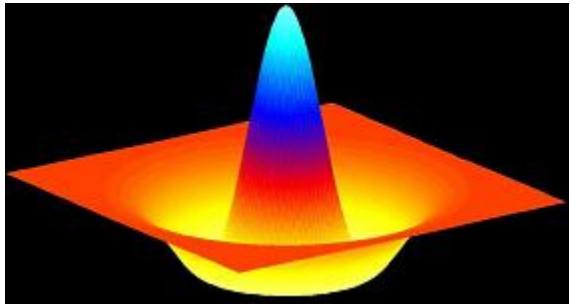


Procesamiento Digital de Imágenes

Detección de Bordes y Segmentación

Segmentación de Imágenes

Los métodos de segmentación de imágenes se caracterizan por tener como entrada una imagen y como salida atributos extraídos de esa imagen.

Estos métodos subdividen la imagen de acuerdo a las regiones y/u objetos que la componen. El nivel de esta subdivisión dependerá del problema que se está analizando.

En muchas aplicaciones, una correcta segmentación determina el éxito o el fracaso del procedimiento automático computarizado (falsas detecciones pueden llevar a incorrectas acciones).

Segmentación de Imágenes

Los algoritmos de segmentación para imágenes monocromáticas se basan generalmente en una de las dos propiedades básicas de los valores de intensidad: la discontinuidad y la similitud.

En los métodos del primer grupo la partición de la imagen se realiza a partir de cambios abruptos de intensidad, como son los bordes. Una herramienta útil para la detección de segmentos lineales de borde es la transformada de Hough. Otra herramienta que utilizan estos métodos es el Umbralado (Thresholding), que en particular tiene gran uso en aplicaciones que requieren velocidad.

En los métodos del segundo grupo la partición de la imagen se realiza a partir de regiones que son similares de acuerdo a un criterio predefinido. Estos métodos pueden utilizar herramientas de procesamiento morfológico.

Detección de Punto

La forma más común para buscar discontinuidades es utilizar una máscara en toda la imagen como vimos anteriormente. Para una máscara de 3x3 se calcula la suma de los productos de los coeficientes y la intensidad de los pixels contenidos en la región delimitada por la máscara. Así,

$$R = w_1 z_1 + w_2 z_2 + \ldots + w_9 z_9 = \sum_{i=1}^{9} w_i z_i$$

donde z_i es la intensidad del pixel asociado al coeficiente w_i de la máscara.

Detección de Punto

La detección de puntos contenidos en un área de intensidad relativamente constante es sencilla. Usando la máscara mostrada podemos detectar un punto cuando al centrar la máscara en el pixel se cumple:

−1	−1	−1
−1	8	−1
−1	−1	−1

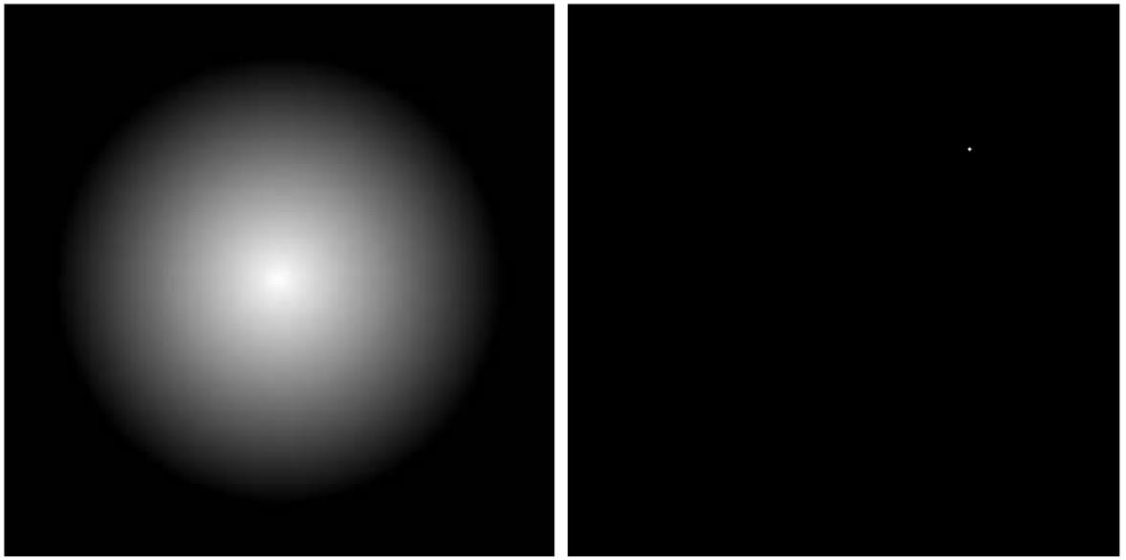
$$|R| \geq T$$

siendo T un umbral no negativo.

Detección de Punto

Con MATLAB podemos usar **imfilter** con la máscara anterior:

```
>> g = abs(imfilter(double(f),w)) >= T
```

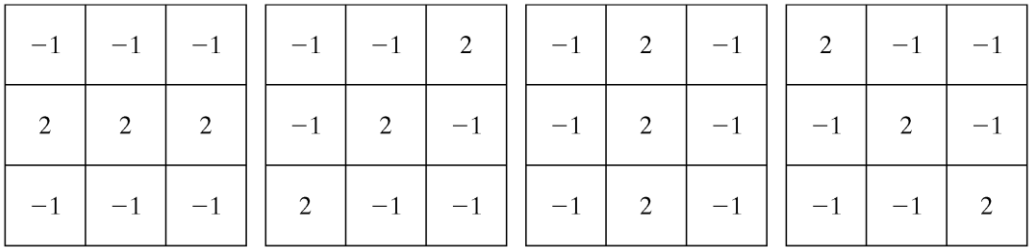


a b

FIGURE 10.2
(a) Gray-scale image with a nearly invisible isolated black point in the dark gray area of the northeast quadrant. (b) Image showing the detected point. (The point was enlarged to make it easier to see.)

