Capítulo 7

DETERMINACION DE ESQUINAS

Los bordes tratados en el capítulo anterior eran definidos como puntos que tienen un alto valor del gradiente mientras que las esquinas son puntos prominentes contenidos en una imagen, caracterizados por presentar también un alto valor del gradiente pero a diferencia de los bordes, este alto valor del gradiente no solo se manifiesta en una dirección sino en diferentes. Considerando la anterior definición podemos imaginar a las esquinas como puntos en la imagen que colindan con diferentes bordes a la vez.

Las esquinas pueden ser utilizadas en una amplia gama de aplicaciones tales como el seguimiento de objetos en secuencia de video (tracking), para ordenar las estructuras de objetos en visión estereoscópica, como puntos de referencia en la medición de características geométricas de objetos o bien en la calibración de cámaras para sistemas de visión. Algunas de las ventajas de las esquinas sobre otras características obtenidas de una imagen son su robustez al cambio de perspectiva así como a su confiabilidad en su localización ante diferentes condiciones de iluminación.

1.1 ESQUINAS EN UNA IMAGEN

Considerando que las esquinas definen una característica robusta y evidente de una imagen, su determinación, como será visto mas adelante, no es sencilla. Un algoritmo para la detección de las esquinas debe de reunir algunos aspectos importantes tales como:

- Detectar esquinas “importantes” de las “no importantes”.
- Detectar las esquinas en presencia del ruido propio de la imagen.
- Ser rápido en la ejecución para permitir su implementación en tiempo real.

Como es natural suponer existen varios enfoques que son capaces de cumplir estas características en donde la mayoría de ellos se basan en la medición del valor del gradiente en el punto que se considera como potencialmente esquina. Mientras un borde se define como un punto de la imagen donde el valor del gradiente es especialmente alto pero en una sola dirección, una esquina también tiene valores altos del gradiente pero en más de una dirección simultáneamente.

La mayoría de los algoritmos utilizados para la detección de esquinas utilizan el criterio de la primera o la segunda derivada sobre la imagen en la dirección x o y como aproximación del valor del gradiente. Un algoritmo representativo de esta clase de métodos es el **Detector de Harris** (otras veces llamado detector de puntos Plessey). Aunque existen algunos otros detectores que presentan interesantes prestaciones y propiedades el detector Harris es por hoy el método usado con mayor frecuencia, por esta razón será el algoritmo que se explicará con una mayor profundidad.

1.2 ALGORITMO DE HARRIS

El Algoritmo desarrollado por Harris y Stephens es un algoritmo que como fué comentado se basa en la idea de que una esquina es un punto de la imagen donde su valor del gradiente muestra un valor alto en varias direcciones simultáneamente. Este algoritmo es robusto en diferenciar las esquinas de los bordes los cuales muestran un valor alto del gradiente pero en una sola dirección, además este detector permite localizar las esquinas de manera robusta a la orientación por lo que no importa la orientación de la esquina.

1.2.1 Matriz de estructuras

El cálculo del algoritmo de Harris se basa en desarrollo de la primera derivada parcial en un píxel $I(x,y)$ en dirección horizontal y vertical tal que:

$$I_x(x,y) = \frac{\partial I(x,y)}{\partial x} \quad \text{y} \quad I_y(x,y) = \frac{\partial I(x,y)}{\partial y} \tag{7.1}$$

Para cada píxel de la imagen (x,y) son calculadas tres diferentes cantidades que serán denominadas $HE_{11}(x,y)$, $HE_{22}(x,y)$ y $HE_{12}(x,y)$, donde:

$$HE_{11}(x,y) = I_x^2(x,y) \tag{7.2}$$

$$HE_{22}(x,y) = I_y^2(x,y) \tag{7.3}$$

$$HE_{12}(x,y) = I_x(x,y) \cdot I_y(x,y) \tag{7.4}$$

Estos valores pueden son interpretados como aproximaciones de los elementos de la matriz de estructuras definida como HE tal que:

$$HE = \begin{bmatrix} HE_{11} & HE_{12} \\ HE_{21} & HE_{22} \end{bmatrix} \tag{7.5}$$

donde $HE_{12} = HE_{21}$.

1.2.2 El Filtrado de la matriz de estructuras

Para la localización de las esquinas utilizando el algoritmo de Harris es necesario suavizar los valores de cada uno de los elementos de la matriz de estructuras mediante su convolución por un filtro Gaussiano H_σ, tal que la matriz HE se redefine como la matriz E, tal que:

$$E = \begin{bmatrix} HE_{11} * H_\sigma & HE_{12} * H_\sigma \\ HE_{21} * H_\sigma & HE_{22} * H_\sigma \end{bmatrix} = \begin{bmatrix} A & C \\ C & B \end{bmatrix} \tag{7.6}$$

1.2.3 Calculo de los valores y vectores propios

La matriz E debido a su simetría puede ser diagonalizada de tal forma que:

$$E' = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} \tag{7.7}$$

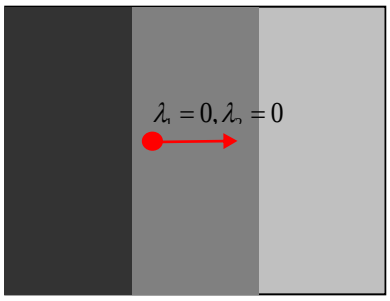
Donde λ_1 y λ_2 son los valores propios de de la matriz E, dichos valores son calculados de acuerdo a:

$$\lambda_{1,2} = \frac{\mathrm{tr}(E)}{2} \pm \sqrt{\left(\frac{\mathrm{tr}(E)}{2}\right)^2 - \det(E)} \tag{7.8}$$

Donde $\mathrm{tr}(E)$ implica la traza de la matriz E y $\det(E)$ el determinante de la matriz E. Desarrollando las operaciones de la traza y determinante de la ecuación 7.7 se obtiene:

$$\lambda_{1,2} = \frac{1}{2}(A + B \pm \sqrt{A^2 - 2AB + B^2 + 4C^2}) \tag{7.9}$$

Ambos valores propios λ_1 y λ_2 son positivos y contienen información importante acerca de las estructuras locales contenidas en la imagen. Dentro de una región de la imagen cuyo contenido homogéneo sea un valor de intensidad determinado el valor de $E = 0$ por lo que también los valores propios en esa región son $\lambda_1 = \lambda_2 = 0$. Sin embargo en un cambio en escalón de intensidad el valor de $\lambda_1 > 0$ y $\lambda_2 = 0$, independientemente de la orientación del borde. De lo anterior puede decirse que los valores propios codifican la magnitud del gradiente, mientras que los vectores propios codifican la dirección. La figura 7.1 muestra como los valores λ_1 y λ_2 dan información esencial de las estructuras contenidas en una imagen.



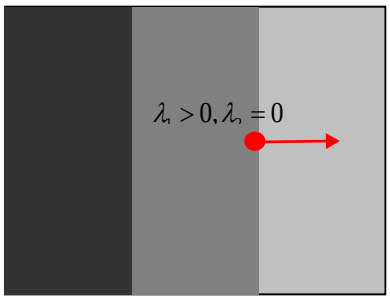


Figura 7.1 Relación de estructuras en una imagen y sus valores propios.

Un punto esquina posee un valor alto del gradiente tanto en la dirección principal, la cual corresponde al valor de ambos valores propios, así como en la dirección normal a la principal, la cual corresponde al valor mas pequeño de los valores propios. De lo anterior puede ser concluido que para que un punto sea considerado como esquina deben los valores propios λ_1 y λ_2 tener valores significativos. Debido a que $A, C \geq 0$, puede entonces advertirse que la $\text{tr}(E) > 0$, con esto y observando las ecuaciones 7.8 y 7.9 puede ser aceptado que $|\lambda_1| \geq |\lambda_2|$. Considerando lo anterior y la aseveración de que un punto esquina debe de tener en ambos valores propios una magnitud significativa, entonces para la determinación de una esquina se considera solamente el valor de λ_2 que es el más pequeño, puesto que si este valor es grande con mayor razón lo será λ_1.

