Procesamiento de Imágenes

UNIDAD 3 - Detección de bordes y Segmentación

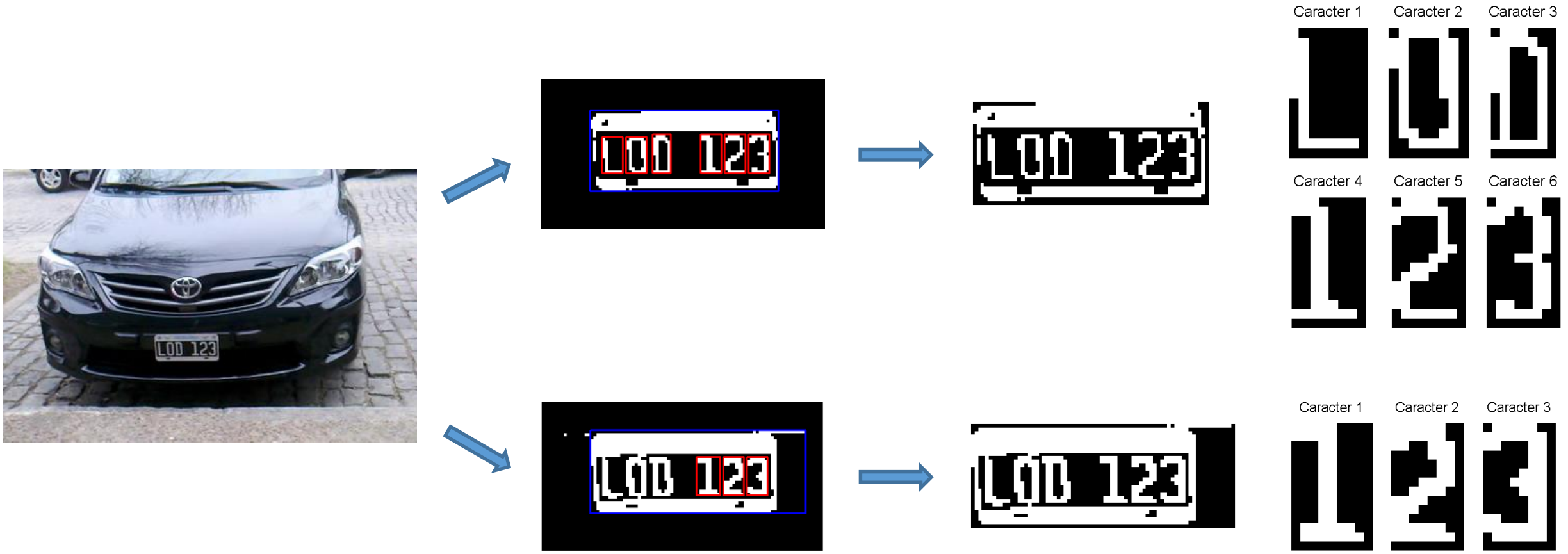
Segmentación

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.

Segmentación

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



Segmentación

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:

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

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

Detección de Punto

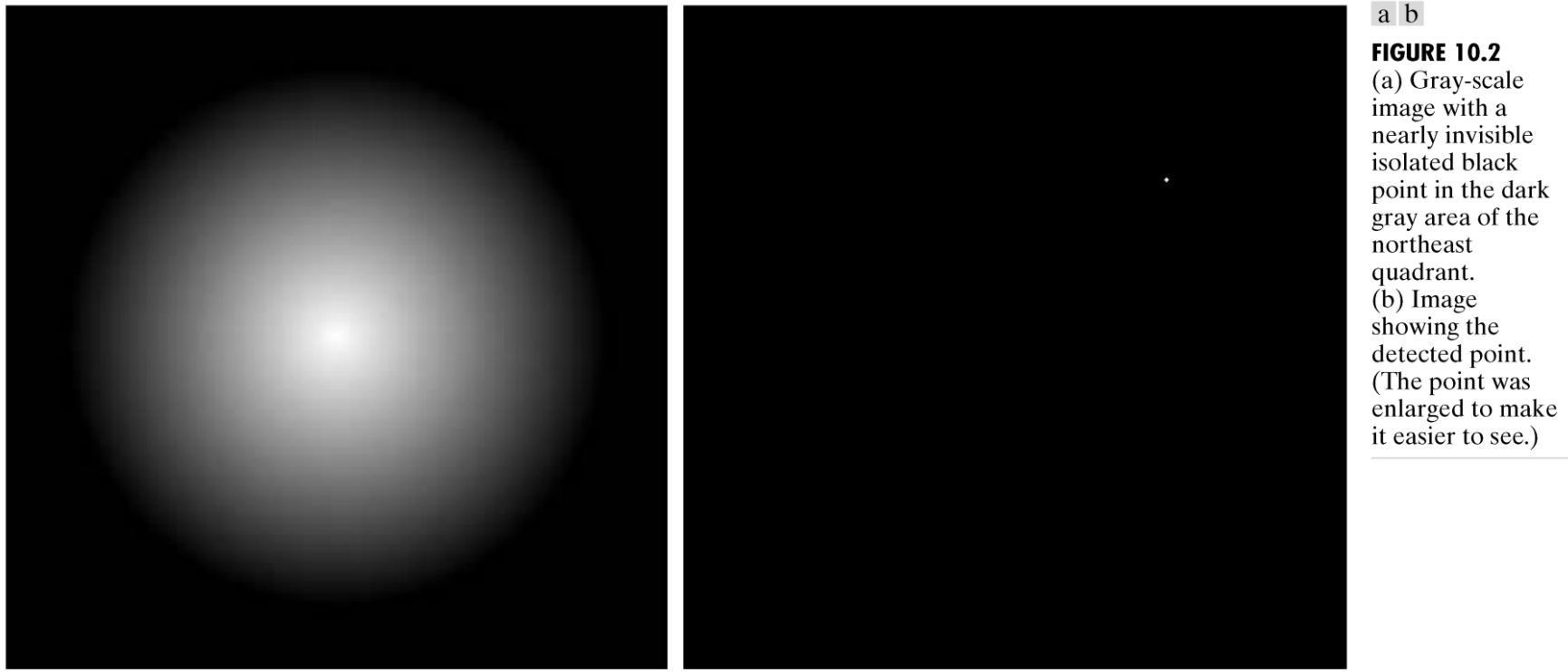
La detección de puntos contenidos en un área de intensidad relativamente constante es sencilla. Usando la máscara mostrada podemos detectar un punto cuando al centrar la máscara en el pixel se cumple:

$$| R | \geq T$$

siendo T un umbral no negativo.

−1	−1	−1
−1	8	−1
−1	−1	−1

Detección de punto

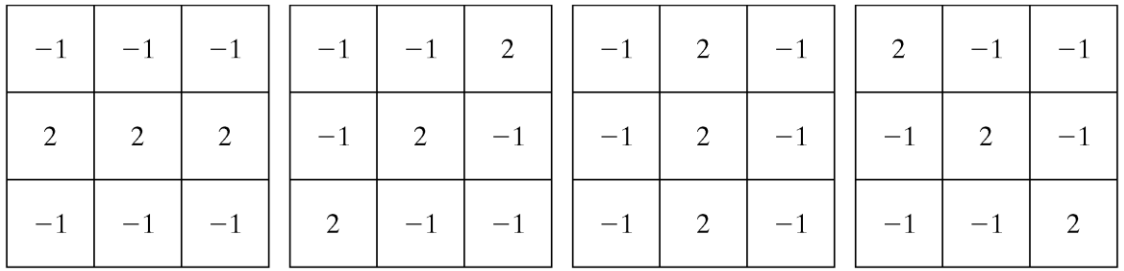


a b

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

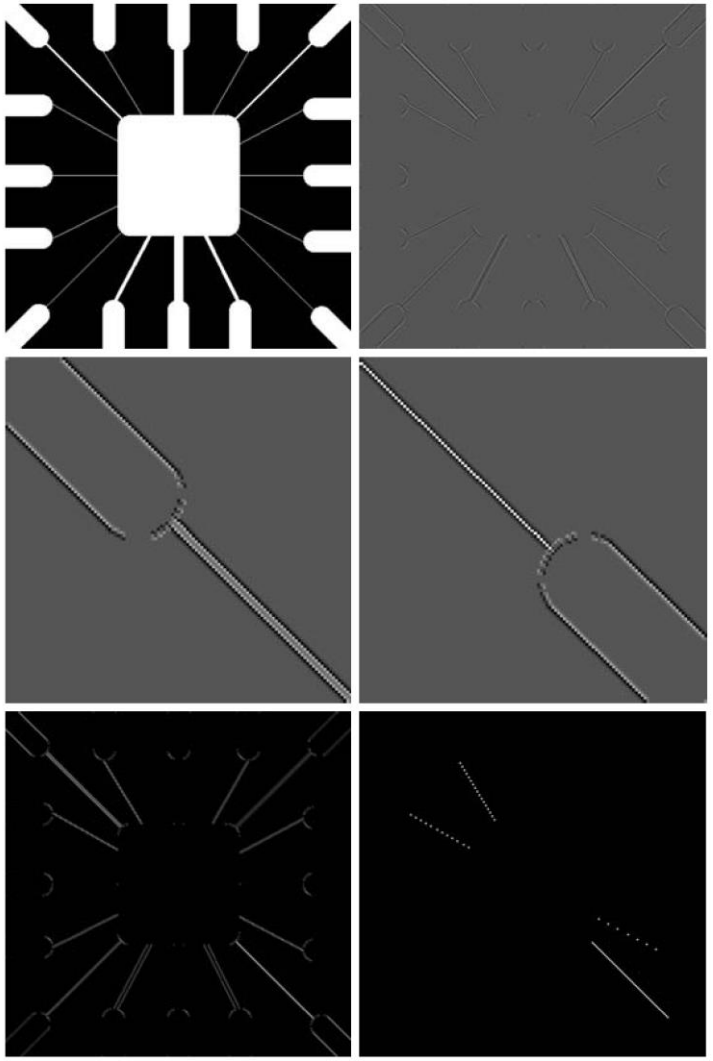
Consideremos las siguientes máscaras:



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



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.