Detección de Línea

Consideremos las siguientes máscaras:

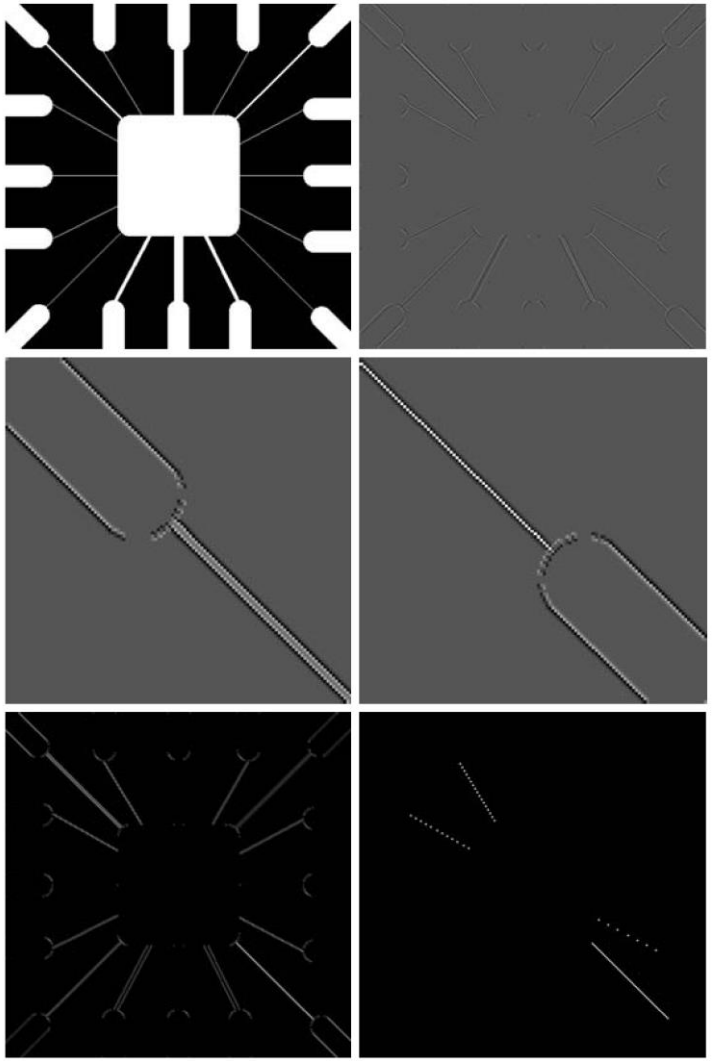


Si la primera máscara se mueve por una imagen, responderá con mayor fuerza a líneas (de un pixel de grosor) orientadas horizontalmente. Con un fondo constante la respuesta máxima se obtendrá cuando la línea quede en la fila central de la máscara.

Nótese que la dirección preferida de cada máscara se pesa con coeficientes de mayor valor (2) que las restantes. La suma de los coeficientes de la máscara es cero de manera que en las zonas de fondo uniforme (misma intensidad para todos los pixels) R = 0.

Detección de Línea

```
>> w = [2 -1 -1; -1 2 -1; -1 -1 2];
>> g = imfilter(double(f),w);
>> imshow(g,[])
>> gtop = g(1:120,1:120);
>> figure, imshow(gtop,[])
>> gbot = g(end-119:end,end-119:end);
>> gbot = pixeldup(gbot,4);
>> figure, imshow(gbot,[])
>> g = abs(g);
>> figure,imshow(g,[])
>> T = max(g(:));
>> g = g >= T;
>> figure,imshow(g,[])
```



Detección de Bordes

Para este tipo de detección se utilizan las derivadas de primer orden (gradiente) y segundo orden.

$$\nabla \mathrm{f} = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \frac{df}{dx} \\ \frac{df}{dy} \end{bmatrix} \Rightarrow \nabla f = mag(\nabla \mathrm{f}) = \left[{G_x}^2 + {G_y}^2 \right]^{1/2} = \left[\left(\frac{df}{dx} \right)^2 + \left(\frac{df}{dy} \right)^2 \right]^{1/2} \approx {G_x}^2 + {G_y}^2$$

Esta última aproximación se comporta como la derivada, es cero en áreas de intensidad constante y tienen un valor proporcional a la variación de intensidad en áreas con variaciones en los valores de pixel.

Una propiedad fundamental del gradiente es que apunta en la dirección de máxima tasa de cambio de *f* en las coordenadas *(x,y)*. El ángulo en el que se produce este máximo es:

$$\alpha(x,y) = tan^{-1}\left(\frac{G_x}{G_y}\right)$$

Detección de Bordes

La derivada de segundo orden en 2D se calcula a partir del Laplaciano:

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

El laplaciano no se usa en si mismo para realizar detección de bordes, sino en combinación con técnicas de detección de borde.

Así, la idea básica detrás de la detección de bordes es encontrar los lugares dentro de la imagen donde la intensidad cambia abruptamente utilizando uno de estos 2 criterios:

1. Encontrar los lugares donde la derivada de primer orden de la intensidad es mayor que un umbral especificado.
2. Encontrar los lugares donde la derivada de segundo orden de la intensidad tiene un cruce por cero.

Detección de Bordes

La función **edge** provee varios estimadores de la derivada basados en los criterios anteriores, y para algunos de estos estimadores se puede especificar si la detección será sensible a bordes verticales, horizontales o ambos. La sintaxis genérica es:

```
[g,t] = edge(f ,'method',parameters)
```

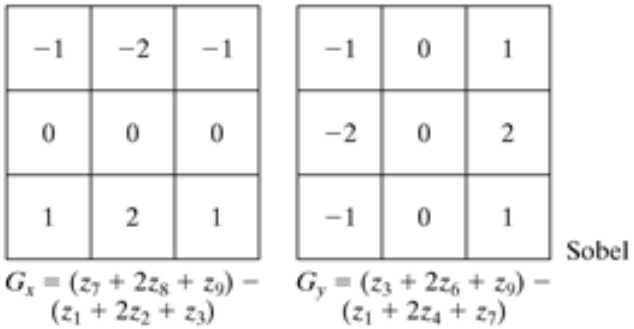
Edge Detector	Basic Properties
Sobel	Finds edges using the Sobel approximation to the derivatives shown in Fig. 10.5(b).
Prewitt	Finds edges using the Prewitt approximation to the derivatives shown in Fig. 10.5(c).
Roberts	Finds edges using the Roberts approximation to the derivatives shown in Fig. 10.5(d).
Laplacian of a Gaussian (LoG)	Finds edges by looking for zero crossings after filtering $f(x, y)$ with a Gaussian filter.
Zero crossings	Finds edges by looking for zero crossings after filtering $f(x, y)$ with a user-specified filter.
Canny	Finds edges by looking for local maxima of the gradient of $f(x, y)$. The gradient is calculated using the derivative of a Gaussian filter. The method uses two thresholds to detect strong and weak edges, and includes the weak edges in the output only if they are connected to strong edges. Therefore, this method is more likely to detect true weak edges.