1.2.4 Función del valor de la esquina (V).

Como puede ser observado de las ecuaciones 7.8 y 7.9 la diferencia entre ambos valores propios se establece como:

$$\lambda_1 - \lambda_2 = 2 \cdot \sqrt{\left(\frac{\text{tr}(E)}{2}\right)^2 - \det(E)} \tag{7.10}$$

Donde en todo caso se cumple que $(1/4) \cdot \text{tr}(E) > \det(E)$. En un punto esquina la diferencia entre ambos valores propios es pequeña. El algoritmo de Harris utiliza esta propiedad como significativa e implementa como medida de valor de la esquina la función:

$$V(x, y) = \det(E) - \alpha(\text{tr}(E))^2 \tag{7.11}$$

Lo cual según la ecuación 7.6 conduce a:

$$V(x, y) = (A \cdot B - C^2) - \alpha(A + B)^2 \tag{7.12}$$

Donde el parámetro α controla la sensibilidad del algoritmo. $V(x, y)$ es definida como la función del valor de la esquina y su valor entre más grande sea mejor caracteriza a un punto esquina en (x, y). El valor de α es fijo y se encuentra en el intervalo de [0.04, 0.25]. Si el valor de α es grande, menos sensible es a los puntos esquinas, por lo que si se desea es configurar el algoritmo de Harris para una mayor sensibilidad el valor de α es 0.04. Por lo anterior es evidente que si α es pequeño una mayor cantidad de puntos esquinas serán encontrados en una imagen determinada. La figura 7.2 muestra el valor de $V(x, y)$ de una imagen. De la imagen puede ser observado como los valores esquina de la imagen experimentan un valor grande de la función $V(x, y)$, mientras en los bordes y regiones homogéneas simplemente el valor será casi cero.

Figura 7.2 Valor de $V(x, y)$ de una imagen

1.2.5 Determinación de los puntos de esquina

Un punto de la imagen (x, y) es considerado como punto esquina potencial si la condición

$$V(x, y) > t_h \tag{7.13}$$

se cumple. Donde t_h es el umbral y normalmente su valor típico se encuentra dentro del intervalo de 900 a 10000, dependiendo del contenido de la imagen. Por lo que a partir de la aplicación de la ecuación 7.13 se tendrá una matriz binaria conteniendo unos (verdadero) donde la condición fué cumplida y ceros (falso) donde no fué válida.

Para impedir la localización de regiones altamente pobladas de puntos esquinas que se calculan debido a una alta sensibilidad del valor α, se seleccionan sólo aquellos píxeles cuyo valor $V(x, y)$ es el más grande dentro de una determinada vecindad. La figura 7.3 muestra una ilustración de este proceso.

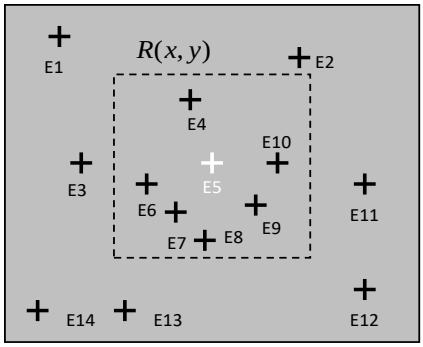


Figura 7.3 Proceso de obtención de la esquina significante. De todas las esquinas encontradas mediante la aplicación del umbral t_h, se selecciona aquel punto esquina cuyo valor $V(x,y)$ es el máximo en una vecindad definida $R(x,y)$. En este caso el valor de la esquina E5 pose el valor máximo de $V(x,y)$ en comparación a los demás valores E4, E6, E7, E8, E9 y E10 que se encuentran dentro de la región de vecindad definida por $R(x,y)$.

1.2.6 Implementación del algoritmo

Debido a que el algoritmo de Harris es el primer proceso "complejo" que se desarrolla en este libro en esta sección se presenta una vista integradora del algoritmo y los pasos que habrán de desarrollarse para la localización de los puntos esquina en una imagen. Todos los pasos resumidos son mostrados en el algoritmo 7.1.

Detector de Harris($I(x,y)$)

1: Suavizado de la imagen original por un prefiltro: $I' = I * H_p$

2: Paso 1. Calcular la función del valor de la esquina $V(x,y)$.

3: Se calcula los gradientes en la dirección horizontal y vertical.

4: $I_x = I' * H_x$ y $I_y = I' * H_y$.

5: Se computan los componentes de la matriz de estructuras.

6:
$$HE = \begin{bmatrix} HE_{11} & HE_{12} \\ HE_{21} & HE_{22} \end{bmatrix}$$

7: donde

8: $HE_{11}(x,y) = I_x^2(x,y)$

9: $HE_{22}(x,y) = I_y^2(x,y)$

10: $HE_{12}(x,y) = I_x(x,y) \cdot I_y(x,y)$

11: donde además $HE_{12} = HE_{21}$.

12: Se suavizan los valores de los componentes por un filtro Gaussiano.

13:
$$E = \begin{bmatrix} HE_{11} * H_\sigma & HE_{12} * H_\sigma \\ HE_{21} * H_\sigma & HE_{22} * H_\sigma \end{bmatrix} = \begin{bmatrix} A & C \\ C & B \end{bmatrix}$$

14: Se calcula el valor de la función del valor de la esquina

15: $V(x,y) = (A \cdot B - C^2) - \alpha(A+B)^2$

16: Paso 2. Se binariza el valor de $V(x,y)$.

17: Se obtiene una matriz binaria de aplicar el umbral t_h.

18: $U(x,y) = V(x,y) > t_h$

19: Paso 3. Se localizan los puntos esquinas significativos. Donde se construye la matriz binaria $S(x,y)$ la cual contendrá unos en aquellos píxeles que sean esquinas significativas y ceros donde la esquina no sea considerada como tal.

20: Se define una región de vecindad $R(x,y)$.

21: Se inicializa $S(x,y)$ con todos sus valores en cero.

22: **for** todas las coordenadas de la imagen (x,y) **do**

23: **if** $(U(x,y) == 1)$ **then**
24: **if** $(V(x,y) \geq$ a cada uno de los valores de $R(x,y))$ **then**
25: $S(x,y) = 1$
26: Fin del **for**

Algoritmo 7.1 Algoritmo de Harris para la detección de puntos esquinas.

Como puede ser observado del algoritmo 7.1 la imagen original deberá ser filtrada con el objetivo de eliminar el posible ruido contenido en la misma, en este paso un filtro promediador pasa-bajas sencillo de 3x3 sería apropiado. Una vez que la imagen ha sido filtrada se obtienen los gradientes de la imagen en las direcciones horizontal y vertical, para ello se cuentan con diferentes opciones que pueden ir desde filtros sencillos como:

$$H_x = \begin{bmatrix} -0.5 & \underline{0} & 0.5 \end{bmatrix} \quad \text{y} \quad H_y = \begin{bmatrix} -0.5 \\ \underline{0} \\ 0.5 \end{bmatrix} \tag{7.14}$$

o los de Sobel

$$H_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad \text{y} \quad H_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \tag{7.15}$$

Para más detalles y posibilidades de filtros para calcular el gradiente véase el capítulo 6. Habiendo obtenido los valores de los gradientes se procede a calcular los valores de la matriz de estructuras como se describe en las líneas de la 6 a la 11 del algoritmo 7.1.

Con los valores de HE calculados de calcula E como ilustra las líneas 12 y 13 mediante la aplicación de un filtro Gaussiano. El filtro Gaussiano tal y como fue tratado en la sección 5.4.1 de este libro tiene una matriz de coeficientes como la ilustrada por la figura 7.4.