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:

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

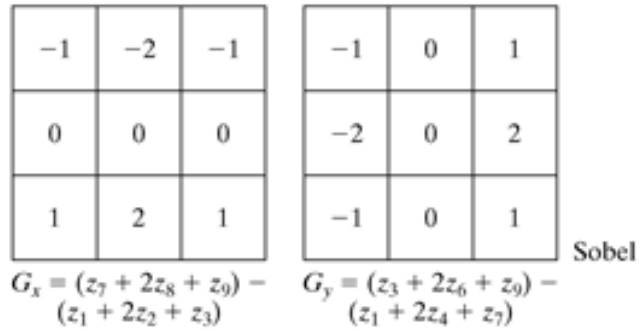
Detección de bordes - Sobel

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

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

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

```
grad = cv2.Sobel(gray, ddepth, dx, dy, ksize)
```



Detección de bordes - Prewitt

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

−1	−1	−1
0	0	0
1	1	1

$G_x = (z_7 + z_8 + z_9) - (z_1 + z_2 + z_3)$

−1	0	1
−1	0	1
−1	0	1

$G_y = (z_3 + z_6 + z_9) - (z_1 + z_4 + z_7)$

Prewitt

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:

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

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.

```
blur = cv2.GaussianBlur(img, (kx,ky), sigmax, sigmay)

laplacian = cv2.Laplacian(blur, ddepth, ksize)
```

Detección de bordes - LoG

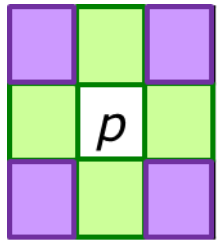
Propiedades de LoG:

- A diferencia de las derivadas de primer orden que requieren dos máscaras (x e y), LoG utiliza 1 sola, pero se pierde la orientación del gradiente.
- LoG genera una mejor localización de los bordes en comparación con las derivadas de primer orden.
- Es un filtro isotrópico, i.e., obtiene la misma magnitud de borde para todas las direcciones de bordes.
- Al igual que los derivadores de primer orden, es muy sensible al ruido.

Etiquetado de Componentes Conectados

Para definir un objeto (formado por pixels conectados) de una imagen es necesario definir primero los diferentes tipos de vecinos. Sea un pixel p con coordendas (x,y) sus 4-vecinos, $N_4(p)$, serán los pixels:

(x+1,y) (x-1,y) (x,y+1) (x,y-1)



De manera similar sus vecinos diagonales, ND(p), serán:

(x+1,y+1) (x+1,y-1) (x-1,y+1) (x-1,y-1)

La unión de $N_4(p)$ y ND(p) forman los 8-vecinos de p, $N_8(p)$.

Luego, decimos que los pixels p y q son 4-adyacentes si q $\in$ $N_4(p)$ y de manera similar los pixels p y q son 8-adyacentes si q $\in$ $N_8(p)$.

Etiquetado de Componentes Conectados

Un camino desde p_1 a p_n podrá ser 4-conectado u 8-conectado si en la secuencia de pixels $p_1, p_2, \ldots, p_{n-1}, p_n$ cualquier par de pixels p_k y p_{k+1} son respectivamente 4-adyacentes u 8-adyacentes.

0	0	0	0	0
0	0	1	1	1
0	0	1	0	0
1	1	1	0	0
0	0	0	0	0

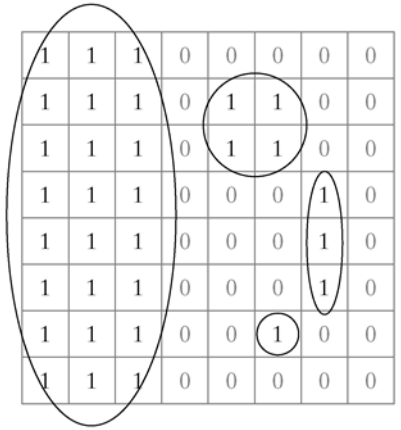
0	0	0	0	0
0	0	1	1	1
0	0	1	0	0
1	1	0	0	0
0	0	0	0	0

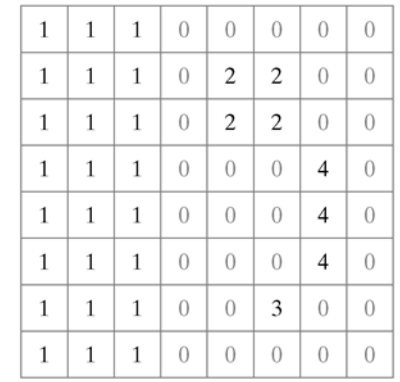
Etiquetado de Componentes Conectados

Dos pixels p y q estarán 4-conectados u 8-conectados si existe un camino 4-conectado u 8-conectado entre ellos respectivamente.

El conjunto de todos los pixels conectados a p formarán un objeto conectado o simplemente objeto.

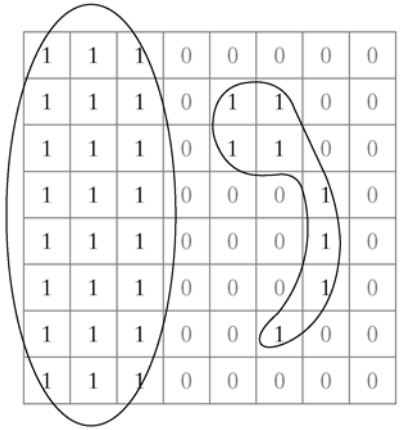
Objetos 4-conectados

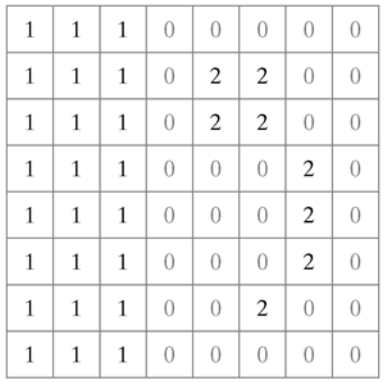




4 Objetos

Objetos 8-conectados





2 Objetos

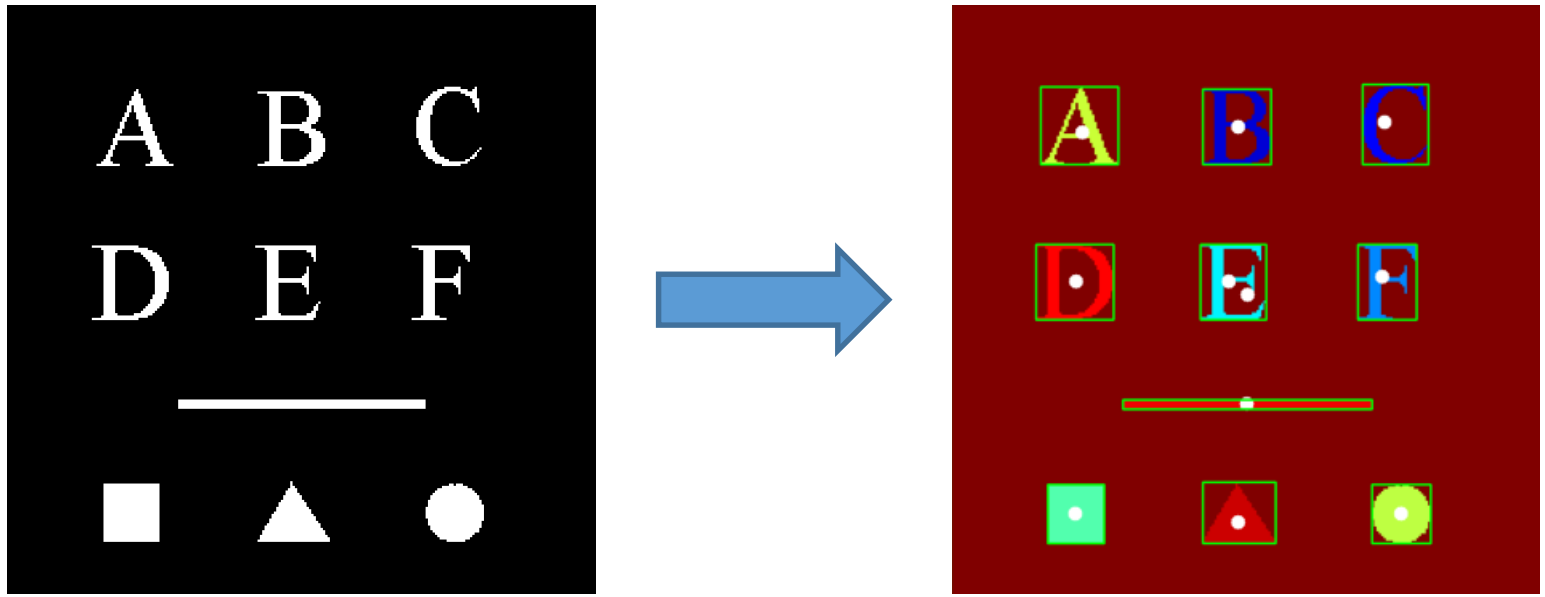
Etiquetado de Componentes Conectados

```
img = cv2.imread('objects.tif', cv2.IMREAD_GRAYSCALE)
num_labels, labels, stats, centroids = cv2.connectedComponentsWithStats(img,
connectivity, cv2.CV_32S)
```

- num_labels ⇒ Cantidad de elementos encontrados.
- labels ⇒ Imagen con etiquetas.
- stats ⇒ Estadísticas para cada etiqueta (bounding box + área).
- centroids ⇒ Coordenadas del centroide para cada etiqueta.