Detección de Bordes - Sobel

Utiliza la máscara mostrada para aproximar las primeras derivadas G_x y G_y en un pixel a partir de sus vecinos:

$$g = \left[G_x^{\,2} + G_y^{\,2}\right]^{1/2} = \{\ [(z_7 + 2z_8 + z_9) - (z_1 + 2z_2 + z_3)]^2\ + \ [(z_3 + 2z_6 + z_9) - (z_1 + 2z_4 + z_7)]^2\ \}^{1/2}$$

Luego, el pixel *(x,y)* será parte de un borde si el valor de g ≥ *T* , siendo *T* un umbral dado.



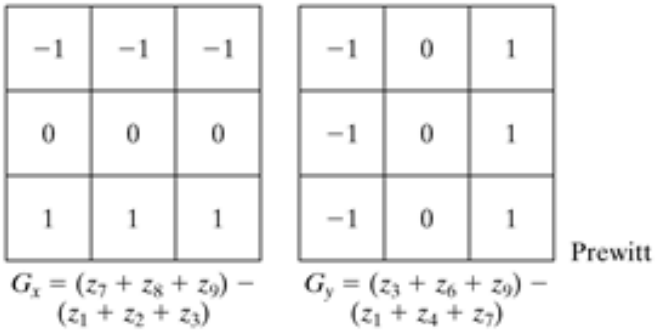
La sintaxis genérica es:

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

siendo **T** un umbral dado y **dir** la dirección preferida de los bordes a detectar ('horizontal', 'vertical' o 'both'). La salida, **g** es una imagen lógica que contiene 1s donde se detectó borde y 0s en los restantes pixels.

Detección de Bordes - Prewitt

Utiliza la máscara mostrada para aproximar las primeras derivadas G_x y G_y en un pixel a partir de sus vecinos.



La sintaxis genérica es:

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

siendo **T** un umbral dado y **dir** la dirección preferida de los bordes a detectar (**'horizontal'**,**'vertical'** o **'both'**). La salida, **g** es una imagen lógica que contiene 1s donde se detectó borde y 0s en los restantes pixels.

Si bien este detector es levemente menos complejo computacionalmente, tiende a producir resultados con mayor ruido.

Detección de Bordes – LoG

Laplaciano de Gaussiano (LoG)

Si consideramos la función Gaussiana:

$$h(r) = -e^{-\frac{r^2}{2\sigma^2}} \qquad con\ r = x^2 + y^2$$

Esta es una función suave que si la aplicamos a la imagen producirá un efecto tipo blur, cuyo grado dependerá del valor de σ. El laplaciano de está función es:

$$\nabla^2 h(r) = -\left[\frac{r^2 - \sigma^2}{\sigma^4}\right] e^{-\frac{r^2}{2\sigma^2}}$$

Como la derivada segunda es una operación lineal, convolucionar (filtrar) una imagen con $\nabla^2 h(r)$ es equivalente a convolucionar la imagen con la función suave $h(r)$ y luego calcular el laplaciano del resultado.

La sintaxis genérica es:

```
[g,t] = edge(f, 'log', T, sigma)
```

Detección de Bordes – Canny

En este método, la imagen se suaviza utilizando un filtro Gaussiano con un valor de sigma específico de manera de reducir el ruido.

El gradiente local (magnitud **g** y dirección **a**) se calcula en todos los pixels:

$$g(x,y) = \left[G_x{}^2 + G_y{}^2\right]^{1/2} \qquad \alpha(x,y) = tan^{-1}\left(\frac{G_x}{G_y}\right)$$

Se define como un punto del borde a todo pixel cuyo valor de **g** es un máximo local en la dirección del gradiente.

Los puntos anteriormente determinados dan lugar a crestas en la imagen de magnitudes del gradiente. El algoritmo se posiciona en estas crestas y hace 0 todos los pixels que no forman parte de la misma, para dar como resultado una línea fina (proceso de supresión de los no máximos).

Los pixels de la cresta se umbralan usando dos valores, T1 y T2. Los pixels mayores que T2 son considerados bordes muy probables y los que son menos a T1 son considerados bordes poco probables.

Detección de Bordes – Canny

Por último se unen los bordes incorporando los bordes poco probables que están 8-conectados a los bordes muy probables.

La sintaxis genérica es:

```
[g,t] = edge(f, 'canny', [T1 T2], sigma)
```

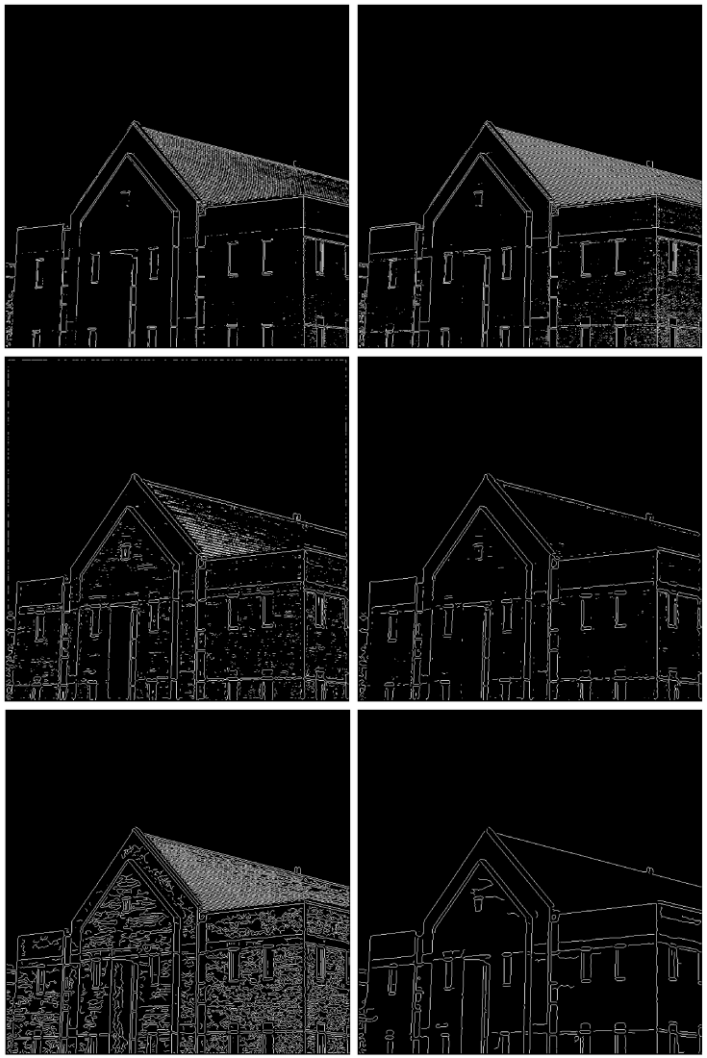
A continuación algunos ejemplos de los resultados producidos por los diferentes métodos de detección de bordes para una imagen dada.

