarse como se trata de un bloque cuya funcionalidad es completamente destinada a fungir como una fuente de imágenes provenientes en este caso de la Webcam.

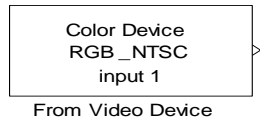


Figura 6.22 Bloque “From Video Device” parte de la librería del Toolbox de Adquisición de Imágenes. Permite capturar imágenes de un dispositivo de video compatible con Windows®, tal como una Webcam.

Para poder utilizar este bloque es necesario configurarlo de tal forma que pueda ser capaz de conectarse con el dispositivo de video (tal como la Webcam). Para ello en su mascarilla de configuración es necesario seleccionar el concepto “Device” la opción winvideo, la cual hará posible que el bloque pueda ser capaz de comunicarse con el driver del dispositivo de video, en este caso con la Webcam. La Figura 6.23 muestra la ventana de configuración del bloque “From Video Device” y el valor seleccionado para poder utilizar a la Webcam como fuente de video.

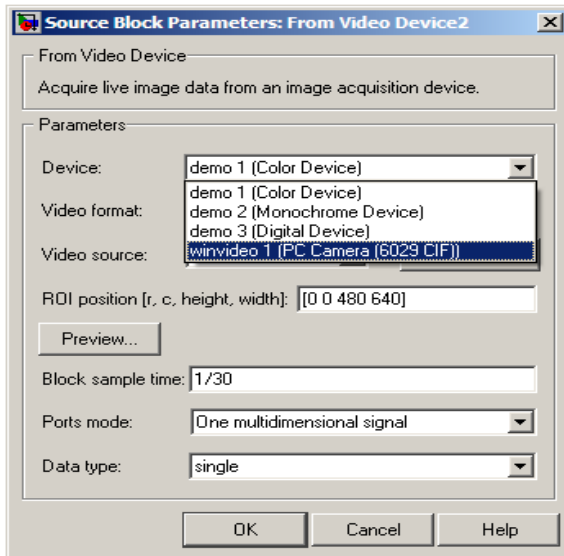


Figura 6.23 Muestra la ventana de configuración del bloque “From Video Device” y el valor seleccionado para poder utilizar a la Webcam como fuente de video. Como puede verse en la figura entre paréntesis se señala el nombre de la Webcam que en ese momento se encuentra conectada y disponible para ser utilizada.

Para ejemplificar la utilización de los bloques de la librería de procesamiento de imagen y video de Simulink, tomaremos de nueva cuenta el programa presentado en la Figura 6.19, con sus respectivas modificaciones, esto es habiendo modificando el “Stop time” a **inf** y colocado como fuente de video el bloque “From Video Device”

de la librería del Toolbox de Adquisición de Imágenes configurado para adquirir imágenes de la Webcam.

La Figura 6.24 muestra el diagrama del algoritmo para la detección de bordes en Simulink procesando frames de video. Cabe mencionar que el valor de las matrices A y B son los mismos (operador Sobel) que los utilizados para el caso del procesamiento de una imagen individual. La única diferencia es que en este caso el valor del umbral escogido para la selección del borde es de 0.2.