Etiquetado de Componentes Conectados

```
im_color = cv2.applyColorMap(np.uint8(255/num_labels*labels), cv2.COLORMAP_JET)
for centroid in centroids:
    cv2.circle(im_color, tuple(np.int32(centroid)), 9, color=(255,255,255), thickness=-1)
for st in stats:
    cv2.rectangle(im_color,(st[0],st[1]),(st[0]+st[2],st[1]+st[3]),color=(0,255,0),thickness=2)
imshow(img=im_color, color_img=True)
```



Detección de bordes - Canny

En los métodos anteriores de detección de bordes (Sobel, LoG, etc.), vimos que se obtiene como resultado una imagen en niveles de grises, donde el valor de cada pixel representa con "cuanta intensidad" existe un borde en dicha posición.

Ejemplos (asumamos rango de valores [0 255]):

- 0: En esa posición no existe un borde.
- 255: En esa posición existe un borde muy marcado.
- 128: En esa posición existe un borde, pero no demasiado marcado.

Luego se aplica un umbral (el cual se debe seleccionar de manera adecuada) para obtener una imagen binaria donde el valor True (o 255) indica la existencia de borde y el valor False (o 0) representa los pixels que no son bordes.

Detección de bordes - Canny

Imagen

Gradiente

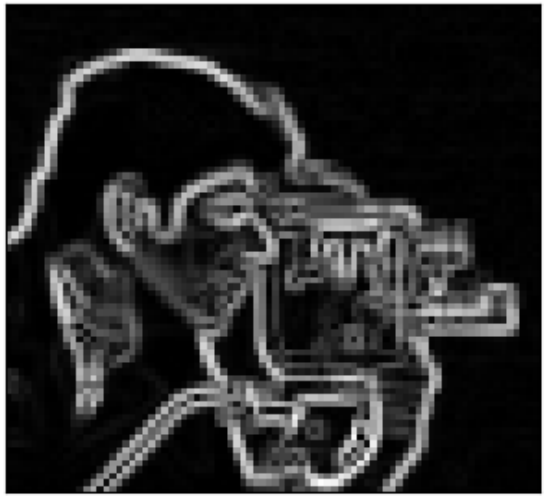
Gradiente horizontal

Gradiente vertical

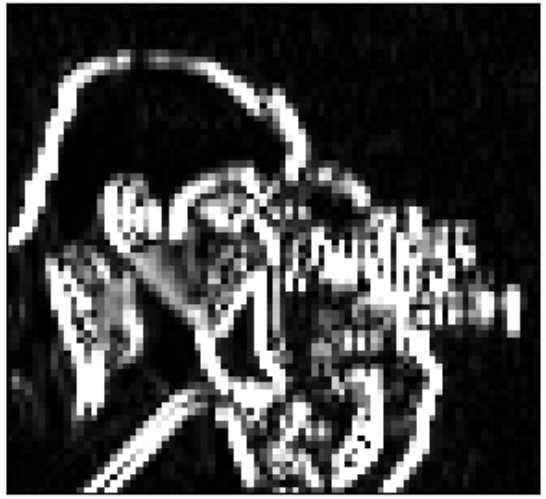
Detección de bordes - Canny

Imagen

Gradiente



Gradiente horizontal



Gradiente vertical

Detección de bordes - Canny

Gradiente

Gradiente horizontal

Gradiente vertical

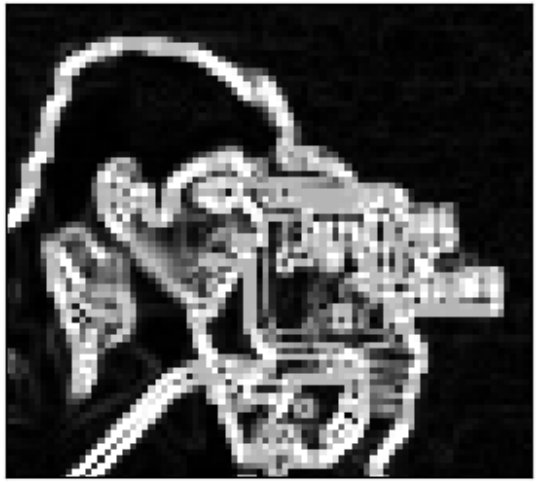
Gradiente + umbral

Gradiente horizontal + umbral

Gradiente vertical + umbral

Detección de bordes - Canny

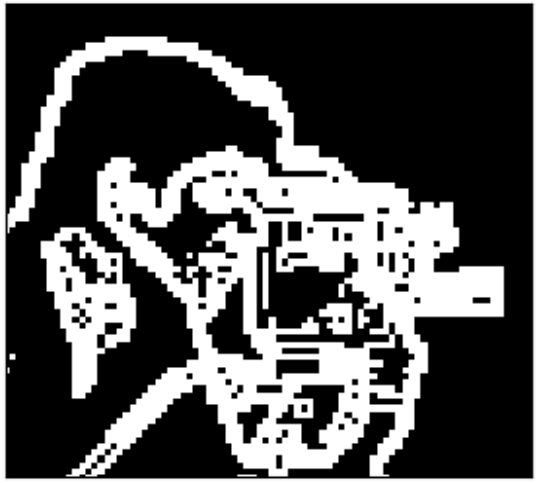
Gradiente



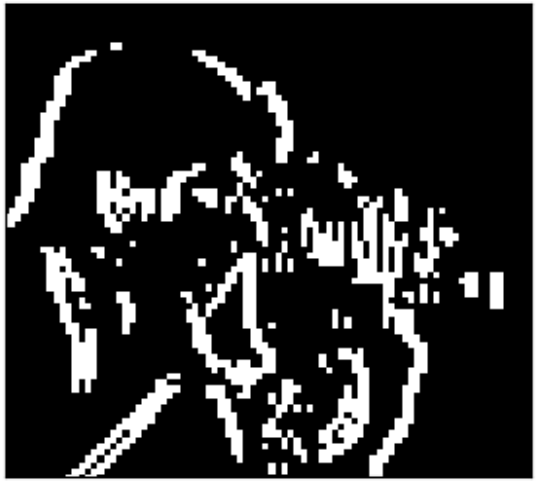
Gradiente horizontal

Gradiente vertical

Gradiente + umbral



Gradiente horizontal + umbral



Gradiente vertical + umbral

Detección de bordes – Canny
Motivación

Como puede observarse, esta metodología posee ciertos problemas:

1) Los bordes generalmente son "gruesos".
2) Si se quisiera obtener bordes mas finos se debe bajar el umbral, pero esto hace que empiecen a desaparecer otros bordes mas finos.
3) Generalmente existen muchos bordes detectados que no están conectados con los demás, es decir, están aislados (~ ruido).

Debido a estos problemas, se desarrollaron métodos de detección de bordes mas robustos, los cuales agregan etapas de pre-procesamiento y post-procesamientos para poder solucionarlos y obtener bordes mas finos, continuos y definidos.

Uno de estos métodos es el método de detección de bordes de Canny.

Detección de bordes - Canny

El detector de bordes de Canny es un operador de detección de bordes que utiliza un algoritmo de varias etapas para detectar una amplia gama de bordes en imágenes. Fue desarrollado por John F. Canny en 1986. Canny también elaboró una teoría computacional de la detección de bordes que explica por qué funciona la técnica.

Detección de bordes - Canny
Resumen

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).

Detección de bordes - Canny Resumen

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 mayores a T1 son considerados bordes poco probables.

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

```
f_blur = cv2.GaussianBlur(f, ksize=(3, 3), sigmaX=2)

edges = cv2.Canny(f_blur, threshold1=0.04*255, threshold2=0.1*255)
```

Detección de bordes - Canny
Procedimiento

El algoritmo de detección de bordes Canny consta de 5 pasos:

1. Reducción del ruido.
2. Cálculo del gradiente (magnitud y ángulo).
3. Supresión de no máximos.
4. Umbralado doble.
5. Seguimiento de bordes por histéresis.

Detección de bordes - Canny
Paso 1 – Reducción de ruido

Dado que este método se basa en el cálculo del gradiente de la imagen, es decir, en el cálculo de la derivada, los resultados de la detección de bordes son muy sensibles al ruido de la imagen. Una forma de eliminar el ruido de la imagen es aplicar un filtro pasa-bajos. Generalmente se suele aplicar un filtro Gaussiano con un kernel de 3x3 y un sigma igual a 2. Luego estos parámetros se pueden ajustar para obtener mejores resultados, pero son un buen punto de partida.