Detección de Bordes

a b
c d
e f

FIGURE 10.6
(a) Original image. (b) Result of function `edge` using a vertical Sobel mask with the threshold determined automatically. (c) Result using a specified threshold. (d) Result of determining both vertical and horizontal edges with a specified threshold. (e) Result of computing edges at 45° with `imfilter` using a specified mask and a specified threshold. (f) Result of computing edges at −45° with `imfilter` using a specified mask and a specified threshold.

Detección de Bordes



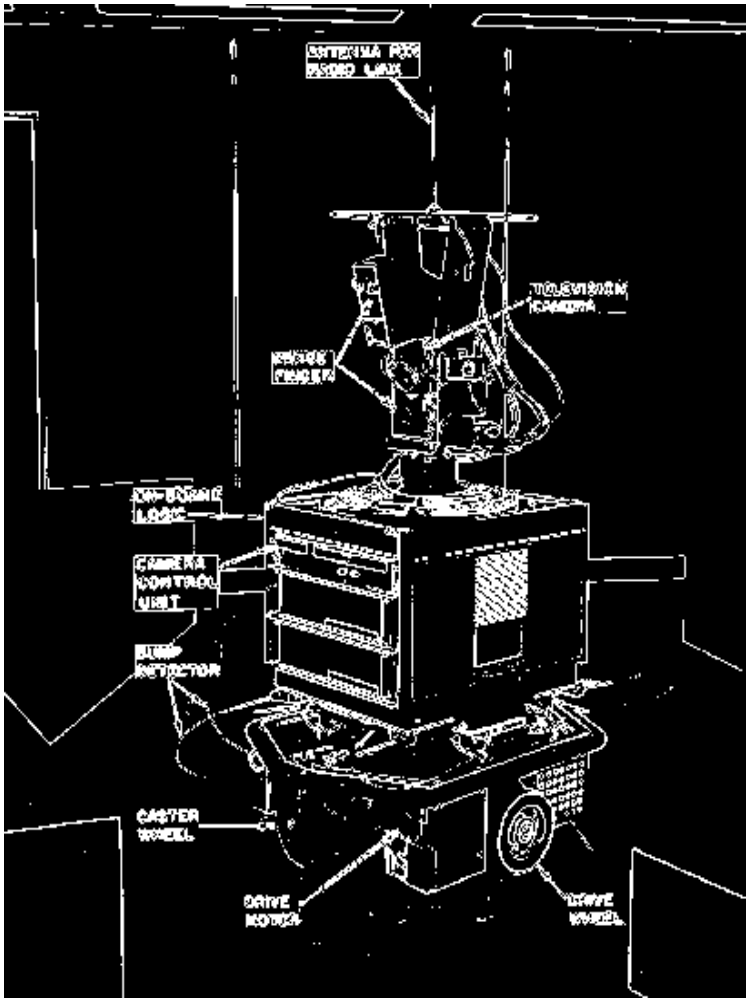
a b
c d
e f

FIGURE 10.7 Left column: Default results for the Sobel, LoG, and Canny edge detectors. Right column: Results obtained interactively to bring out the principal features in the original image of Fig. 10.6(a) while reducing irrelevant, fine detail. The Canny edge detector produced the best results by far.

Detección de Líneas

Luego de aplicar algún método de segmentación tenemos sólo puntos.

En diferentes aplicaciones, se requiere información más compleja, por ejemplo, determinar las líneas en la imagen.



Detección de Líneas Rectas
Transformada Hough

La transformada de Hough es una estrategia para encontrar y unir segmentos de línea en una imagen.

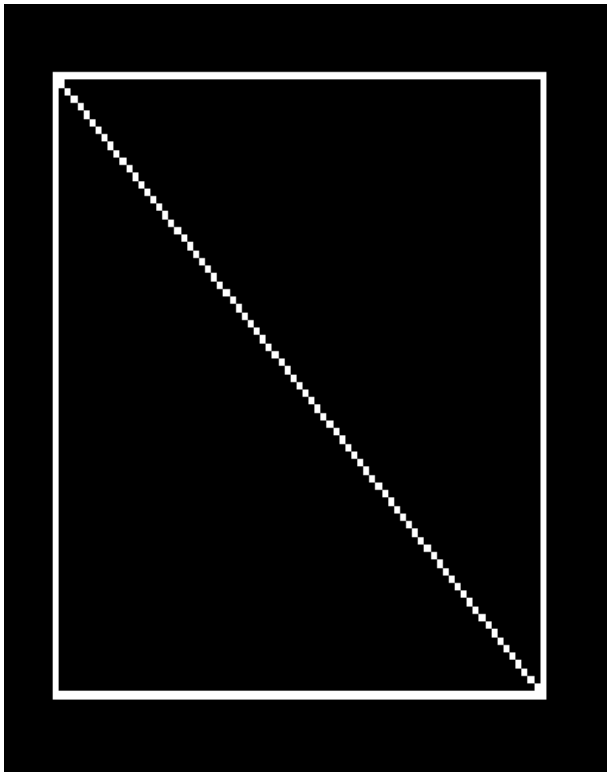
Dado un conjunto de puntos en una imagen binaria queremos encontrar el subconjunto de puntos que se encuentran formando una línea recta.

Una solución posible es encontrar todas las líneas determinadas por cada par de puntos y luego encontrar los subconjuntos de puntos cercanos a líneas particulares. (**Computacionalmente prohibitivo**)

Transformada Hough

Las líneas rectas en la imagen pueden ser parametrizadas por una ecuación.