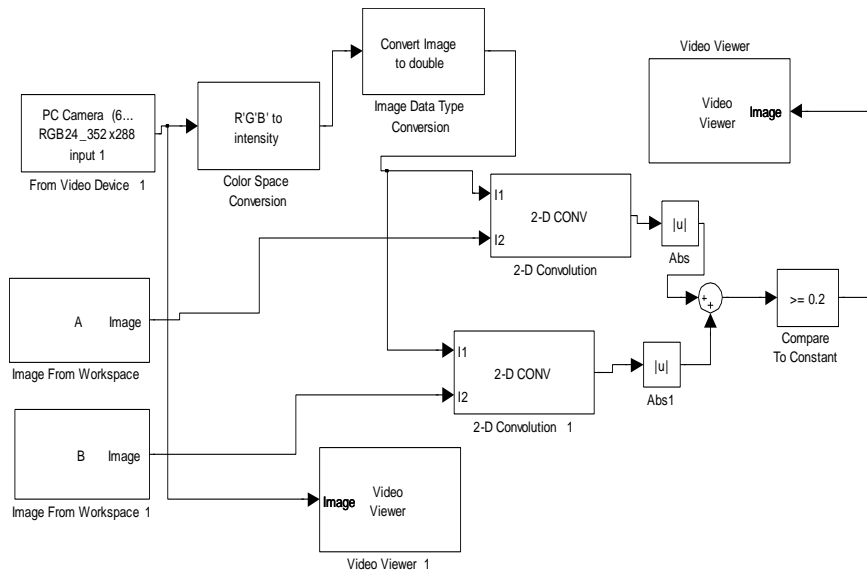


Figura 6.24 Muestra el diagrama del algoritmo para la detección de bordes en Simulink procesando frames de video. El diagrama es igual al utilizado para encontrar los bordes de la imagen, con la diferencia de que en este se incorpora el bloque “From Video Device” para adquirir imágenes de la Webcam y el umbral utilizado para la detección del borde es de 0.2.

En la Figura 6.25 se muestra el resultado de haber ejecutado el programa en Simulink de la Figura 6.24.

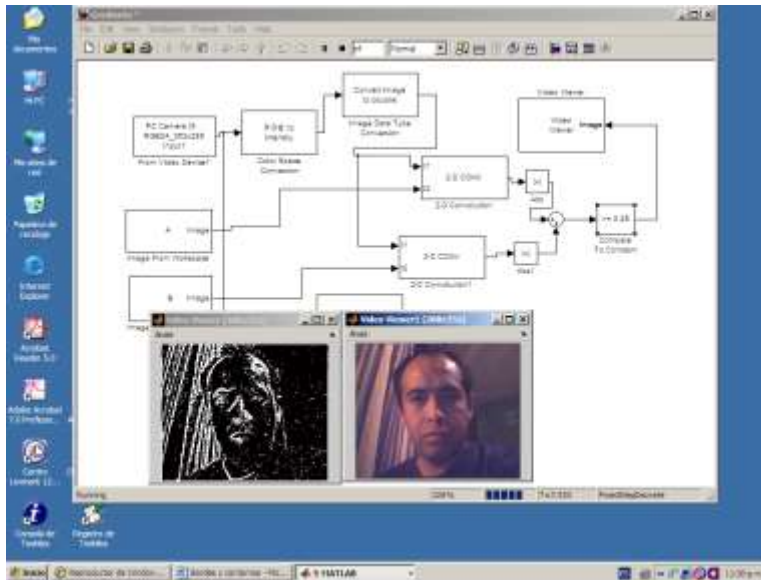


Figura 6.25 Resultado de haber ejecutado el programa en Simulink mostrado en la Figura 6.23.

6.5 OPERADORES BASADOS EN LA SEGUNDA DERIVADA

Tal y como fueron descritos en la sección 6.2 el grupo de operadores basados en la primera derivada, como forma de aproximar el gradiente de la imagen, existen otro tipos de operadores los cuales se fundamentan en la segunda derivada de la función de la imagen. En una imagen, los cambios de intensidad pueden ser calculados mediante el valor de la derivada en una determinada dirección (gradiente máximo) o mediante el paso por cero de la segunda derivada (Laplaciano) en cualquier dirección. Como puede verse el problema de utilizar la primera derivada como enfoque para la detección de bordes resulta en la difícil localización del mismo al representar un método altamente direccional o bien cuando el borde no este bien definido.

Aunque el valor del gradiente permite detectar un borde, en ocasiones es importante tener la información sobre si el píxel esta en la transición positiva o negativa del gradiente, que equivaldría a conocer si el píxel esta del lado oscuro o de luz del borde.