Detección de bordes - Canny
Paso 2 – Gradiente

El paso de cálculo del gradiente detecta la intensidad y la dirección del borde calculando el gradiente de la imagen mediante operadores de detección de bordes. Generalmente se suelen utilizar Sobel de 3x3 para el cálculo:

−1	−2	−1
0	0	0
1	2	1

$G_x = (z_7 + 2z_8 + z_9) - (z_1 + 2z_2 + z_3)$

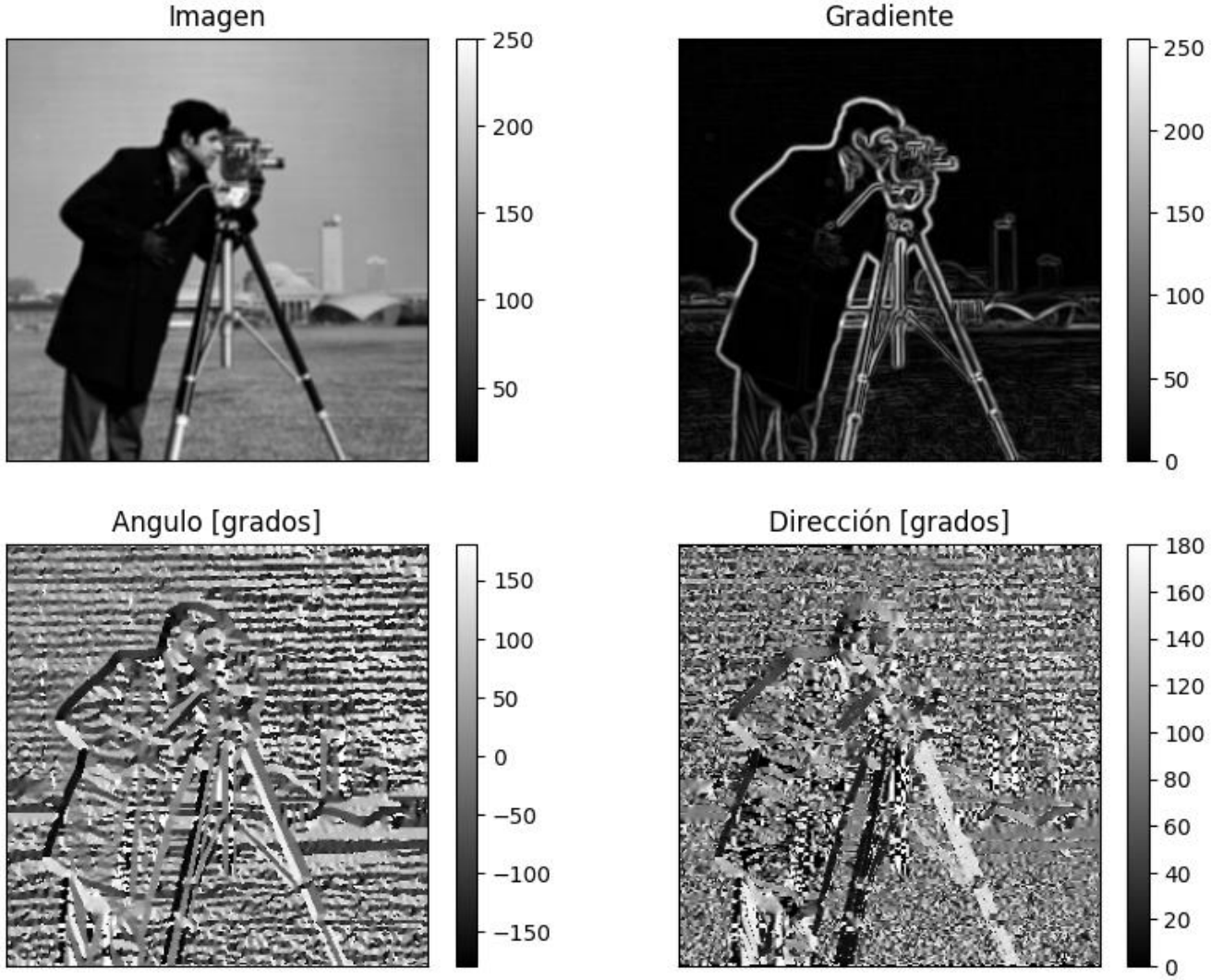
−1	0	1
−2	0	2
−1	0	1

$G_y = (z_3 + 2z_6 + z_9) - (z_1 + 2z_4 + z_7)$

Sobel

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

Detección de bordes - Canny
Paso 2 – Gradiente



Detección de bordes - Canny
Paso 3 – Supresión de no máximos

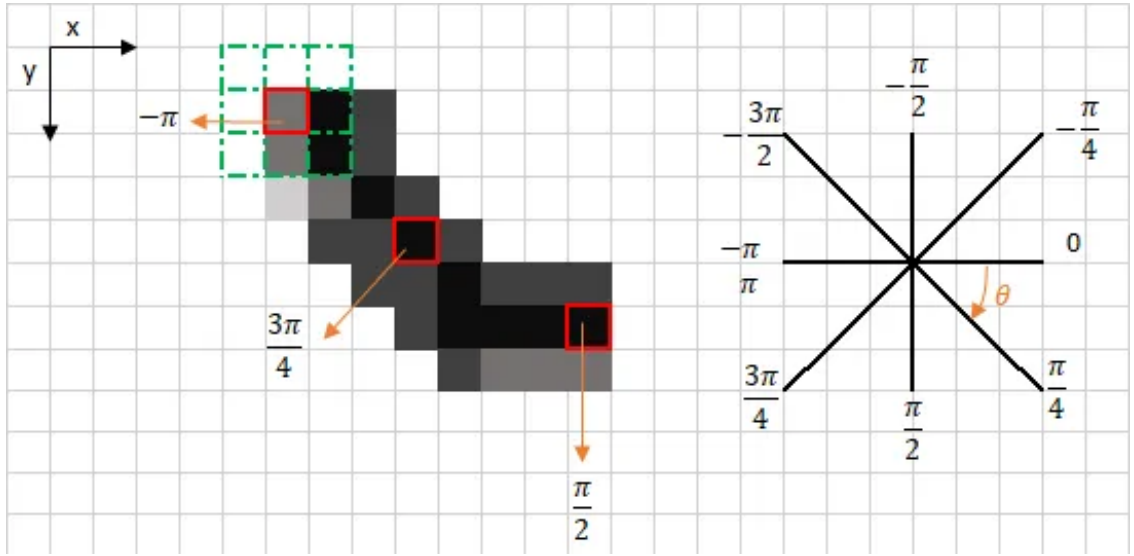
Idealmente, el resultado debería ser una imagen cuyos bordes sean finos. Como ya vimos anteriormente, el resultado obtenido al calcular el gradiente + umbral no genera buenos resultados, por lo cual, se debe aplicar algún método para hacer mas finos los bordes. En este método se realiza la supresión de los no máximos.

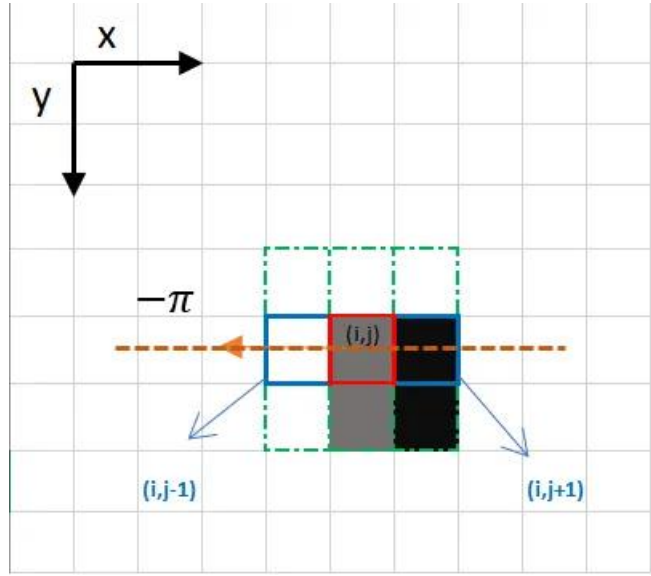
El principio es sencillo: el algoritmo recorre todos los puntos de la matriz de intensidad del gradiente (magnitud) y encuentra los píxeles con el valor máximo en las direcciones de los bordes (ángulo del gradiente).

Si el valor del pixel bajo análisis no es el mayor en la dirección del gradiente (dentro de un vecindario pre-definido), su valor se lleva a 0, caso contrario, mantiene su valor. El objetivo del es comprobar si los píxeles de la misma dirección son más o menos intensos que el que se está analizando.