Cada punto de la imagen, idealmente, podría pertenecer a un número infinito de rectas.



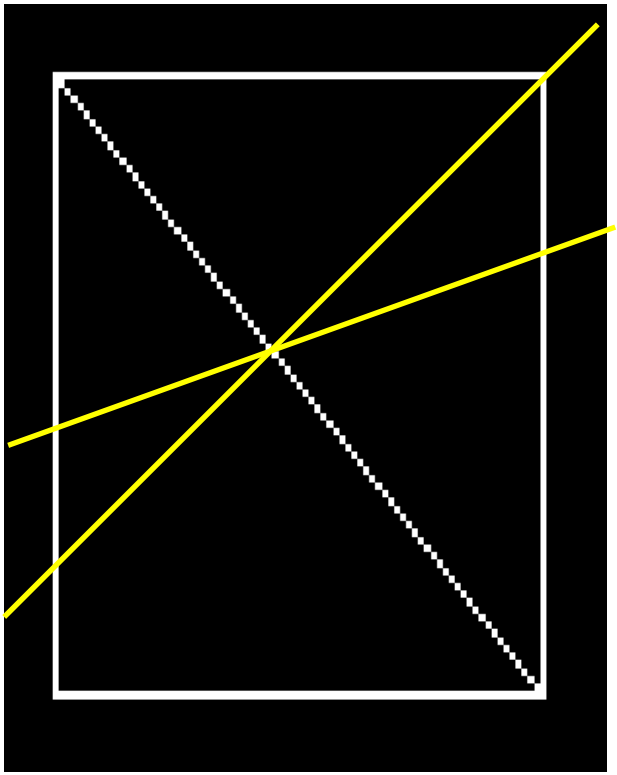
Transformada Hough

Las líneas rectas en la imagen pueden ser parametrizadas por una ecuación.

Cada punto de la imagen, idealmente, podría pertenecer a un número infinito de rectas.

En la transformada de Hough cada punto “vota” para las líneas a las cuales puede pertenecer.

Finalmente, las líneas con más votos ganan.



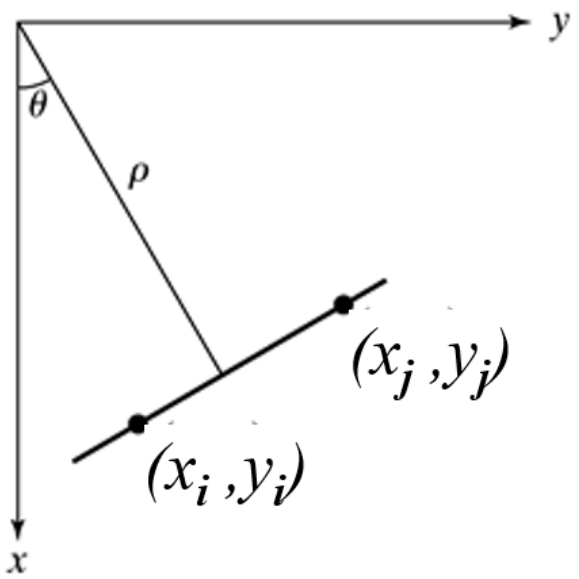
Transformada Hough

Cada recta es representada con 2 parámetros.

$$x \cos \theta + y \operatorname{sen} \theta = \rho$$

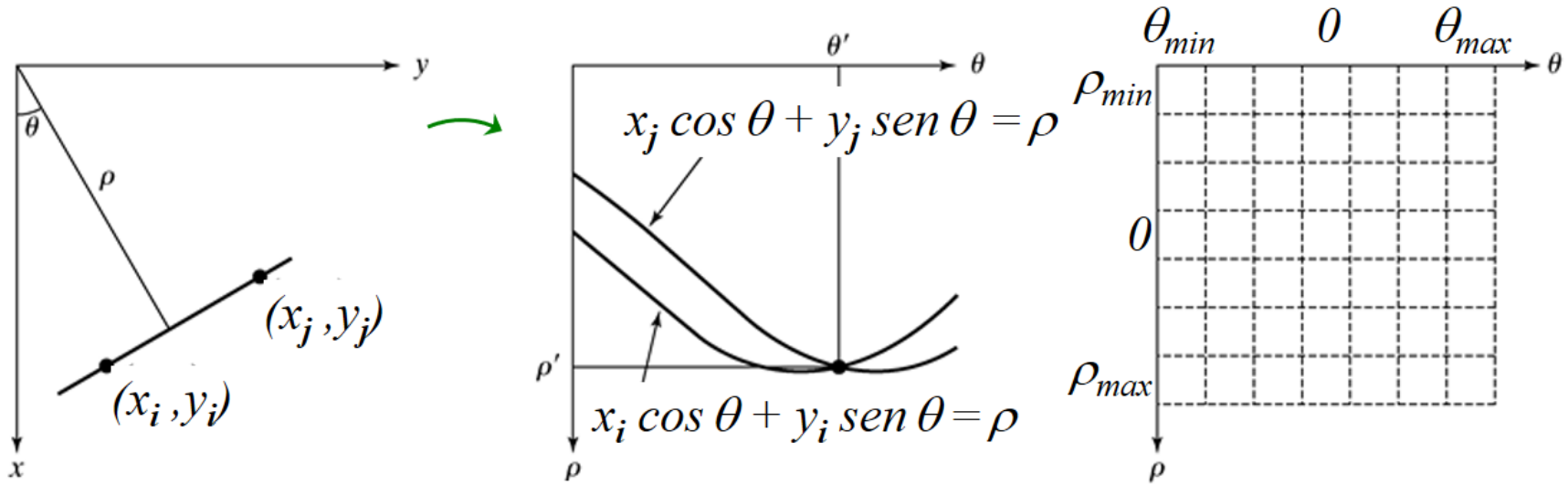
Las rectas se buscan para un los posibles valores de rho y theta.

En este ejemplo, los 2 puntos aportarían a la recta representada.



Transformada Hough

La búsqueda se realiza sobre un conjunto finito de valores para rho y theta. Cada punto aporta las posibles rectas que pasan por él.