6.5.1 Detección de bordes mediante la técnica de la segunda derivada

La técnica se basa en encontrar lo que se denomina pasos por cero (zero-crossing). Un paso por cero no es más que la transición de positivo a negativo o viceversa y es estimado a partir de la segunda derivada.

Si la definición de derivada para una función unidimensional puede ser considerada como:

$$\frac{\partial f}{\partial x} = f(x+1) - f(x) \quad (6.26)$$

A si mismo partir de la ecuación 6.26 puede decirse que la segunda derivada se define como:

$$\frac{\partial^2 f}{\partial x^2} = f(x+1) - 2f(x) + f(x-1) \quad (6.27)$$

Como fué mencionado la detección de bordes mediante la segunda derivada se basa en el paso por cero del valor del gradiente en cualquier dirección, dicha característica de hacer a este método insensible a la rotación es llamada isotrópica.

El operador Laplaciano es un filtro isotrópico definido por la siguiente expresión:

$$\nabla^2 I(x, y) = \frac{\partial^2 I(x, y)}{\partial x^2} + \frac{\partial^2 I(x, y)}{\partial y^2} \quad (6.28)$$

Este filtro al utilizar derivadas es lineal. Considerando la ecuación 6.27 en la expresión 6.28 del Laplaciano se tiene:

$$\frac{\partial^2 I(x, y)}{\partial x^2} = I(x+1, y) - 2I(x, y) + I(x-1, y) \quad (6.29)$$

$$\frac{\partial^2 I(x, y)}{\partial y^2} = I(x, y+1) - 2I(x, y) + I(x, y-1) \quad (6.30)$$

Por lo que si se juntan las expresiones 6.29 y 6.30 para formar 6.28, se tiene:

$$\nabla^2 I(x, y) = I(x+1, y) + I(x-1, y) + I(x, y+1) + I(x, y-1) - 4I(x, y) \quad (6.31)$$

Si se expresara la ecuación anterior en términos de un filtro que se pudiera utilizar para el procesamiento espacial de una imagen este quedaría definido por el mostrado en la Figura 6.26.

0	1	0
1	-4	1
0	1	0

Figura 6.26 Filtro que representa el cálculo del Laplaciano de forma espacial sobre una imagen, obtenido a partir de la ecuación 6.27.

1	1	1
1	-8	1
1	1	1

Figura 6.27 Filtro del Laplaciano con la extensión de los vecinos diagonales al píxel central.

Un ejemplo de la aplicación del filtro Laplaciano sobre una imagen en escala de grises es mostrado en la figura 6.28. En ella se muestran los diferentes efectos de aplicar la doble derivada en cada una de las direcciones y por ultimo el resultado de la suma de las dos tal y como es definido el Laplaciano. La ecuación definida en 6.28 y expresada en forma de filtro en la figura 6.26 no contempla las variaciones en los vecinos diagonales al píxel en cuestión, los cuales pueden incorporarse al filtro, con la consideración de que al hacer esto aumentaremos en 4 el número de unos contenidos en el filtro por ello habrá que aumentar el coeficiente central a 8. Considerando lo anterior el filtro Laplaciano quedaría definido tal como se describe en la Figura 6.27.

Como puede observarse del cálculo de los filtros Laplacianos, tienen una respuesta de cero en regiones en escala de grises uniformes y diferentes para regiones con

escalas de grises variantes. Esto es debido a que este tipo de filtros se comportan como filtros pasa altas, la suma de los coeficientes es cero.