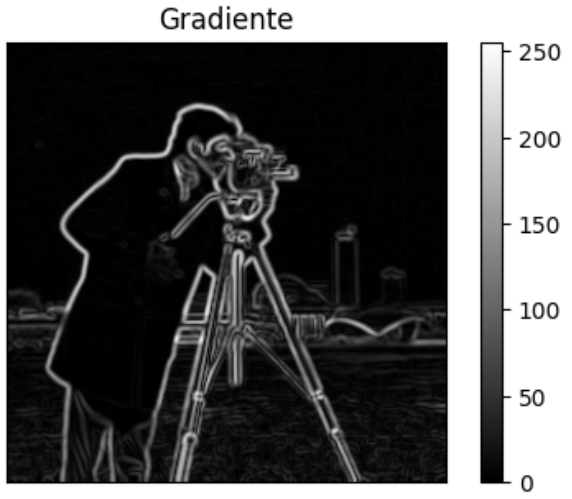
Detección de bordes - Canny
Paso 3 – Supresión de no máximos

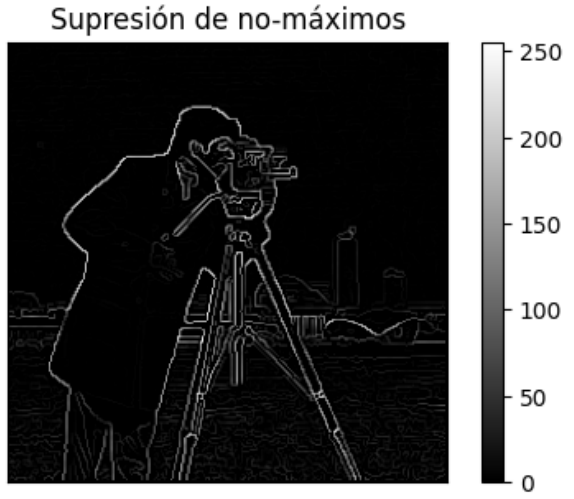
En este ejemplo, se está analizando el pixel (i, j), y los pixeles en la misma dirección están resaltados en azul (i, j-1) y (i, j+1). Si uno de esos dos pixeles es más intenso que el que se está analizando, su valor se lleva a 0. El pixel (i, j-1) parece ser más intenso, porque es blanco (255). Por lo tanto, el valor de intensidad del pixel actual (i, j) se establece en 0. Si no hay pixeles en la dirección del borde que tengan valores más intensos, entonces se mantiene el valor del pixel actual.

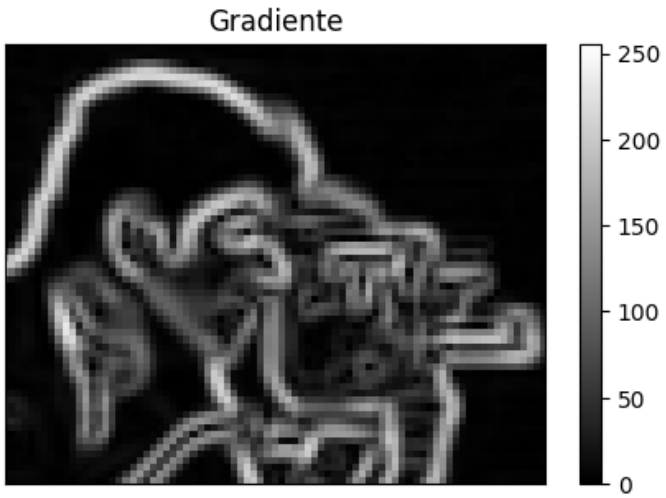




Detección de bordes - Canny
Paso 3 – Supresión de no máximos







Detección de bordes - Canny
Paso 4 – Umbralado doble

Este paso tiene como objetivo identificar 3 tipos de píxeles: fuertes, débiles y no relevantes:

1. Los píxeles fuertes son los que tienen una intensidad tan alta que estamos seguros de que contribuyen al borde final.
2. Los píxeles débiles son píxeles que tienen un valor de intensidad que no es suficiente para ser considerados fuertes, pero no lo suficientemente pequeño para ser considerados no relevantes para la detección del borde.
3. Los demás píxeles se consideran no relevantes para el borde.

Detección de bordes - Canny
Paso 4 – Umbralado doble

Por lo tanto, se definen 2 umbrales: umbral alto y umbral bajo:

1. El umbral alto se utiliza para identificar los píxeles fuertes (intensidad superior al umbral alto).
2. El umbral bajo se utiliza para identificar los píxeles no relevantes (intensidad inferior al umbral bajo).
3. Todos los píxeles que tienen una intensidad entre ambos umbrales se marcan como débiles y el mecanismo de histéresis (paso siguiente) ayudará a identificar los que podrían considerarse fuertes y los que se consideran no relevantes.

Detección de bordes - Canny

Paso 4 – Umbralado doble

Supresión de no-máximos

Doble umbralado

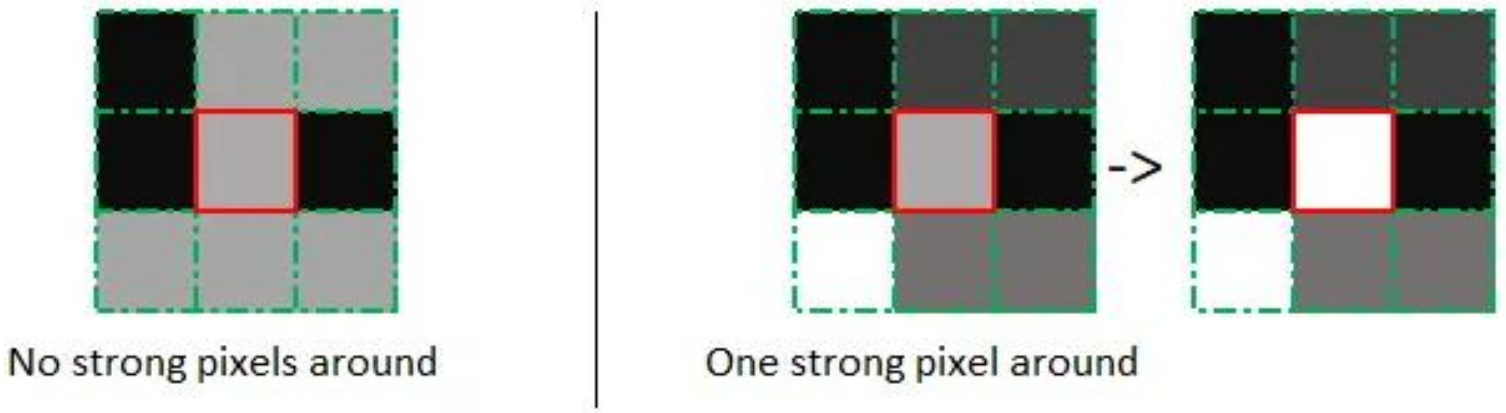
Supresión de no-máximos

Doble umbralado

Detección de bordes - Canny
Paso 5 – Histéresis

Basándose en los resultados del paso anterior (umbralado doble), el seguimiento de bordes por histéresis consiste en aplicar un ciclo de histéresis para decidir si conservar los bordes débiles o no en el resultado final.

Esta histéresis consiste en transformar los píxeles débiles en fuertes, si y sólo si al menos uno de los píxeles que rodean al que se está analizando (8-vecinos) es un píxel fuerte.