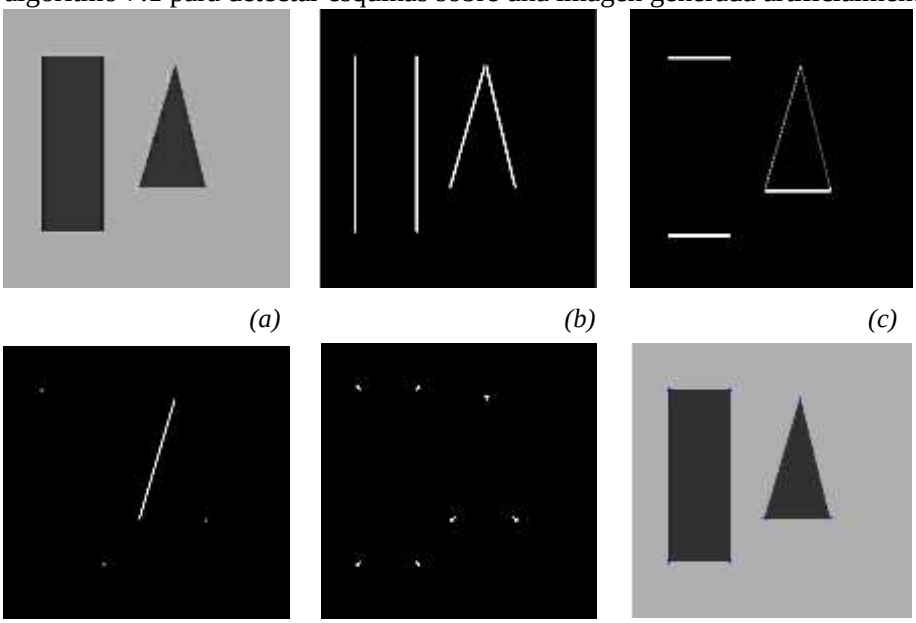
0	1	2	1	0
1	3	5	3	1
2	5	9	5	2
1	3	5	3	1
0	1	2	1	0

Figura 7.4 El filtro Gauss para suavizado.

Con el valor de E calculado se procede a obtener la magnitud para cada píxel de la función del valor esquina según línea 15 del algoritmo.

Considerando un valor apropiado de t_h, se obtiene la matriz binaria $U(x,y)$ según las líneas 17 y 18. A este nivel del algoritmo tenemos en los elementos de la matriz $U(x,y)$ la información de aquellos puntos considerados como potencialmente esquinas, ya que todos los elementos que son unos tendrán un valor de $V(x,y)$ que es mayor al considerado como el necesario (t_h). Sin embargo

debido a las características propias de sensibilidad del algoritmo (controlada por el parámetro α) y ruido contenido en la imagen, existirán varios puntos esquinas concentrados cerca de un punto cuyo valor máximo de $V(x,y)$ en comparación a ellos, hace suponer que es el verdadero punto esquina. Para resolver esta situación se implementa el paso 3 del algoritmo descrito en las líneas de la 19 a la 26. En esta parte utilizando la información contenida en la matriz $U(x,y)$ se buscan los "verdaderos puntos esquinas", que son aquellos que experimentan un valor máximo de $V(x,y)$ en comparación a los demás puntos esquinas contenidos en una región de vecindad $R(x,y)$ centrada alrededor del punto de prueba. La figura 7.5 muestra una secuencia de imágenes con resultados parciales de la aplicación del algoritmo 7.1 para detectar esquinas sobre una imagen generada artificialmente.



(a) *(b)* *(c)*

(d) *(e)* *(f)*

Figura 7.5 Aplicación del algoritmo de Harris a una imagen generada artificialmente con algunos resultados parciales. (a) Imagen original, (b) imagen $I_x^2(x,y)$, (c) imagen $I_y^2(x,y)$, (d) imagen $I_x(x,y)\cdot I_y(x,y)$, imagen de $V(x,y)$ y (f) imagen original con el localización de los "puntos esquinas verdaderos".

1.3 DETERMINACIÓN DE LOS PUNTOS ESQUINAS USANDO MATLAB®.

MatLAB no dispone de ninguna función que permita extraer de manera directa las esquinas de una imagen, por lo que en este caso se implementara el programa de manera directa que permitirá realizar este proceso. La forma de programar el algoritmo de Harris seguirá los pasos indicados por el algoritmo 7.1. El programa 7.1 muestra el programa realizado en MatLAB que implementa completamente el operador de Harris.

A continuación serán tratados algunos aspectos importantes que deben ser tomados en cuenta en la programación del algoritmo de Harris. El primer aspecto interesante se da al momento de que las matrices tanto $U(x,y)$ como $S(x,y)$ son inicializadas con ceros. Esta acción es una situación estándar en aquellas aplicaciones donde se hace localización de píxeles con alguna característica especial, los cuales en cantidad evidentemente serán mucho menos en comparación al tamaño de la imagen.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Implementación del algoritmo de Harris
% para la detección de esquinas en una imagen
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Erik Cuevas, Daniel Zaldivar, Marco Pérez
% Versión: 11/01/08
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Se obtiene el tamaño de la imagen Ir a la cual se
%le extraerán las esquinas (PASO 1).
[m n]=size(Ir);
%Se inicializan las Matrices U y S con ceros
U=zeros(size(Ir));
S=zeros(size(Ir));
%Se crea la matriz de coeficientes del Pre-filtro
%suavizador
h=ones(3,3)/9;
%Se cambia el tipo de dato de la imagen original a
%double
Id=double(Ir);
%Se filtra la imagen con el filtro h promediador
If=imfilter(Id,h);
%Se generan las matrices de coeficientes para %calcular
el gradiente horizontal Hx y vertical Hy
Hx=[-0.5 0 0.5];
Hy=[-0.5;0;0.5];
%Se calculan los gradientes horizontal y vertical
Ix=imfilter(If,Hx);
Iy=imfilter(If,Hy);
%Se obtienen los coeficientes de la matriz de
%estructuras
HE11=Ix.*Ix;
HE22=Iy.*Iy;
HE12=Ix.*Iy; %y HE21
%Se crea la matriz del filtro Gaussiano
Hg=[0 1 2 1 0; 1 3 5 3 1;2 5 9 5 2;1 3 5 3 1;0 1 2 1
0];
Hg=Hg*(1/57);
%Se filtran los coeficientes de la matriz de
%estructuras con %el filtro Gaussiano
A=imfilter(HE11,Hg);
B=imfilter(HE22,Hg);
C=imfilter(HE12,Hg);
%Se fila el valor de alfa a 0.04 (Muy sensible)
alfa=0.04;
%Se obtiene la magnitud del valor de la esquina
Rp=A+B;  %Resultado parcial
Rp1=Rp.*Rp;   %Resultado parcial
%Valor de la esquina (matriz Q)
Q=((A.*B)-(C.*C))-(alfa*Rp1);
%Se fija el valor del umbral
th=1000;
%Se obtiene la matriz U (PASO 2).
U=Q>th;
%Se fija el valor de la vecindad
pixel=10;
%Se obtiene el valor mas grande de Q, de una vecindad
%definida por la variable píxel (PASO 3).