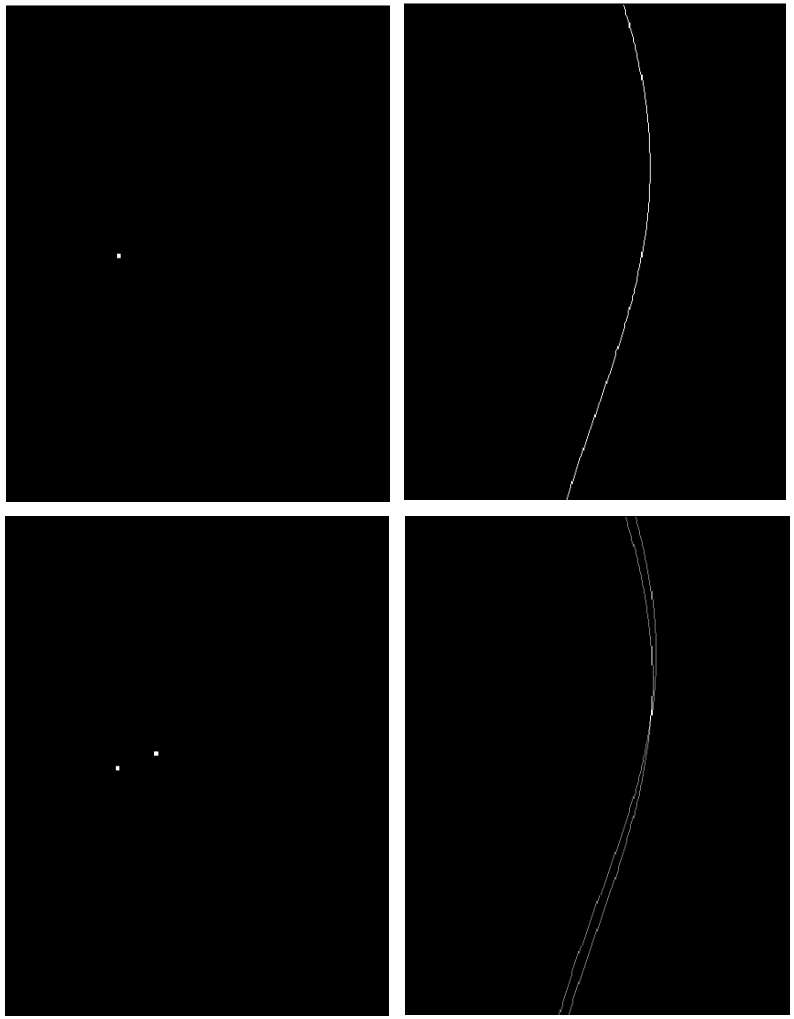


Transformada Hough

Un punto en el espacio de la imagen corresponde a una curva en el espacio de Hough.

Dos puntos corresponden a 2 curvas en el espacio de Hough.

La intersección de estas 2 curvas suma 2 votos.



Transformada Hough

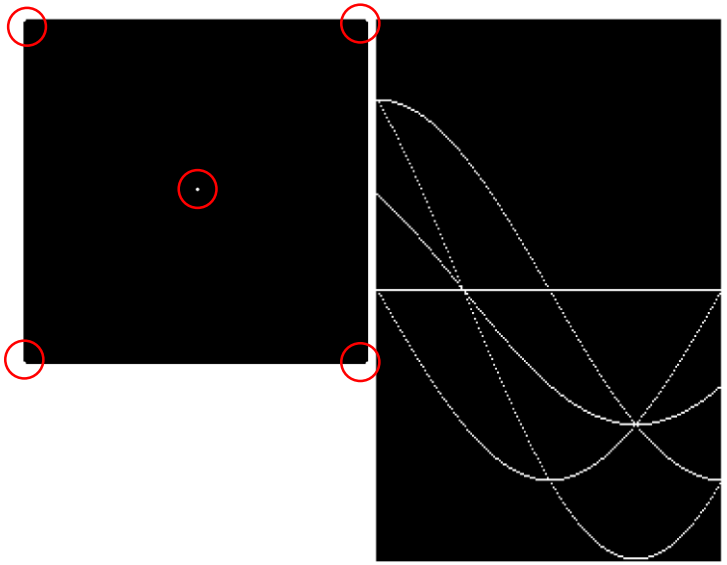
La implementación de la Transformada Hough se encuentra en la función con sintaxis:

```
[ h , theta , rho ] = hough( f , dtheta , drho )
```

siendo **`dtheta`** y **`drho`** los valores equi-espaciados en grados y bins en los que se subdividen los ejes del espacio de parámetros.

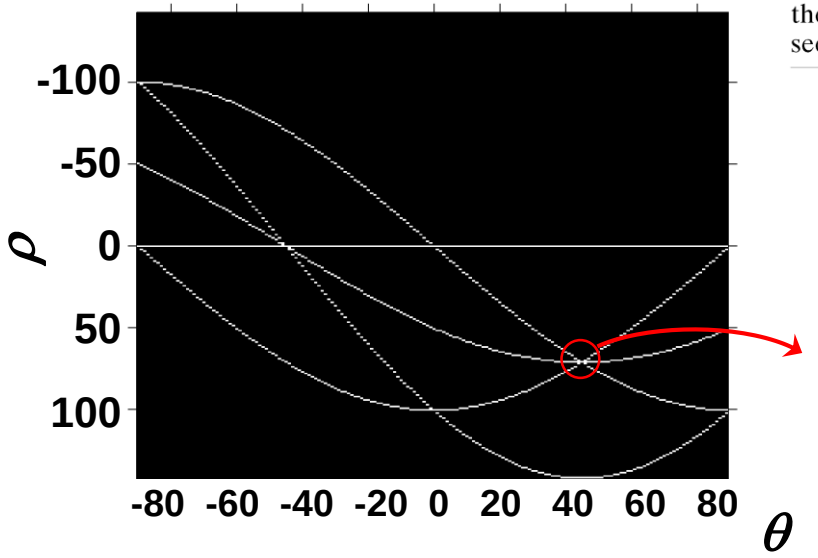
Transformada Hough

Imagen con 5 puntos blancos



a b
c

FIGURE 10.10
(a) Binary image with five dots (four of the dots are in the corners).
(b) Hough transform displayed using `imshow`.
(c) Alternative Hough transform display with axis labeling. (The dots in (a) were enlarged to make them easier to see.)