Detección de bordes - Canny

Paso 5 – Histéresis

Doble umbralado

Resultado Canny

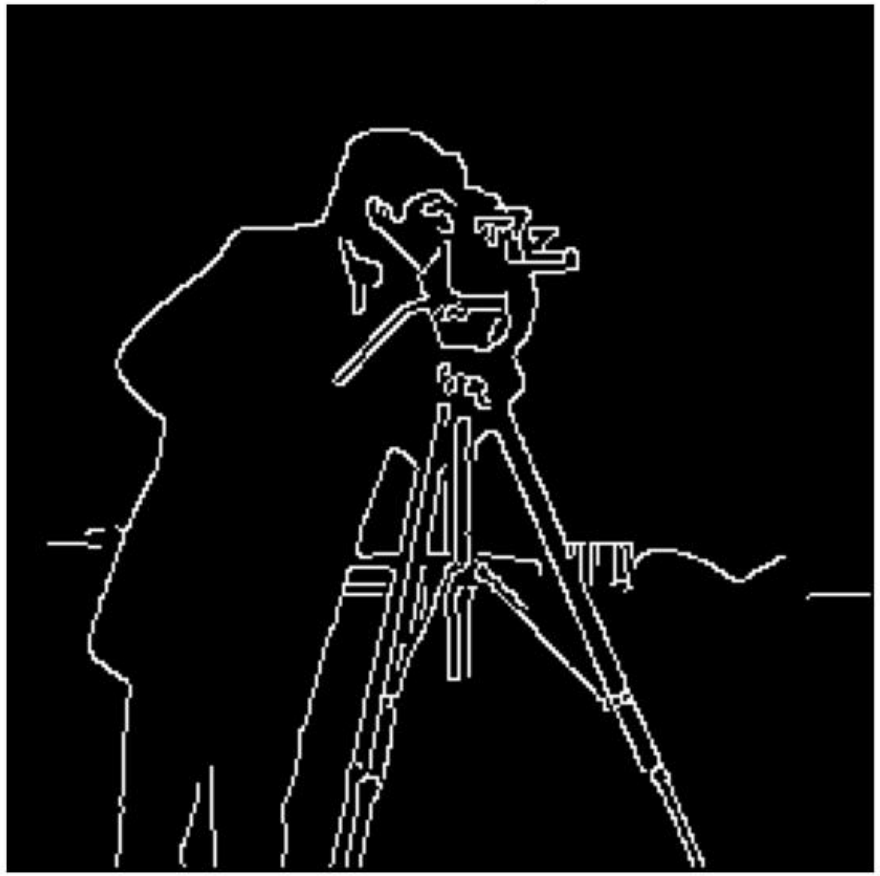
Doble umbralado

Resultado Canny

Detección de bordes - Canny
Resultado Final

Imagen

Resultado Canny

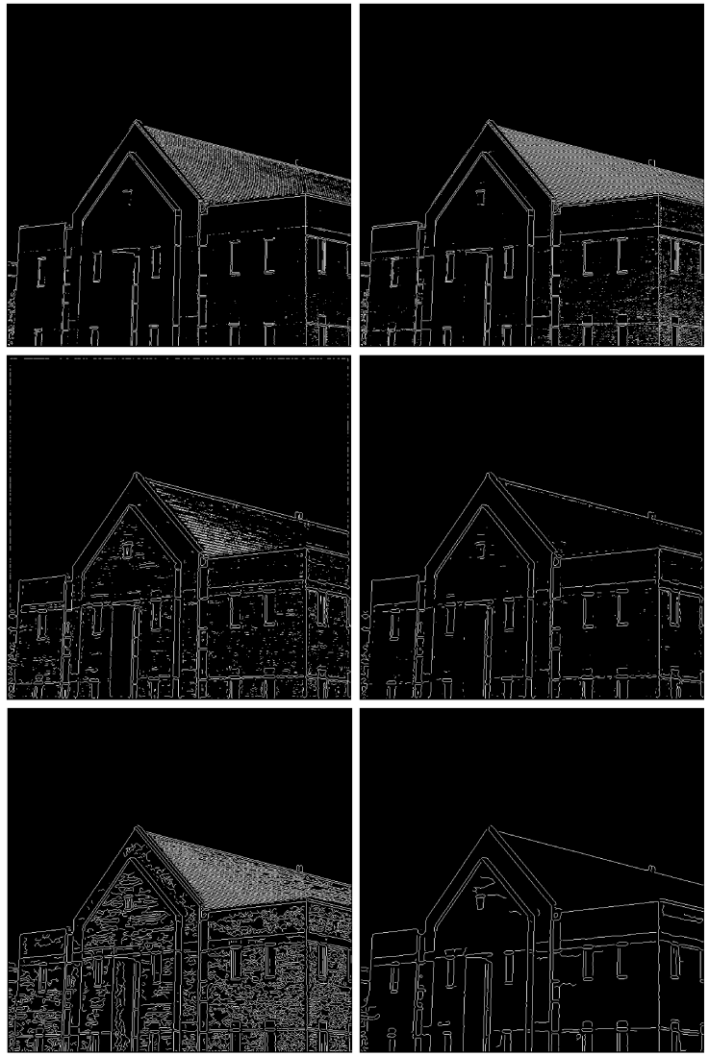


Detección de bordes - Canny

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 - Canny



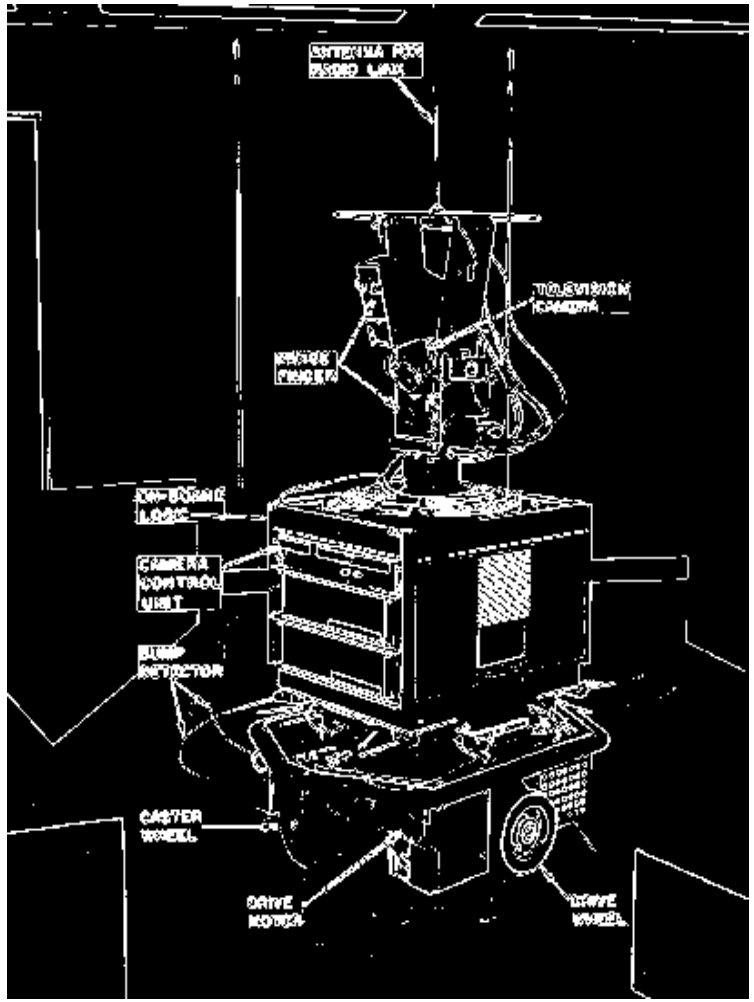
a b
c d
e f

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 de Hough

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

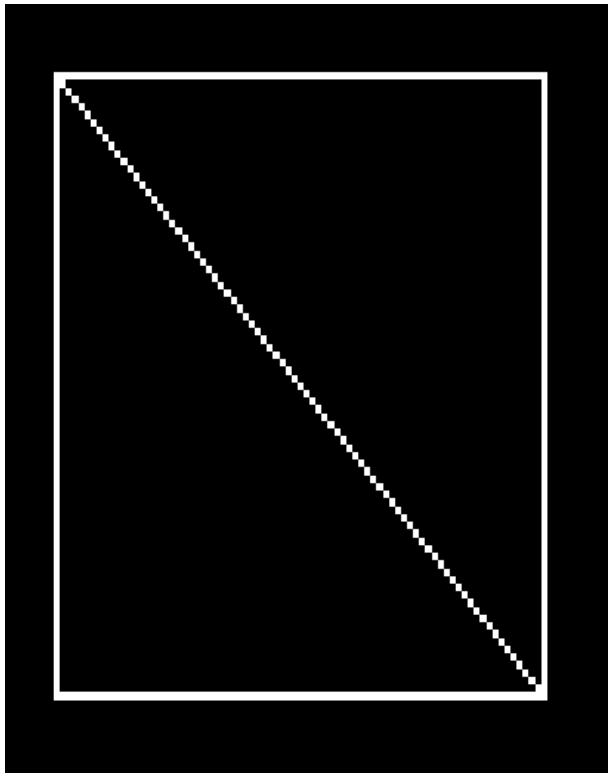
Dado un conjunto de puntos en una imagen binaria queremos encontrar el subconjunto de puntos que se encuentran formando una línea recta.

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)

Detección de Líneas Rectas Transformada de 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.



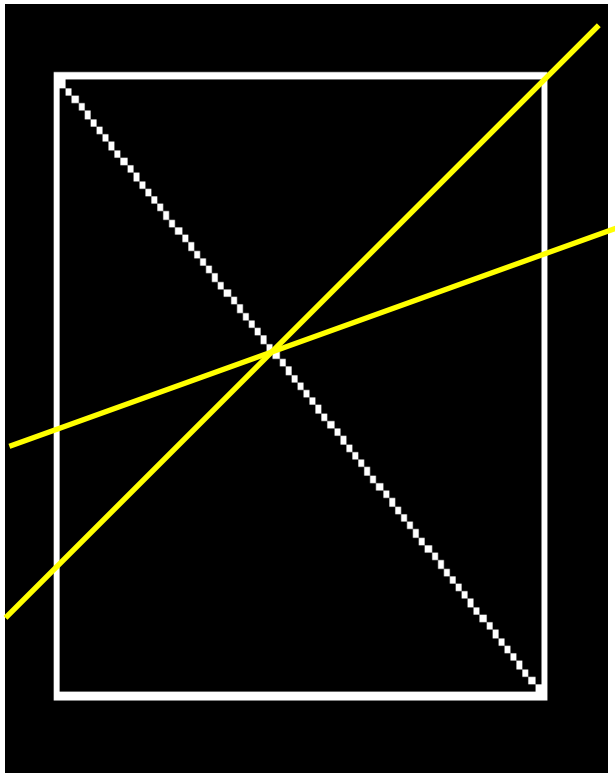
Detección de Líneas Rectas Transformada de Hough

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.