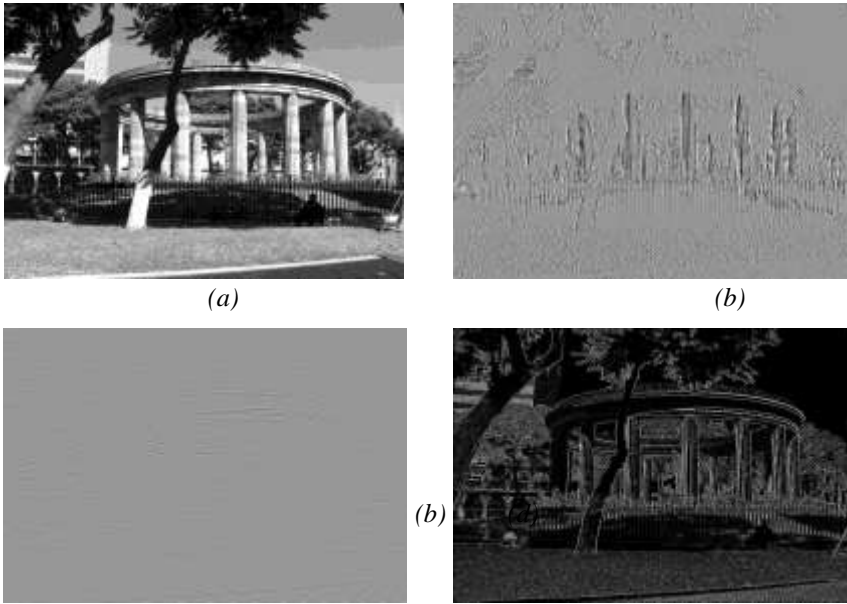


Figura 6.28 Utilización del filtro Laplaciano. (a) Figura original, (b) derivada parcial de segundo orden horizontal $\partial^2 I(x, y) / \partial x^2$, (c) derivada parcial del segundo orden vertical $\partial^2 I(x, y) / \partial y^2$ y (d) el operador Laplaciano $\nabla^2 I(x, y)$.

6.6 MEJORA DE NITIDEZ EN LAS IMÁGENES

Si se aplica el operador Laplaciano a una imagen obtendremos sus bordes. Sin embargo si lo que se desea es mejorar la nitidez de una imagen entonces lo que sería necesario hacer será conservar la información de baja frecuencia de la imagen original y enfatizar los detalles presentes en la imagen a través del filtro Laplaciano. Para realizar este efecto, es necesario restar a la imagen original una versión escalada del valor del filtro Laplaciano. Por lo que la imagen con una nitidez mejorada quedaría definida como se describe en la siguiente ecuación:

$$I(x, y)_{Mejorada} = I(x, y) - w \cdot \nabla^2 I(x, y) \quad (6.32)$$

La Figura 6.29 muestra la idea por medio de la cual la imagen mejora en nitidez al hacer más evidente la presencia de sus bordes. Para facilitar la explicación se considera exclusivamente el caso unidimensional.

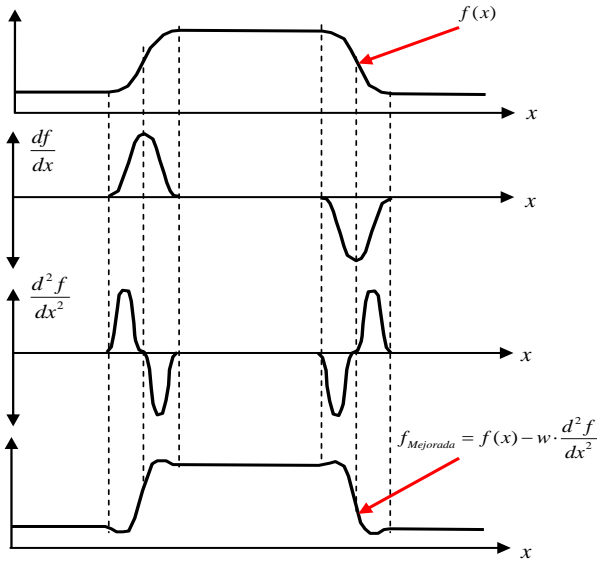


Figura 6.29 Aumento de la nitidez mediante la aplicación de la segunda derivada. Al restar un factor de la segunda derivada a la función permite maximizar la presencia de los contornos en la imagen.