for r=1:m
    for c=1:n
        if(U(r,c))
```

```
                %Se  define  el  limite  izquierdo  de  la
%vecindad
                I1=[r-pixel 1];
                %Se define el limite derecho de la %vecindad
                I2=[r+pixel m];
                %Se  define  el  limite  superior  de  la
%vecindad
                I3=[c-pixel 1];
                %Se define el limite inferior de a de la
%vecindad
                I4=[c+pixel n];
                %Se definen posiciones teniendo en cuenta
%que su valor es relativo a r y c.
                datxi=max(I1);
                datxs=min(I2);
                datyi=max(I3);
                datys=min(I4);
                %Se extrae el bloque de la matriz Q
                Bloc=Q(datxi:1:datxs,datyi:1:datys);
                %Se obtiene el valor máximo de la %vecindad
                MaxB=max(max(Bloc));
                %Si el valor actual del píxel es el %máximo
entonces en esa posición se coloca un 1 en          %la
matriz S.
                if(Q(r,c)==MaxB)
                    S(r,c)=1;
                end
            end

        end
    end
    %Se despliegue la imagen original Ir
    imshow(Ir);
    %Se mantiene el objeto grafico para que los demás
    %comandos gráficos tengan efecto sobre la imagen Ir
    %desplegada
    hold on

    %Se grafican sobre la imagen Ir las esquinas %calculadas
    por  el  algoritmo  de  Harris  en  las  %posiciones  donde
    existen unos en la matriz S
    for r=1:m
        for c=1:n
            if(S(r,c))
           %Donde  hay  un  uno  se  añade  a  la  imagen  Ir  un
    %símbolo +
            plot(c,r,'+');
            end
         end
    end
```

Programa 7.1 Implementación en MatLAB del algoritmo de Harris.

Considerando lo anterior resulta mucho más práctico y rápido inicializar las matrices con ceros y solamente colocar en uno a aquellos elementos que cumplan alguna determinada propiedad.

El pre-filtro suavizador implementado es un filtro del tipo “Box” promediador de tamaño 3x3, mientras que los filtros utilizados para el cálculo de los gradientes son los definidos en la ecuación 7.14. Todas las operaciones realizadas entre las imágenes y los filtros en el programa 7.1

son desempeñadas por la función de MatLAB `imfilter` tratada en detalle en la sección 5.8.3 de este libro.

Las operaciones realizadas para el cálculo de los elementos de la matriz de estructuras como se tratan de procesos elemento a elemento se desempeñan aplicando el operador punto de MatLAB, esto es considerando las expresiones

$$A * B \tag{7.16}$$

$$A .* B \tag{7.17}$$

la primera (7.16) implica la multiplicación matricial entre las imágenes A y B, mientras que la segunda (7.17) define una multiplicación punto a punto entre todos los elementos de las matrices A y B.

Otra parte importante del programa 7.1 lo representa el paso 3, integrado por las instrucciones necesarias para la elección de los píxeles cuyo valor de $V(x, y)$ sea máximo dentro de un bloque considerado como la región de vecindad. Para programar tal hecho se recorre secuencialmente píxel a píxel la imagen encontrando el valor máximo de un bloque o región de vecindad establecida alrededor del punto en cuestión. Para ello se fijan los límites del bloque relativos al píxel en cuestión, en el caso del programa 7.1 se fija el intervalo igual a 10, que considerándolo en ambos sentidos sería 20. Una vez establecido los límites se extrae el bloque y se encuentra el máximo, si el máximo del bloque corresponde al píxel central del bloque, relativo al del proceso de barrido, se coloca un uno en esa posición en $S(x, y)$, de tal forma que donde se encuentren unos en $S(x, y)$ se encontraran los puntos esquinas de la imagen original. La figura 7.3 muestra el proceso de encontrar el máximo dentro de la vecindad considerada alrededor del punto en cuestión.