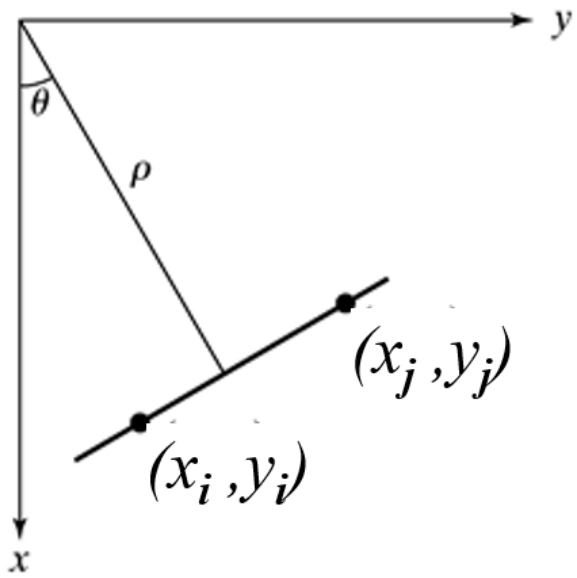
Detección de Líneas Rectas Transformada de Hough

Cada recta es representada con 2 parámetros:

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

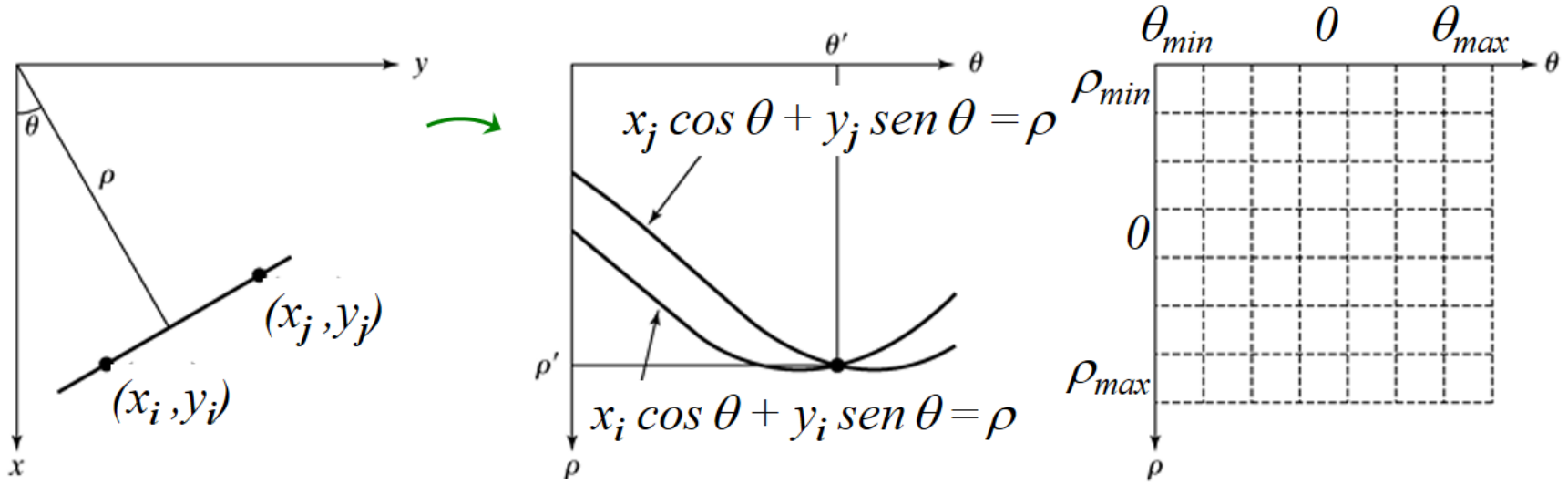
Las rectas se buscan para los posibles valores de rho y theta.

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



Detección de Líneas Rectas Transformada de 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.

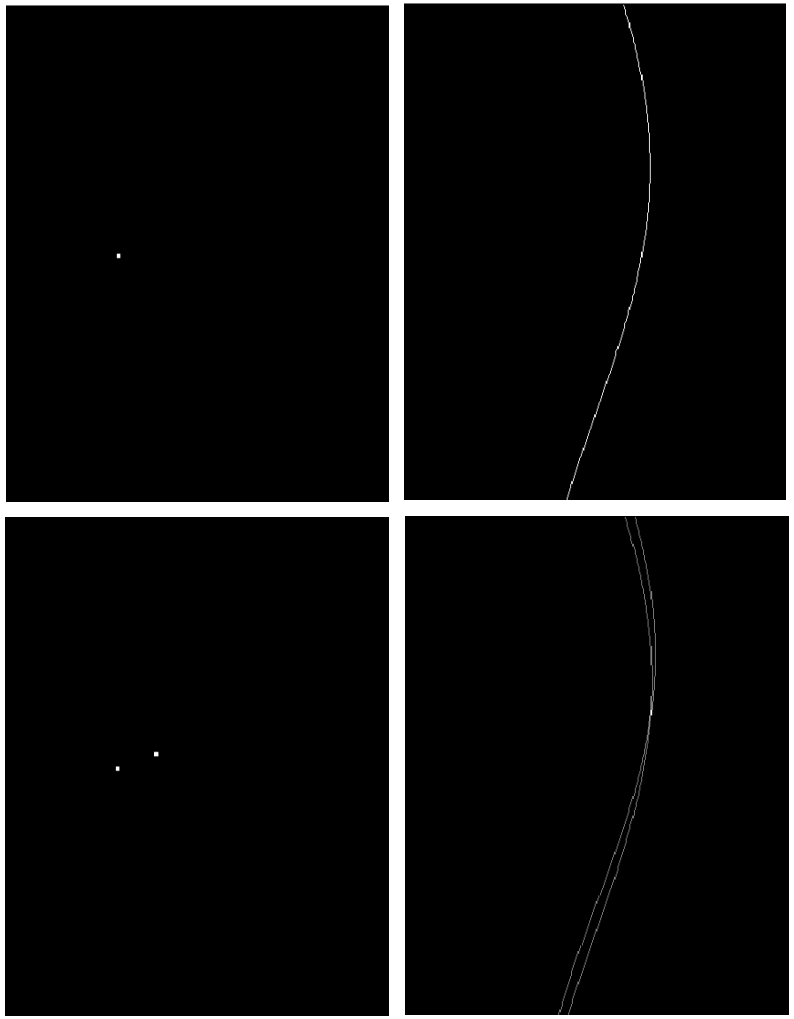


Detección de Líneas Rectas Transformada de Hough

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

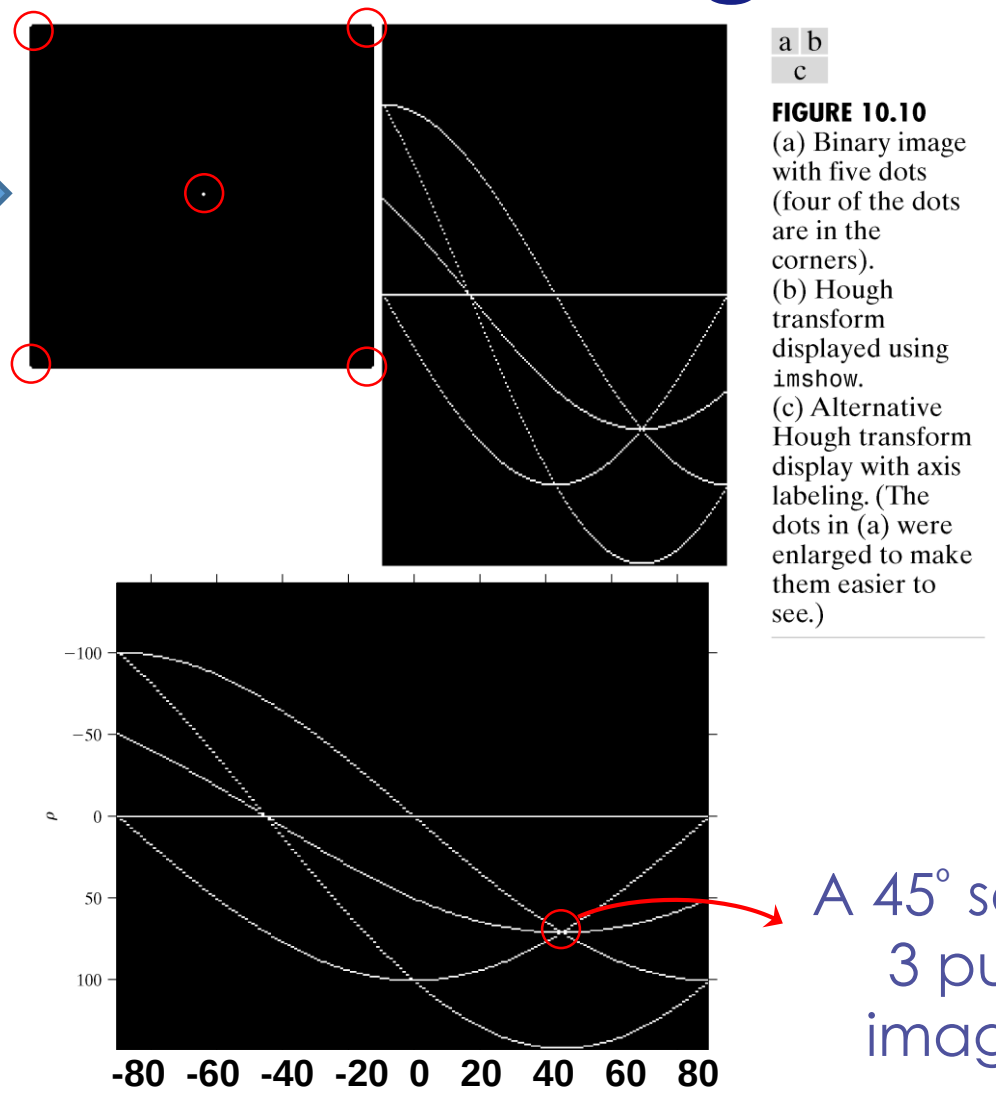
Dos puntos corresponden a 2 curvas en el espacio de Hough.

La intersección de estas 2 curvas suma 2 votos.