El efecto de mejorar la nitidez de una imagen se puede realizar en una sola pasada, puesto que si se considera $w = 1$, se tiene que

$$I(x, y)_{Mejorada} = I(x, y) - (1) \cdot \nabla^2 I(x, y) \quad (6.33)$$

Y considerando que

$$\nabla^2 I(x, y) = I(x+1, y) + I(x-1, y) + I(x, y+1) + I(x, y-1) - 4I(x, y) \quad (6.34)$$

La figura 6.30 muestra la mejora de la nitidez de una imagen, utilizando a $w = 1$



(a)

(b)

Figura 6.30 Aplicación del filtro Laplaciano para la mejora de nitidez de una imagen. (a) Imagen original y (b) la imagen resultado de aplicar $I(x, y)_{Mejorada} = I(x, y) - w \cdot \nabla^2 I(x, y)$, considerando $w=1$.

Se tendría que

$$I(x, y)_{Mejorada} = 5I(x, y) - [I(x+1, y) + I(x-1, y) + I(x, y+1) + I(x, y-1)] \quad (6.35)$$

O bien expresado en un filtro, tendríamos que su matriz de coeficientes estaría definida por

$$I(x, y)_{Mejorada} = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & -1 \end{bmatrix} \quad (6.36)$$

6.6.1 Utilización de las herramientas de MatLAB para la implementación del filtro Laplaciano y mejora de la nitidez de imagen.

En esta sección se describe la utilización de las herramientas de MatLAB para mejorar la nitidez de imágenes por considerarlo un ejercicio integrador, ya que para mejorar una imagen es necesario calcular el Laplaciano y a partir de él generar la mejora de la imagen. En esta sección se revisará la forma de llevar a cabo esta tarea desde dos perspectivas. La primera de utilizar a MatLAB como lenguaje de alto nivel, programando así manualmente el cálculo del Laplaciano para después utilizarlo en la mejora de la nitidez de la imagen. La segunda es utilizar las librerías de bloques para el procesamiento de imágenes y video de Simulink, de tal forma que las imágenes cuya nitidez será mejorada sean tomadas de una Webcam procesando los frames en tiempo real.

El código MatLAB para mejorar la nitidez de una imagen puede ser dividido en dos sencillos pasos, en el primero se calcula el filtro Laplaciano a partir del filtro descrito en la figura 6.27, en el segundo se mejora la nitidez de la imagen considerando la información de baja frecuencia contenida en la imagen original menos el valor del

Laplaciano (véase ecuación 6.32) el cual incorpora la información que permite realzar los detalles de la imagen.

El programa 6.2 muestra la codificación en MatLAB para la mejora de la nitidez de una imagen mediante el uso del Laplaciano.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Mejora de la nitidez de una imagen
% utilizando el operador Laplaciano
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Erik Cuevas, Daniel Zaldivar, Marco Pérez
% Versión: 25/10/07
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Se define el factor con el que el operador Laplaciano
%afecte a la imagen (véase ecuación 6.32)
w=1;
%Se obtienen los valores de dimensión de la imagen:
[m n]=size(Im);
%Se convierte la imagen a doblé para evitar problemas
en %la conversión del tipo de dato:
Im=doublé(Im);
%Se crea la matriz L con ceros:
L=seros(size(Im));