Una vez que se tiene los puntos almacenados en $S(x, y)$, se grafican. Para realizar este proceso primeramente se despliega la imagen mediante la función conocida de despliegue usada por MatLAB: `imshow`. Después mediante el comando `hold on` se mantiene la imagen como tapete gráfico y sobre ella se dibujan los puntos uno a uno mediante el comando `plot`, colocando en cada punto un signo de más +.

La figura 7.6 muestra la localización de los puntos esquina detectados, una vez ejecutado el programa 7.1 sobre una imagen ejemplo.

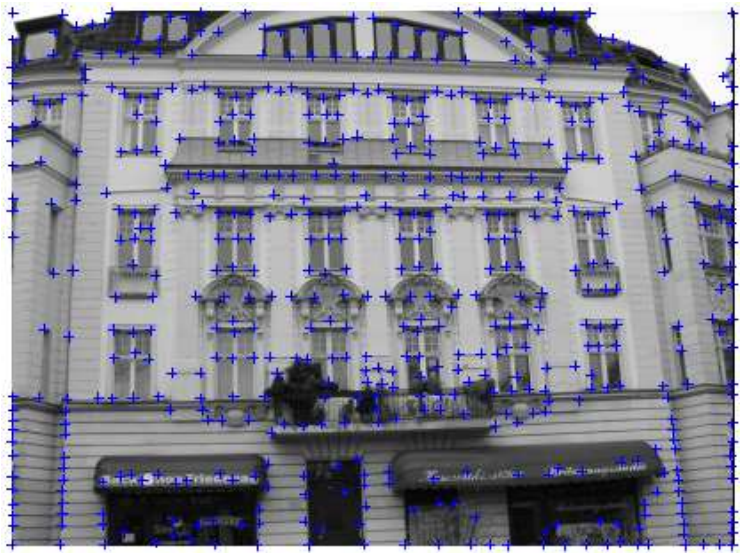


Figura 7.6 Localización de los puntos esquinas según el algoritmo de Harris.

1.4 DETERMINACIÓN DE LOS PUNTOS ESQUINAS USANDO LOS BLOQUES DE SIMULINK®.

Al igual que en el caso de las funciones de MatLAB tampoco existe bloque alguno que permita calcular los puntos esquinas de imágenes pertenecientes a frames de video, considerando lo anterior será necesario programar directamente el algoritmo de Harris mostrado en el algoritmo 7.1 mediante la utilización de bloques existentes en Simulink.

Para la detección de puntos esquinas se establece el programa en Simulink mostrado en la figura 7.7.

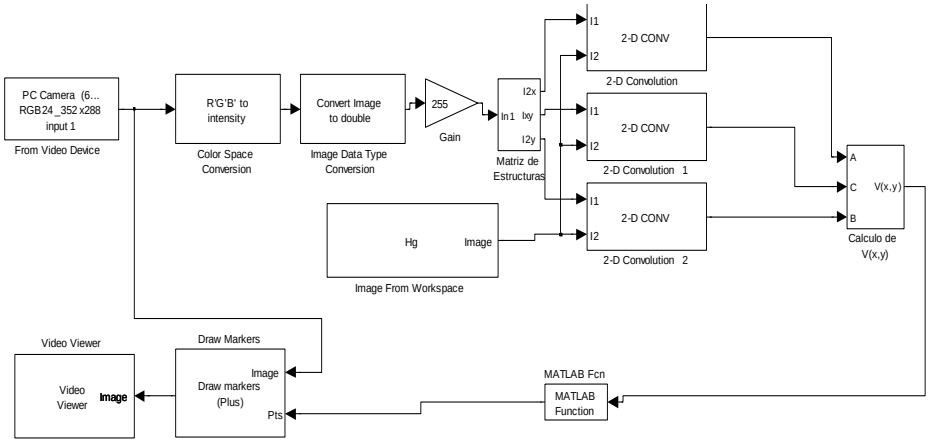


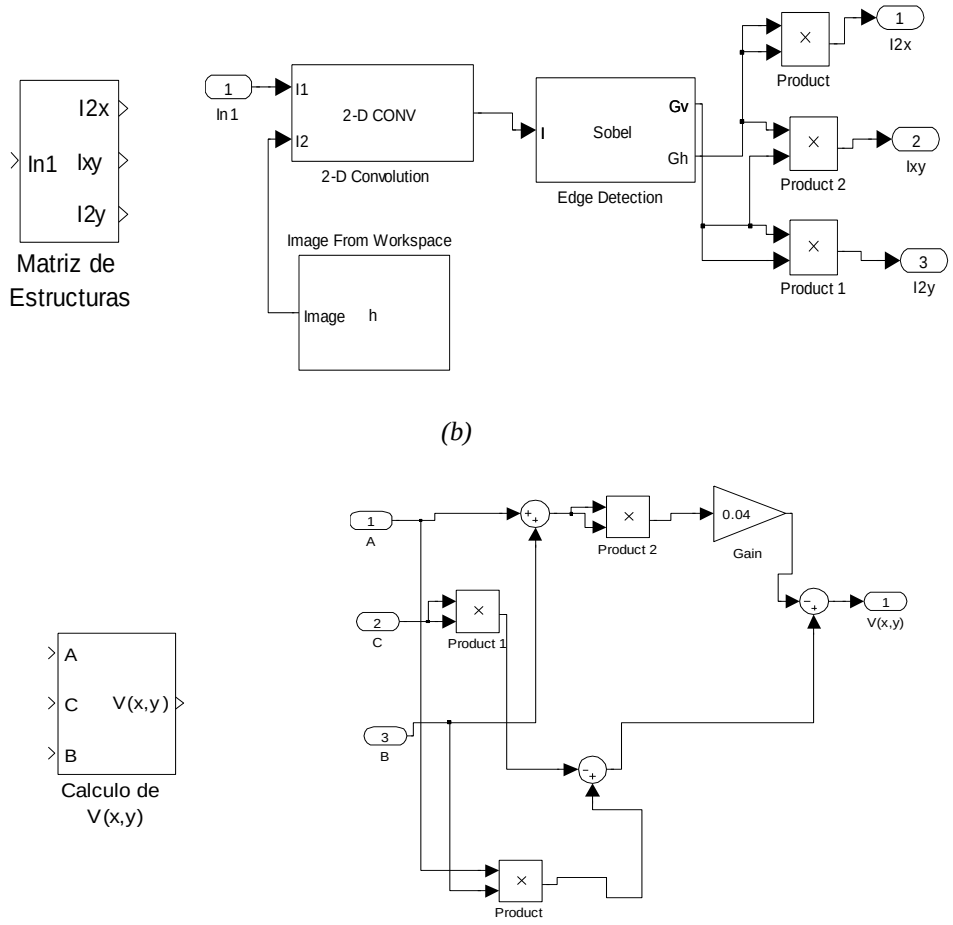
Figura 7.7 Programa en Simulink para la detección de esquinas de imágenes provenientes de la Webcam.

Como puede verse de la figura 7.7 el programa esta compuesto de bloques cuya funcionalidad y aplicaciones ya han sido tratados en capítulos anteriores, sólo que en esta ocasión se utilizó para mejorar la legibilidad del programa los llamados subsistemas. Los subsistemas son recursos que posee Simulink para empaquetar segmentos de programa en unidades monolíticas que permiten reducir el

espacio necesario para mostrar un programa. En el caso de Simulink lo anterior es una gran ventaja, ya que al tratarse de una forma de programar iconográfica normalmente el espacio necesario para representar un programa se vuelve grande conforme aumenta la longitud del algoritmo. En el caso de este programa se describen 2 subsistemas el etiquetado como **Matriz de estructuras** y como **Calculo de V(x,y)**.

La manera sencilla de generar un subsistema en Simulink es la de seleccionar con el ratón el segmento de programa que se quiera empaquetar y accionando el botón de la derecha se despliega un menú del cual se selecciona la opción “Create Subsystem”.

La figura 7.8 muestra el contenido de los subsistemas Matriz de estructuras y Cálculo de V(x,y).



(a) *(b)*

(c) *(d)*

Figura 7.8 Contenido de los subsistemas Matriz de estructuras y Cálculo de V(x,y). (a) Icono de subsistema Matriz de estructuras, (b) contenido del subsistema Matriz de estructuras, (c) icono del subsistema Cálculo de V(x,y) y (d) contenido del subsistema cálculo de V(x,y).

A continuación se describe el funcionamiento del programa ilustrado en la figura 7.7 y algunos detalles importantes de implementación. Primero la imagen captada por la Webcam a través del bloque “From Video Device” es convertida a imagen de intensidad utilizando el bloque “Color space conversión”. Un punto importante es convertir la imagen de intensidad del tipo de dato `uint8`

a `double` mediante el bloque "Image data type conversión", esto es un paso que no puede faltar en el sentido que las operaciones que se desempañaran más adelante necesitan para su correcta realización el tipo de dato `double`. La imagen que obtenemos hasta este punto varia en el intervalo de [0, 1], para lo cual será necesario escalar los datos para trabajar con el intervalo acostumbrado de 0 a 255. Sobre esta imagen ya preprocesada actúa el subsistema **Matriz de estructuras**, el cual primeramente filtra a la imagen mediante la actuación de un filtro pasabajas `h` del tipo "Box". Para que pueda funcionar correctamente este programa deberá de definirse en línea de comandos el valor de la matriz de coeficientes que en este caso será:

```
>>h=ones(3,3)/9;
```

Una vez filtrada la imagen se utiliza el bloque "Edge Detection" (véase capítulo 6) para calcular el valor de los gradientes horizontal (Gh) y vertical (Gv), en el caso de este programa se utilizo el operador Sobel cuyas matrices de coeficientes se encuentran definidas en la ecuación 7.15. Los valores de los gradientes son multiplicados para obtener los elementos de la matriz de estructuras definidos por la ecuación 7.5, el caso del gradiente vertical por si mismo para obtener (I_x^2 que es I2x), el del gradiente vertical por si mismo para obtener (I_y^2 que es I2y) y finalmente en el caso del gradiente vertical con el horizontal para obtener ($I_x I_y$ que es Ixy).

Los elementos calculados por el subsistema de Matriz de estructuras son filtrados mediante la acción de un filtro Gaussiano Hg por medio del bloque "2-D Convolution". Considerando lo anterior el programa en Simulink necesita la definición en línea de comandos de la matriz de coeficientes Hg antes de correr por primera vez el programa tal que:

```
>>Hg=[0 1 2 1 0; 1 3 5 3 1;2 5 9 5 2;1 3 5 3 1;0 1 2 1 0]*(1/57);
```

Estos valores que ahora corresponderán a los definidos por la matriz E expresada en 7.6, serán tomados por el subsistema **Cálculo de V(x,y)** para calcular el valor de la matriz $V(x, y)$ según la ecuación 7.12. Es importante tomar en cuenta que el bloque de ganancia que se encuentra a 0.04 corresponde al valor de α por lo que el algoritmo se encuentra configurado a su máxima sensibilidad, de tal manera que si se eleva este valor la sensibilidad será menor.

Hasta este momento se tienen ya los valores de la matriz $V(x, y)$, que como se sabe representan en base a su magnitud la posibilidad de la existencia de un punto esquina en la imagen (véase figura 7.2). Considerando lo anterior y con el objetivo de implementar el paso 3 del algoritmo 7.1, así como también para preparar los datos y disponerlos en el formato adecuado para su despliegue se elaboro una función en MatLAb la cual es operada por el bloque de Simulink "MATLAB Fcn". La función como puede verse en el programa 7.2 es una versión parcial de el programa 7.1. La función recibe como entrada la matriz correspondiente a $V(x, y)$ calculada por el subsistema **Cálculo de V(x,y)** y entrega como resultado una matriz de dimensión 2x140, en donde 140 representa al número de esquinas que es capaz de encontrar como significativos en la imagen, el número fué calculado a partir de pruebas realizadas en el sistema terminado y evidentemente puede ser cambiado si se

requiere un mayor número de puntos. El programa podría ser modificado con un poco de esfuerzo adicional para soportar una dimensión dinámica en los datos sin embargo por motivos didácticos del libro se prefirió mantener la sencillez en la realización de los programas.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Función usada en el programa de la figura 7.7 de
Simulink
% para calcular el paso 3 del algoritmo 7.1 y disponer
los
% datos que representan a las esquinas como los requiere
% el bloque "Draw Markers" para su despliegue
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Erik Cuevas, Daniel Zaldivar, Marco Pérez
% Versión: 14/01/08
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function Puntos = paso3(V)
%Se genera U a través de la aplicación del umbral a
V(x,y)
U=V>4000;
%Se define el tamaño de los puntos esquinas que
%serán devueltos para la graficación (#140)
re1=ones(1,140);
co1=ones(1,140);
%Se obtiene el tamaño de la imagen V(x,y)
[m n]=size(V);
%Se inicializa con ceros a S(x,y)
S=zeros(size(V));
%Artificialmente un píxel se pone en uno para
%asegurar que la función devuelva algo, este será el
%punto de la esquina superior de la izquierda de la
%imagen
S(10,10)=1;
%Se fija el valor de los limites de la región de
%vecindad sobre la cual se encuentra el valor máximo
pixel=10;
%Se genera el barrido de la imagen píxel a píxel
%encontrando el valor máximo en la vecindad definida
%alrededor del píxel en cuestión
for r=1:m
    for c=1:n
        if(U(r,c))
            I1=[r-pixel 1];
            I2=[r+pixel m];
            I3=[c-pixel 1];
            I4=[c+pixel n];
            datxi=max(I1);
            datxs=min(I2);
            datyi=max(I3);
            datys=min(I4);
            Bloc=A(datxi:1:datxs,datyi:1:datys);
            MaxB=max(max(Bloc));
            if(V(r,c)==MaxB)

                %Matriz binaria con unos en las
%posiciones de las esquinas
                S(r,c)=1;
                end
        end
    end
end
```

```
%Se averiguan los valores en renglones y columnas
%de los puntos que no son cero de la matriz S, que
%representarían los puntos esquinas
[re co]=find(S);

%Los datos se empaquetan en un vector 2x140 que es el
%formato requerido por el bloque "Draw Markers" para
%el despliegue de los puntos.
re1(1,1:l)=re;
co1(1,1:l)=co;
Puntos=[re1;co1];
```

Programa 7.2 Función realizada para obtener las esquinas significativas a partir de $V(x,y)$ encontrando el máximo dentro de un bloque considerado como una vecindad relativa al píxel considerado. La función es operada por el bloque "MATLAB Fcn" dentro del programa de Simulink.

La función encuentra el valor máximo de $V(x,y)$ considerado en una vecindad relativa a un píxel, el cual a través de un proceso de barrido se recorre en toda la imagen. El resultado de los valores máximos de $V(x,y)$ correspondientes a un único bloque se encuentra señalizado en la matriz binaria $S(x,y)$, los elementos de esta matriz que se encuentren en uno representaran a los píxeles en la imagen que son esquinas, mientras que los puntos que sean cero, no formularán en la localización.

Obtenidas las posiciones de las esquinas en $S(x,y)$, será necesario desplegarlas en la imagen original, para ello se utiliza el bloque "Draw Markers" el cual como ya ha sido visto en capítulos anteriores permite remarcar puntos en una imagen mediante la impresión de símbolos, en el caso de este ejemplo se selecciona en la mascarilla de configuración del bloque el símbolo +. No obstante, para graficar, el bloque necesita le sean entregados los puntos en un formato de $2xN_p$, donde N_p es el número de puntos.

Los valores correspondientes a los puntos esquinas son obtenidos mediante la aplicación de la función de MatLAB `find` cuya sintaxis puede ser definida como:

```
[reng col]=find(Mat);
```

Donde `reng` y `col` son vectores que reciben las coordenadas de los puntos que son diferentes de cero de la matriz `Mat`, donde `reng` especifica el renglón y `col` la columna. Obtenidos estos datos se empaquetan en una sola estructura y es enviada como respuesta de la función al siguiente bloque "Draw Markers".

Siempre que se utiliza el bloque de la función de usuario, "MATLAB Fcn" en Simulink, y esta no envía como respuesta una matriz con las mismas dimensiones que la de la matriz recibida como entrada, habrá que especificar la dimensión de la matriz de respuesta o de salida, que en el caso de este ejemplo como fué mencionado será de 2x140 (1). La figura 7.9 ilustra este proceso de configuración en la mascarilla de este bloque.

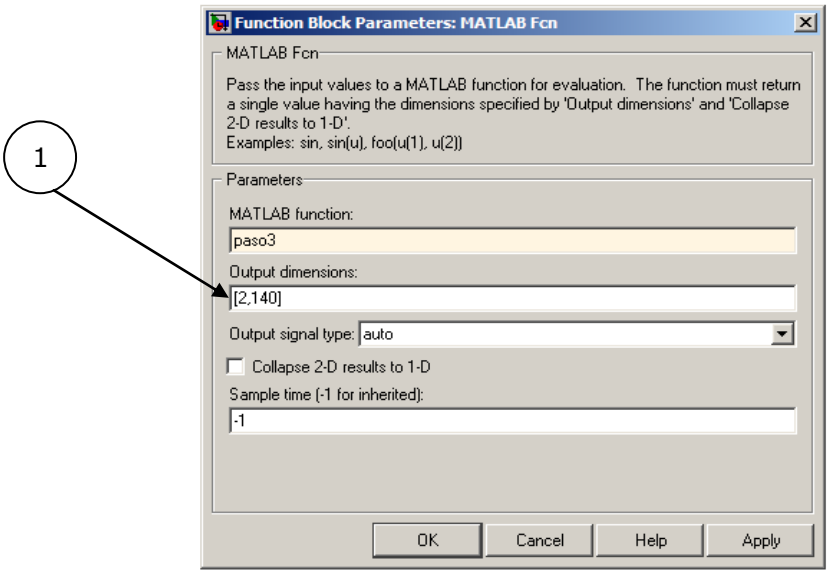


Figura 7.9 Configuración de la mascarilla del bloque MATLAB Fcn, para permitir el regreso de datos diferentes a los de la configuración de entrada.

Como parte final de esta implementación es importante notar que de acuerdo a la implementación aquí desarrollada si el número de esquinas es menor a 140 (lo cual normalmente sucede) aparecerán múltiples esquinas en el punto (1,1) por lo cual no se notaran en el resultado final. El peor caso resulta cuando el número de esquinas sea mayor a 140, lo cual provocaría un error en el programa. Para evitar esta consideración lo mejor será colocar en las dimensiones de los vectores `re1` y `co1`, un valor superior a 140 quizás 500, en el caso de esta implementación se hicieron pruebas y el numero máximo de esquinas posibles de acuerdo a la vecindad configurada no excedían los 120. La figura 7.10 muestra la imagen obtenida después de ejecutar el programa en Simulink representado en la figura 7.7.

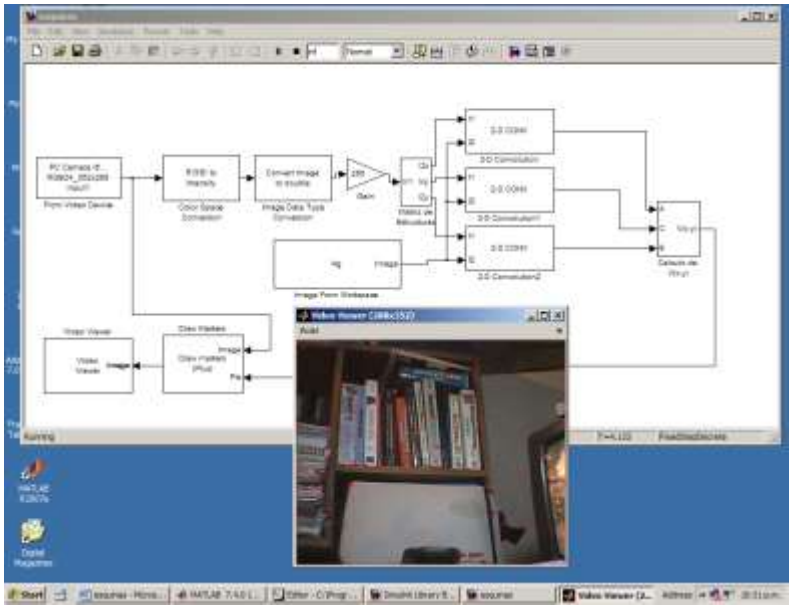


Figura 7.10 Resultado de ejecutar el programa en Simulink mostrado en la figura 7.7.