A 45° se intersectan 3 puntos en la imagen original

Transformada Hough

En general, los bordes de los objetos en una imagen no son rectos, y los picos en el espacio transformado no estarán en una única celda.

Una estrategía para resolver esto puede ser:

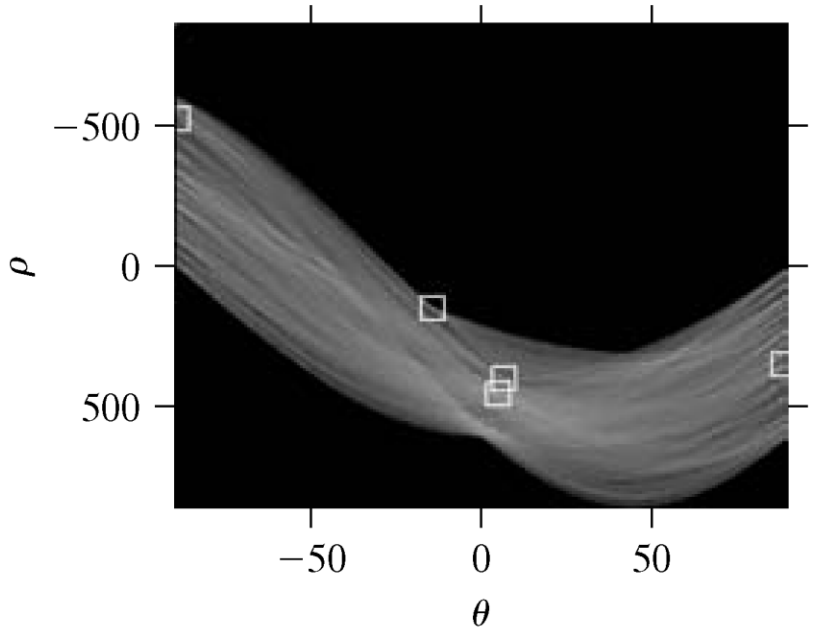
1. Encontrar la celda que tiene el valor máximo y guardar sus coordenadas
2. Llevar a cero las celdas vecinas a la hallada en el punto anterior
3. Repetir hasta que el número de picos deseados haya sido encontrado o hasta que el umbral especificado no sea superado por ninguna celda

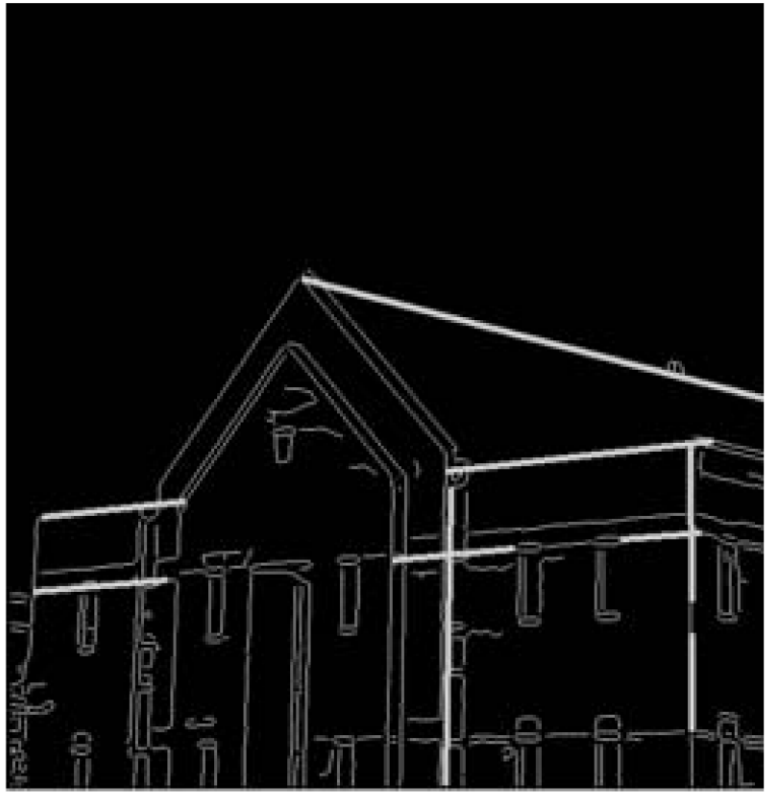
La siguiente función de MATLAB implementa esta estrategia.

```
peaks = houghpeaks(h, numpeaks, threshold, nhood)
```

siendo numpeaks el número de picos buscados y threshold el umbral. El vector nhood tiene 2 valores que especifican el tamaño del vecindario de la celda considerada pico.

Transformada Hough



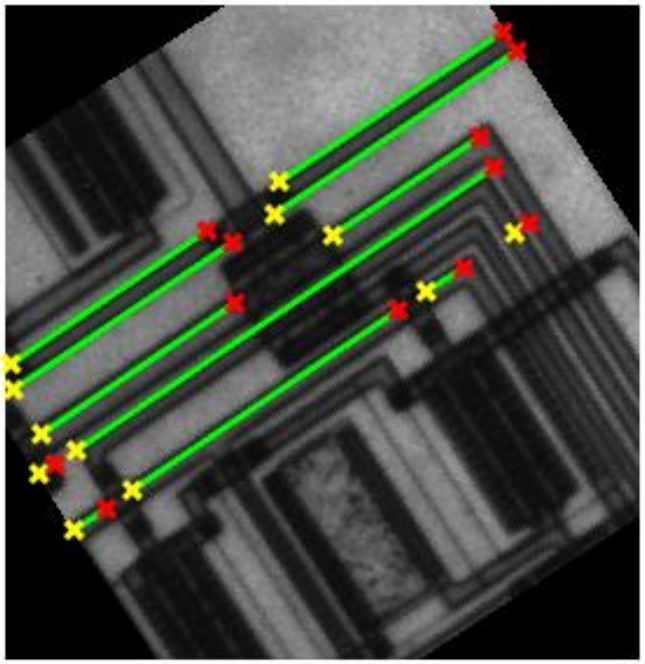


Transformada Hough

Para extraer segmentos de líneas basados en la transformación Hough, Matlab tiene la siguiente función.

```
lines = houghlines(BW, theta, rho, peaks)
```

Cuyos parámetros de entrada son el resultados de hough y houghpeaks, respectivamente.