%PRIMERA PARTE.
%Se aplica el filtro Laplaciano:
for r=2:m-1
    for c=2:n-1
        L(r,c)=Im(r-1,c-1)+Im(r-1,c)+Im(r-1,c+1)...
            + Im(r,c-1)-8*Im(r,c)+Im(r,c+1)...
            + Im(r+1,c-1)+Im(r+1,c)+Im(r+1,c+1)...
    end
end
%SEGUNDA PARTE.
%Se calculan los nuevos pixeles de la imagen cuya
nitidez %se pretende mejorar (véase fórmula 6.32):
Init=Im-w*L;
%Se obtiene el valor mínimo para la imagen Init:
VminInit=min(min(Init));
%Utilizando el valor mínimo se desplaza para evitar
%negativos:
GradOffL=Init-VminInit;
%Se obtiene el valor máximo para normalizar a 1:
VmaxInit=max(max(GradOffL));
%Se normalizan los gradientes a 255:
InitN=(GradOffL/VmaxInit)*255;
%Se convierte la imagen para el despliegue:
GInitN=uint8(InitN);
```

Programa 6.2 Mejora de la nitidez de una Imagen a través de la aplicación del operador Laplaciano con MatLAB®.

Para poder generar a través de la utilización de la librería de bloques de procesamiento de imagen y video de Simulink un programa que permita mediante la utilización del operador Laplaciano la mejora de la nitidez de frames de video capturados de una Webcam, es necesario utilizar bloques cuya funcionalidad ya ha sido tratada en este capítulo. La figura 6.31 muestra el programa realizado para este fin en Simulink.

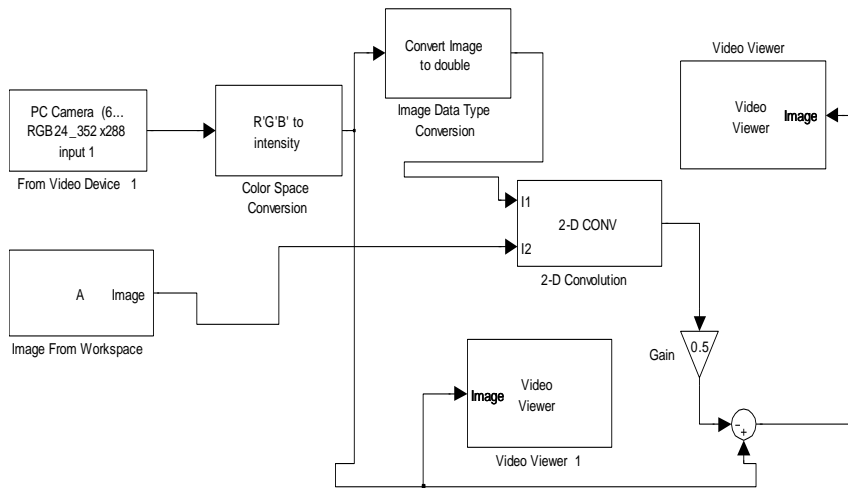


Figura 6.31 Programa que mejora la nitidez aplicando el operador Laplaciano de las imágenes capturadas a través de la Webcam.

Como puede notarse en este programa vuelve a hacerse uso del bloque “2-D Convolution” el cual realiza el cálculo del Laplaciano de la imagen que es capturada instantáneamente de la Webcam.

La imagen capturada por la cámara es transformada primero a escala de grises y convertida al tipo de dato double a través del bloque “Image Data Type Conversion”. Una vez teniendo la imagen este formato es utilizada como entrada al bloque “2-D Convolution” el cual convoluciona espacialmente esta imagen con el filtro Laplaciano definido en *A* desde la consola de MatLAB. El filtro definido en *A* es el descrito en la figura 6.27, el cual al ser isotrópico (no sensible a la rotación) no es necesario tener ninguna consideración, ya que como fue explicado la matriz definida en *I2* en este bloque es rotada 180° antes de ser utilizada en la multiplicación por los coeficientes de la imagen definida en *I1*. El valor de *A* puede de esta manera ser definido por línea de comandos por:


```
>>A=[1 1 1; 1 -8 1; 1 1 1];
```

El valor calculado del Laplaciano es multiplicado por un factor (0.5) que define la forma en como este operador modificara a la imagen a la cual se desea mejorar su nitidez. Después este valor es restado a la imagen original produciendo así a la imagen mejorada. Un problema importante es que este operador como fué mencionado en la sección 6.5 funciona como un filtro pasa altas, por lo que ruido presente en la imagen original es maximizado también, es por eso que si se ejecuta este programa se verá claramente como la nitidez de la imagen resultado mejora, sin embargo de la misma manera aparecerán algunos artefactos producto del ruido propio del frames que se procesan. La figura 6.32 muestra el resultado de ejecutar el programa en Simulink definido en la figura 6.31

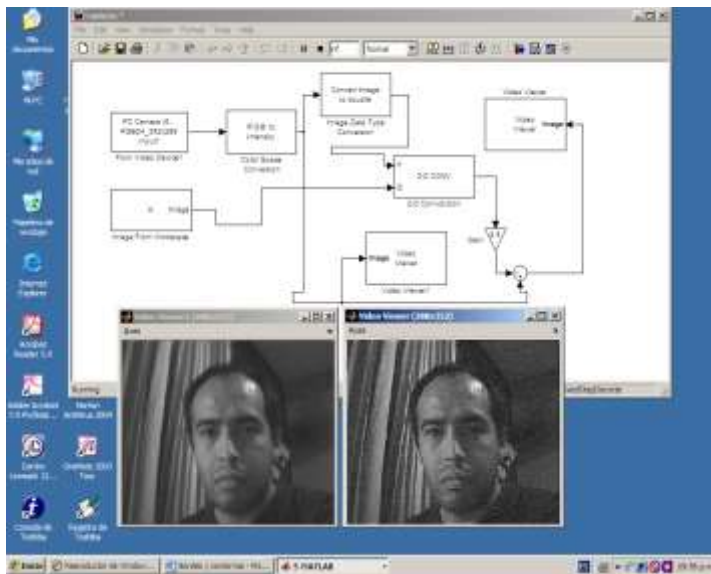


Figura 6.32 Resultado de haber ejecutado el programa en Simulink mostrado en la Figura 6.31.

6.7 EL FILTRO CANNY

Un conocido método para la detección de bordes en imágenes es el filtro de Canny. Este método es basa en la aplicación de una serie de filtros en direcciones y resoluciones diferentes, los cuales al final son combinados en un resultado único. El método intenta alcanzar 3 diferentes objetivos, (a) minimizar el número de bordes falsos, (b) una mejor localización de los bordes en la imagen y (c) entregar una imagen cuyo ancho de borde es un píxel. El filtro de Canny es en esencia un filtro basado en métodos de gradiente, pero sin embargo usa también para la localización de bordes como criterio la segunda derivada o Laplaciano. La mayoría de las veces

este algoritmo es utilizado en su forma sencilla, es decir configurando solamente el parámetro de suavizado σ . La Figura 6.33 muestra ejemplos de la aplicación de este algoritmo utilizando para ello diferentes valores del parámetro σ .

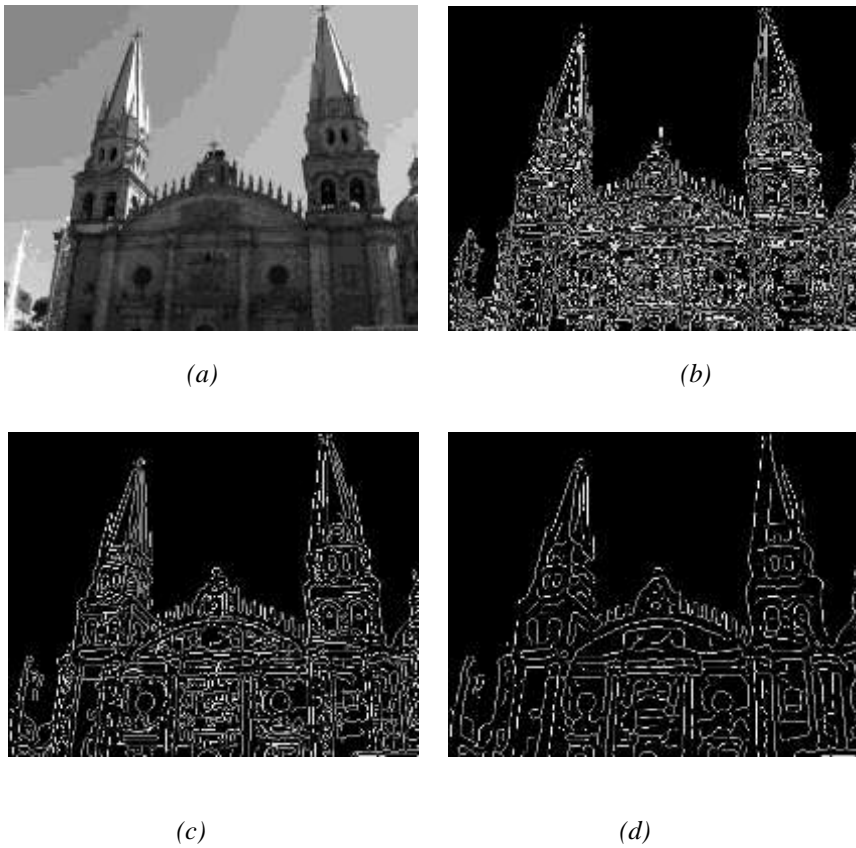


Figura 6.33 Algoritmo de Canny aplicado a una imagen. (a) Imagen original, (b) bordes de la imagen con $\sigma=0.1$, (c) bordes de la imagen con $\sigma=1$ y (d) bordes de la imagen con $\sigma=2$.

6.7.1 Herramientas de MatLAB que implementan el filtro de Canny.

Debido a la amplia utilización de este filtro como etapa previa en la segmentación y clasificación de objetos y a su robustez en la determinación de bordes, el filtro es normalmente implementado en la mayoría de las librerías comerciales y herramientas de procesamiento digital de imágenes. En MatLAB el algoritmo de Canny puede ser calculado mediante la función `Edge`, su formato general es