1.5 ALGUNOS OTROS DETECTORES DE ESQUINAS

En esta sección se analizarán algunos otros detectores existentes que permiten encontrar esquinas de una imagen. Entre los enfoques analizados se encuentran: el operador Beaudet, el Kitchen & Rosenfeld y el Wang & Brady.

Todos los operadores aquí tratados se basan en el cálculo de alguna operación desarrollada con la matriz Hessiana o el Hessiano.

La matriz hessiana de una función f de n variables, es la matriz cuadrada de n×n, de las segundas derivadas parciales. Por lo que dada una función real f de 2 variables reales:

$$f = (x, y) \tag{7.18}$$

Si todas las segundas derivadas parciales de f existen, se define la **matriz Hessiana** de f como

$$\mathrm{H}_f(x, y) \tag{7.19}$$

donde

$$\mathrm{H}_f(x, y) = \begin{bmatrix} \dfrac{\partial^2 f(x, y)}{\partial x^2} & \dfrac{\partial^2 f(x, y)}{\partial x \partial y} \\ \dfrac{\partial^2 f(x, y)}{\partial x \partial y} & \dfrac{\partial^2 f(x, y)}{\partial y^2} \end{bmatrix} \tag{7.20}$$

Normalmente son definidas las componentes del Hessiano como:

$$\mathrm{I}_{xx} = \frac{\partial^2 f(x, y)}{\partial x^2} \tag{7.21}$$

$$\mathrm{I}_{yy} = \frac{\partial^2 f(x, y)}{\partial y^2} \tag{7.22}$$

$$\mathrm{I}_{xy} = \frac{\partial^2 f(x, y)}{\partial x \partial y} \tag{7.23}$$

1.5.1 Detector Beaudet

El detector Beaudet es un operador isotrópico basado en el cálculo del determinante del Hessiano. Por lo que si consideramos el Hessiano como el definido en la ecuación 7.20 su determinante sería:

$$\det(\mathrm{H}_f(x,y)) = \mathrm{I}_{xx}\mathrm{I}_{yy} - \mathrm{I}_{xy}^2 \tag{7.24}$$

Basado en el cálculo del determinante se obtiene la matriz definida como:

$$B(x,y) = \frac{\det(\mathrm{H}_f(x,y))}{(1+\mathrm{I}_x^2+\mathrm{I}_y^2)^2} \tag{7.25}$$

Donde se definen:

$$\mathrm{I}_x = \frac{\partial f(x,y)}{\partial x} \quad \text{y} \quad \mathrm{I}_y = \frac{\partial f(x,y)}{\partial y} \tag{7.26}$$

Como los gradientes en el sentido horizontal y vertical. De tal forma que bajo este operador serán considerados esquinas aquellos puntos de $B(x,y)$ que sobrepasen o sean iguales a un determinado umbral prefijado. El programa 7-3 muestra el programa realizado en MatLAB que implementa completamente el detector Beaudet.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Implementación del algoritmo de Beaudet
% para la detección de esquinas en una imagen
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Erik Cuevas, Daniel Zaldivar, Marco Pérez
% Versión: 11/01/08
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Se carga la imagen a detectar las esquinas.
Iorig = imread('taza.bmp');
%Se convierte a tipo de dato double para evitar
%problemas de calculo
Im = double(rgb2gray(Iorig));
%Se define la matriz del prefiltro
h=ones(3)/9;
%Se filtra la imagen
Im = imfilter(Im,h);
%Se definen los filtros Sobel
sx=[-1,0,1;-2,0,2;-1,0,1];
sy=[-1,-2,-1;0,0,0;1,2,1];
%Se obtiene la primera derivada parcial
Ix = imfilter(Im,sx);
Iy = imfilter(Im,sy);

%Se obtiene la segunda derivada parcial
Ixx = imfilter(Ix,sx);
Iyy = imfilter(Iy,sy);
Ixy = imfilter(Ix,sy);
%Se calcula el denominador de 7.25
B = (1 +Ix.* Ix + Iy.* Iy) .^2;
%Se obtiene el determinante definido en 7.24
A = Ixx.*Iyy - (Ixy).^2;
%Se calcula el valor de B(x,y) de 7.25
B = (A./B);
%Se escala la imagen
B=( 1000/max(max(B)))*B;
%Se binariza la imagen
```

```
V1= (B)>10;
%Se define la vecindad de búsqueda
pixel = 10;
%Se obtiene el valor mas grande de B, de una vecindad
%definida por la variable píxel
[n,m] = size(V1);
res = zeros(n,m);
for r=1:n
    for c=1:m
        if (V1(r,c))
            I1=[r-pixel,1];
            I1 =max(I1);
            I2=[r+pixel,n];
            I2=min(I2);
            I3=[c-pixel,1];
            I3 =max(I3);
            I4=[c+pixel,m];
            I4=min(I4);

            tmp = B(I1:I2,I3:I4);
            maxim = max(max(tmp));
            if(maxim == B(r,c))
              res(r,c) = 1;
            end
         end
    end
end
%Se grafican sobre la imagen Iorig las esquinas
%calculadas por el algoritmo de Beaudet en las
%posiciones donde existen unos en la matriz res
imshow(uint8(Iorig));
hold on
[re,co] = find(res');
plot(re,co,'+');
```

Programa 7.3 Implementación en MatLAB del algoritmo de Beaudet.

La figura 7.11 muestra las esquinas detectadas en una imagen ejemplo utilizando el algoritmo de Beaudet.

Figura 7.11 Localización de los puntos esquinas según el algoritmo de Beaudet.

1.5.2 Operador Kitchen & Rosenfeld

Kitchen & Rosenfeld propusieron un detector de esquinas basado en el cambio de la dirección del gradiente a lo largo de un borde multiplicado por la dirección del gradiente local en el píxel en cuestión. Por lo que Kitchen y Rosenfeld propusieron calcular la matriz:

$$KR(x,y) = \frac{I_{xx} \cdot I_y^2 + I_{yy} \cdot I_x^2 - 2 \cdot I_{xy} \cdot I_y}{I_x^2 + I_y^2} \tag{7.27}$$

En este enfoque se considera a las esquinas como aquellos valores de que sobrepasen un valor prefijado considerado como umbral. El programa 7.4 muestra el programa realizado en MatLAB que implementa completamente el operador de Kitchen & Rosenfeld.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Implementación del algoritmo de Kitchen y Rosenfeld
% para la detección de esquinas en una imagen
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Erik Cuevas, Daniel Zaldivar, Marco Pérez
% Versión: 11/01/08
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Se carga la imagen a detectar las esquinas.
Iorig = imread('taza.bmp');
%Se convierte a tipo de dato double para evitar
%problemas de calculo
Im = double(rgb2gray(Iorig));
%Se define la matriz del prefiltro
h=ones(3)/9;
%Se filtra la imagen
Im = imfilter(Im,h);
%Se definen los filtros Sobel
sx=[-1,0,1;-2,0,2;-1,0,1];
sy=[-1,-2,-1;0,0,0;1,2,1];
%Se obtiene la primera derivada parcial
Ix = imfilter(Im,sx);
Iy = imfilter(Im,sy);
%Se obtiene la segunda derivada parcial
Ixx = imfilter(Ix,sx);
Iyy = imfilter(Iy,sy);
Ixy = imfilter(Ix,sy);
%Se calcula el numerador de la ecuación 7.27
A = (Ixx.*(Iy.^2)) + (Iyy.*(Ix.^2))- (2*Ixy.*Iy);
%Se calcula el denominador de la ecuación 7.27
B = (Ix.^2) + (Iy.^2);
%Se calcula  la ecuación 7.27
V = (A./B);
%Se escala la imagen
V=( 1000/max(max(V)))*V;
%Se binariza la imagen
V1= (V)>100;
%Se define la vecindad de búsqueda
pixel = 10;
%Se obtiene el valor mas grande de V, de una vecindad
%definida por la variable píxel
[n,m] = size(V1);
res = zeros(n,m);
for r=1:n    %rows
    for c=1:m  %cols
        if (V1(r,c))
```

```
                I1=[r-pixel,1];
                I1 =max(I1);
                I2=[r+pixel,n];
                I2=min(I2);
                I3=[c-pixel,1];
                I3 =max(I3);
                I4=[c+pixel,m];
                I4=min(I4);

                tmp = V(I1:I2,I3:I4);
                maxim = max(max(tmp));
                if(maxim == V(r,c))
                  res(r,c) = 1;
                end
            end
        end
    end
    %Se grafican sobre la imagen Iorig las esquinas
    %calculadas por el algoritmo de Kitchen y Rosenfels %en
    las posiciones donde existen unos en la matriz %res
    imshow(uint8(Iorig));
    hold on
    [re,co] = find(res');
    plot(re,co,'+');
```