Detección de Líneas Rectas Transformada de 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

Detección de Líneas Rectas Transformada de Hough

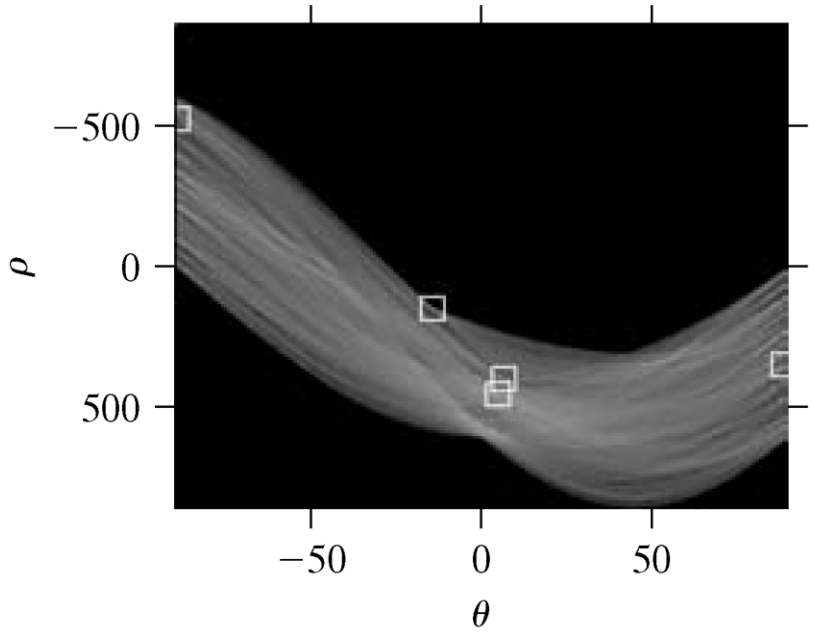
En general, los bordes de los objetos en una imagen no son rectos, y los picos en el espacio transformado no estarán en una única celda.

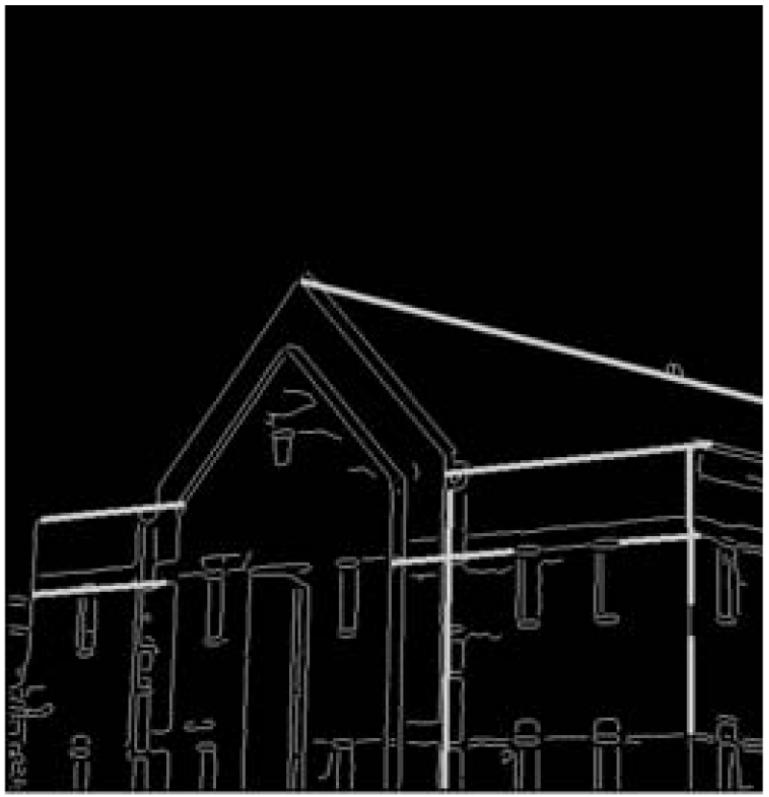
Una estrategia para resolver esto puede ser:

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.

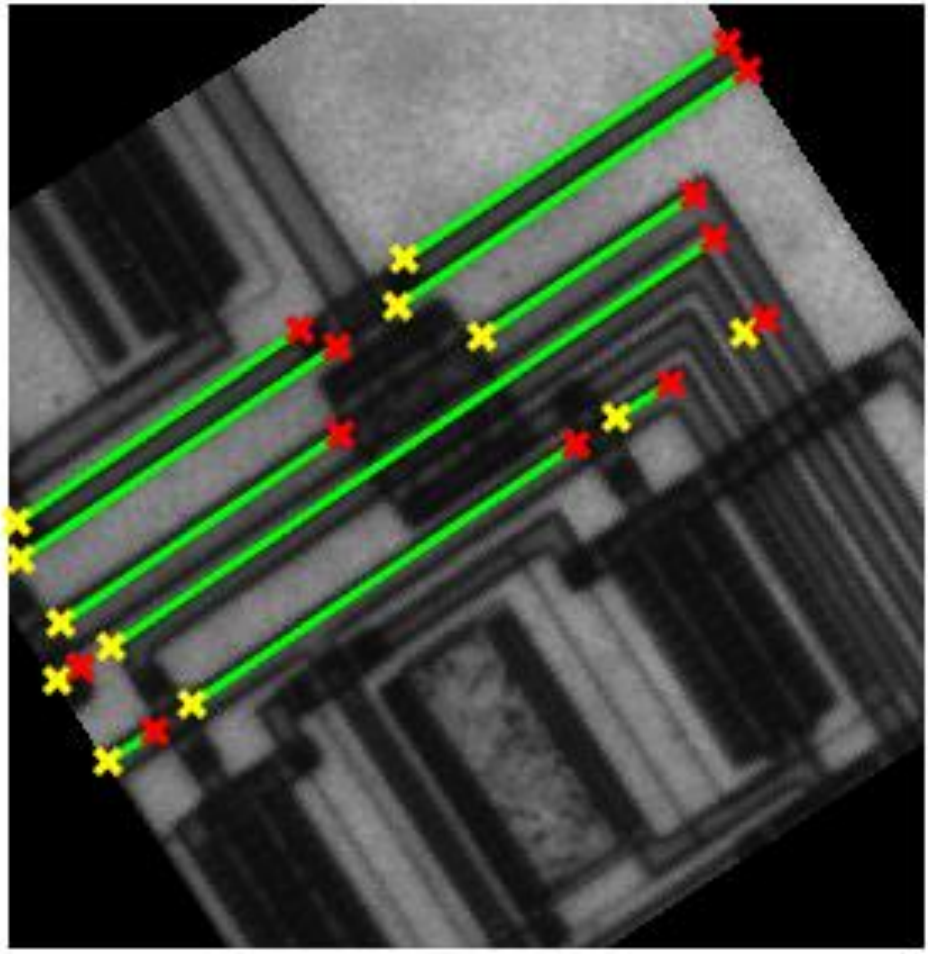
```
lines = cv2.HoughLines(edges, rho=1, theta=np.pi/180, threshold=100)
```

Detección de Líneas Rectas
Transformada de Hough





Detección de Líneas Rectas
Transformada de Hough



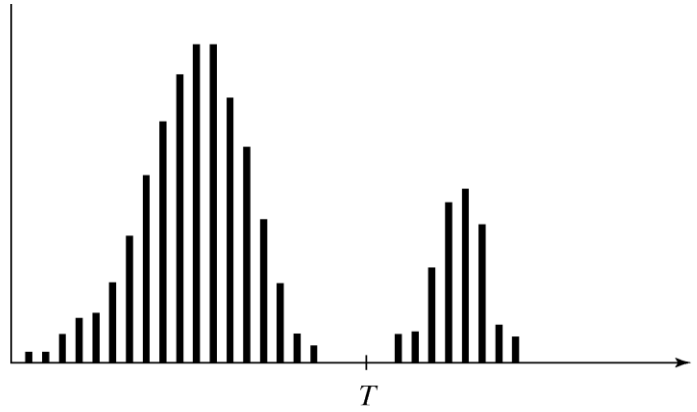
Umbralado Global

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 pixels: G1 pixels con valores ≥ T y G2 los pixels con valores < T.
3. Calcular las intensidades promedio, μ1 y μ2, 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

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:

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

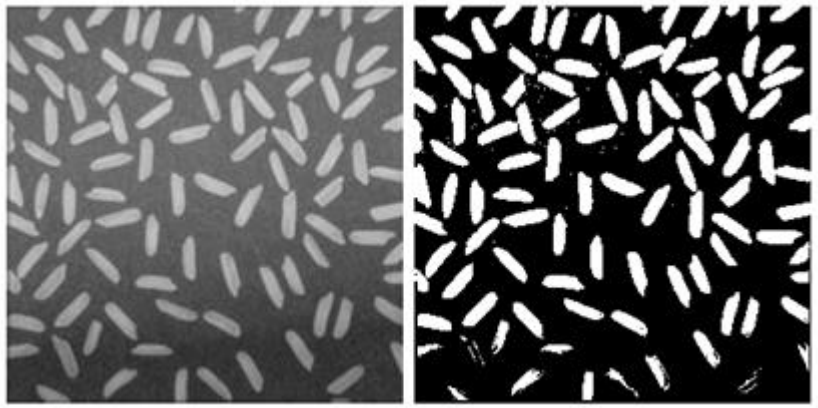
donde:

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

Umbralado 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)