```
BW = edge(I, 'canny', U , sigma);
```

Donde `BW` es la imagen con los bordes extraídos a partir del algoritmo de Canny, `I` es la imagen en escala de grises a la cual se le extraerán los bordes, `U` es el valor del umbral a partir del cual los píxeles que tengan un gradiente mayor a este serán considerados candidatos para ser bordes y `sigma`, el cual es el parámetro (σ) de suavizado que tiene el efecto de minimizar el número de bordes falsos. También es posible utilizar el algoritmo de Canny en las librerías de procesamiento de imagen y video de Simulink, para ello solo bastaría con configurar el bloque “Edge Detection”, indicándole (1) que se utilizara el método de Canny, (2) si se usara un umbral definido por el usuario o no, (3) el porcentaje de los bordes que considerados falsos serán eliminados y (4) el parámetro σ . La Figura 6-34 muestra el bloque “Edge Detection” y su mascarilla de configuración.

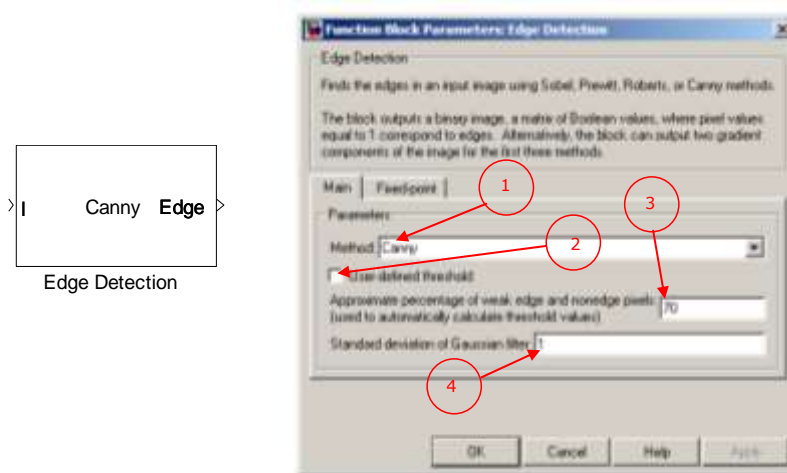


Figura 6.34 Bloque “Edge Detection” y su mascarilla de configuración.