Programa 7.4 Implementación en MatLAB del algoritmo de Kitchen & Rosenfeld.

La figura 7.12 muestra las esquinas detectadas en una imagen ejemplo utilizando el algoritmo de Kitchen & Rosenfeld.

Figura 7.12 Localización de los puntos esquinas según el algoritmo de Kitchen & Rosenfeld.

1.5.3 Detector de Wang & Brady.

El operador de Wang & Brady para la detección de esquinas considera a una imagen como una superficie, de tal forma que el algoritmo busca lugares en la imagen donde la dirección de un borde cambia abruptamente, como es el caso en las esquinas. Para ello se mide un coeficiente $C(x,y)$ que se establece para cada píxel definido como:

$$C(x,y) = \nabla^2 I(x,y) + c\left|\nabla I(x,y)\right| \tag{7.28}$$

Donde c representa un parámetro que calibra la sensibilidad del algoritmo. Mientras que $\nabla^2 I(x,y)$ y $\nabla I(x,y)$ se definen como:

$$\nabla^2 I(x,y) = \frac{\partial^2 I(x,y)}{\partial x^2} + \frac{\partial^2 I(x,y)}{\partial y^2} \quad \text{y} \quad \nabla I(x,y) = \frac{\partial I(x,y)}{\partial x} + \frac{\partial I(x,y)}{\partial y} \qquad (7.29)$$

Por lo que para elegir si un píxel es esquina o no solamente se considera a partir del cálculo de $C(x,y)$ la aplicación de un umbral y la elección de aquellos puntos que sean superiores a este.

Utilizando la aproximación de la derivada definida en la ecuación 7.2 mencionada en el capítulo 6, se puede establecer que:

$$\begin{aligned} \frac{\partial I(x,y)}{\partial x} &= -0.5I(x-1,y) + 0.5(x+1,y) \\ \frac{\partial I(x,y)}{\partial y} &= -0.5I(x,y-1) + 0.5(x,y+1) \end{aligned} \qquad (7.30)$$

De esta manera se redefine el operador $\nabla I(x,y)$, tal que:

$$\begin{aligned} \nabla I(x,y) &= -0.5I(x-1,y) + 0.5I(x+1,y) - 0.5I(x,y-1) + 0.5I(x,y+1) \\ \nabla I(x,y) &= \begin{bmatrix} 0 & -0.5 & 0 \\ -0.5 & 0 & 0.5 \\ 0 & 0.5 & 0 \end{bmatrix} \end{aligned} \qquad (7.31)$$

De igual manera se establece en las ecuaciones 6.29 y 6.30 del capítulo 6 que:

$$\begin{aligned} \frac{\partial^2 I(x,y)}{\partial x^2} &= I(x+1,y) - 2I(x,y) + I(x-1,y) \\ \frac{\partial^2 I(x,y)}{\partial y^2} &= I(x,y+1) - 2I(x,y) + I(x,y-1) \end{aligned} \qquad (7.32)$$

Por lo que:

$$\begin{aligned} \nabla^2 I(x,y) &= I(x+1,y) + I(x-1,y) + I(x,y+1) + I(x,y-1) - 4I(x,y) \\ \nabla^2 I(x,y) &= \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \end{aligned} \qquad (7.33)$$

El programa 7.5 muestra el programa realizado en MatLAB que implementa completamente el operador de Wang & Brady.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Implementación del algoritmo de Wang y Brady
% para la detección de esquinas en una imagen
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Erik Cuevas, Daniel Zaldivar, Marco Pérez
% Versión: 11/01/08
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Se carga la imagen a detectar las esquinas.
Iorig = imread('taza.bmp');
%Se convierte a tipo de dato double para evitar
%problemas de calculo
Im = double(rgb2gray(Iorig));
%Se define la matriz del prefiltro
h=ones(3)/9;
%Se filtra la imagen
Im = imfilter(Im,h);
%Se define el filtro descrito en la ecuación 7.31
d1=[0,-0.5,0;-0.5,0,0.5;0,0.5,0];
%Se define el filtro descrito en la ecuación 7.33
d2=[0,1,0;1,-4,1;0,1,0];
%Se calculan las expresiones 7.31 y 7.33
I1 = imfilter(Im,d1);
I2 = imfilter(Im,d2);
%se define el parámetro de sensibilidad a 1
c= 1;
%Se calcula el operador de Wang y Brady Ec. 7.28
V = (I2 - c*abs(I1));
%Se escala la imagen
V=( 1000/max(max(V)))*V;
%Se binariza la imagen
V1= (V)>100;
%Se define la vecindad de búsqueda
pixel = 10;
%Se obtiene el valor mas grande de V, de una vecindad
%definida por la variable píxel
[n,m] = size(V1);
res = zeros(n,m);
for r=1:n
    for c=1:m
        if (V1(r,c))
            I1=[r-pixel,1];
            I1 =max(I1);
            I2=[r+pixel,n];
            I2=min(I2);
            I3=[c-pixel,1];
            I3 =max(I3);
            I4=[c+pixel,m];
            I4=min(I4);

            tmp = V(I1:I2,I3:I4);
            maxim = max(max(tmp));
            if(maxim == V(r,c))
              res(r,c) = 1;
            end
         end
    end
end
%Se grafican sobre la imagen Iorig las esquinas
%calculadas por el algoritmo de Wang y Brady en las
%posiciones donde existen unos en la matriz res
imshow(uint8(Iorig));
```

```
        hold on
        [re,co] = find(res');
        plot(re,co,'+');
```

Programa 7.5 Implementación en MatLAB del algoritmo de Wang & Brady.

La figura 7.13 muestra las esquinas detectadas en una imagen ejemplo utilizando el algoritmo de Wang & Brady.

Figura 7.13 Localización de los puntos esquinas según el algoritmo de Wang & Brady.

Con el objetivo de mostrar una visión comparativa de los operadores tratados en esta sección, la figura 7.14 muestra la aplicación de los diferentes algoritmos: Harris, Beaudet, Kitchen & Rosenfeld, y el Wang & Brady.

Figura 7.14 Comparación de los algoritmos para la detección de esquinas: Rojo ○ Harris, + Amarillo Beaudet, Cyan ▽ Kitchen & Rosenfeld y Verde Δ Wang & Brady.