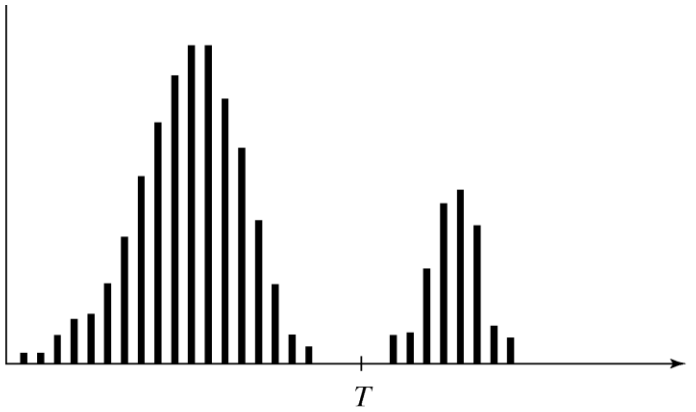
Umbralado Global

Cuando se utiliza un mismo valor de T para aplicar el umbral en toda la imagen, se dice que la operación es global.

$$g(x,y) = \begin{cases} 1 & si\ f(x,y) \geq T \\ 0 & si\ f(x,y) < T \end{cases}$$

De manera ideal, se puede tener un imagen donde la separación entre "fondo" y "objetos" se ve clara en el histograma de la misma.

En este caso, la elección adecuada de T separará de manera perfecta los objetos de interés.



Umbralado Global

Una manera de elegir un umbral automáticamente puede ser:

1. Seleccionar una estima inicial de T (por ej, valor promedio de intensidad de la imagen).

2. Segmentar la imagen con ese valor, con lo cuál obtenemos 2 grupos de pìxels: G1 pixels con valores ≥ T y G2 los pixels con valores < T.

3. Calcular las intensidades promedio, $\mu 1$ y $\mu 2$, de las regiones G1 y G2.

4. Calcular un nuevo valor de umbral: $T = (\mu 1 + \mu 2)/2$

5. Repetir los pasos 2 al 4 hasta que las sucesivas iteraciones produzcan un valor menor a un valor predeterminado To.

Umbralado Global

La función **`graythresh`** calcula automáticamente el umbral utilizando el método de **Otsu**.

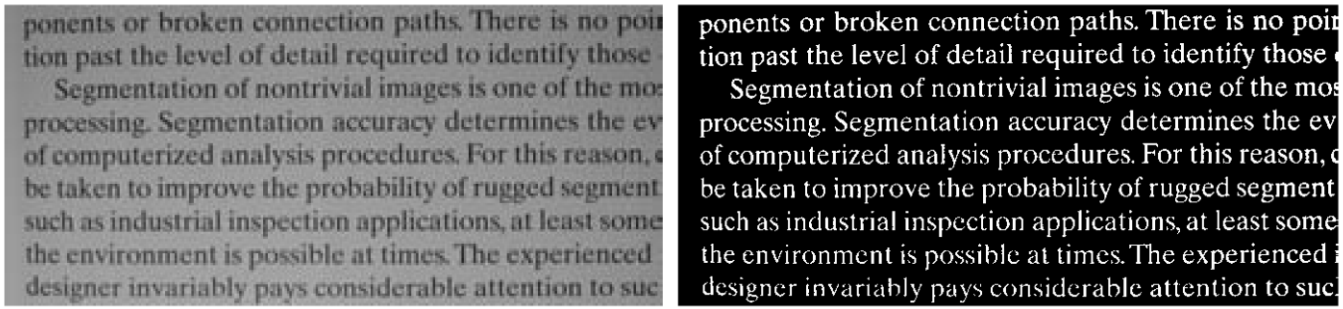
Este método calcula el umbral que minimiza la varianza intraclase de los píxeles blancos y negros resultantes, definida como la suma pesada de las varianzas de las 2 clases:

$$\sigma_B{}^2 = \omega_0(\mu_0 - \mu_T)^2 + \omega_1(\mu_1 - \mu_T)^2$$

donde

$$\omega_0 = \sum_{q=0}^{k-1} p_r(r_q) \qquad \omega_1 = \sum_{q=k}^{L-1} p_r(r_q) \qquad \mu_0 = \sum_{q=0}^{k-1} q\, p_r(r_q)/\omega_0 \qquad \mu_1 = \sum_{q=k}^{L-1} q\, p_r(r_q)/\omega_1$$

Umbralado Global



Opción 1:

```
T = 0.5*( double(min(f(:))) + double(max(f(:))) );
done = false;
while ~done
    g = f >= T;
    Tnext = 0.5*( mean(f(g)) + mean(f(~g)) );
    done = abs( T - Tnext ) < 0.5;
    T = Tnext;
End
```

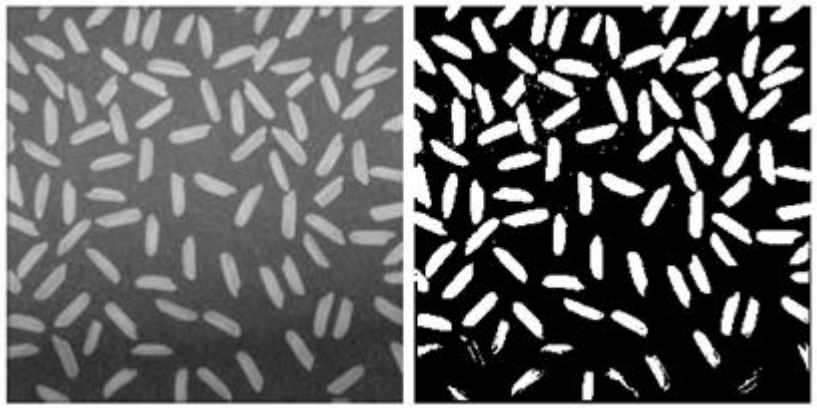
Opción 2:

```
T2 = 255*graythresh(f) % Salida de graythresh está entre 0 y 1
```

Umbralado Local

El umbralado global puede fallar cuando la iluminación del fondo no es pareja. En lugar de utilizar un mismo umbral para toda imagen, éste puede depender del pixel en consideración, esto es:

$$g(x,y) = \begin{cases} 1 & si\ f(x,y) \geq T(x,y) \\ 0 & si\ f(x,y) < T(x,y) \end{cases} \quad con\ \ T(x,y) = f_O(x,y) + T_O$$



Umbral global

Umbral local
Usando top-hat como T(x,y)