CAPÍTULO 4: PROCESAMIENTO DIGITAL DE IMÁGENES

4.1.- Introducción

El *toolbox* de procesamiento de imágenes de Matlab contiene un conjunto de funciones de los algoritmos más conocidos para trabajar con imágenes binarias, transformaciones geométricas, morfología y manipulación de color que junto con las funciones ya integradas en Matlab permite realizar análisis y trasformaciones de imágenes en el dominio de la frecuencia.

La segmentación de una imagen es un proceso de extracción de objetos de interés insertados en la escena capturada. La agrupación de los píxeles se hace a razón de que sus vecinos sean similares en criterios como de luminancia, color, bordes, texturas, movimientos,... Una vez que la imagen ha sido particionada, la unidad dejará de ser el píxel para ser la agrupación de píxeles que constituye el objeto. La imagen estará definida por un conjunto de objetos, habiendo pasado de un nivel bajo a otro más elaborado o nivel medio visual. La información estará preparada para el reconocimiento e interpretación de la imagen.

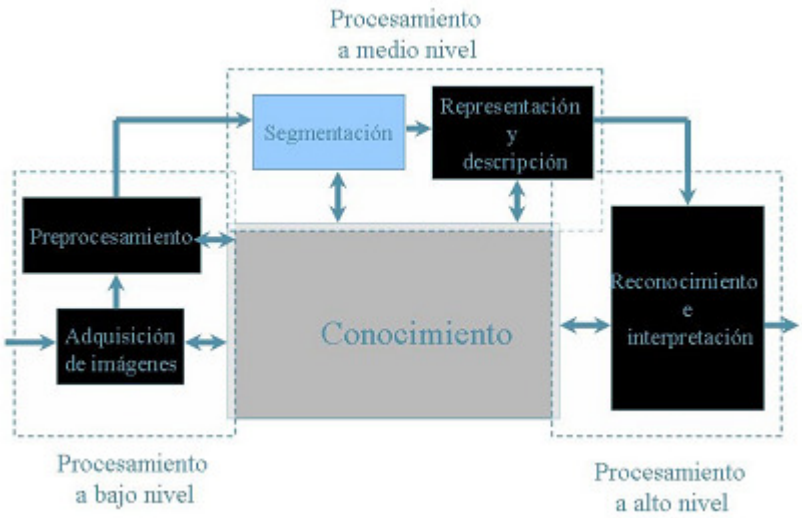

Figura 4.1: Esquema general de Visión Artificial

Para la segmentación de las imágenes se usan tres conceptos básicos:

Similitud: los píxeles agrupados del objeto deben ser similares respecto algún criterio (nivel de gris, color, borde, textura,....).

Conectividad: los objetos corresponden a áreas de píxeles con conectividad. Las particiones corresponden con regiones continuas de píxeles.

Discontinuidad: los objetos tienen formas geométricas que definen unos contornos. Estos bordes delimitan unos objetos de otros.

En la práctica, la imposición de estas condiciones sobre la estrategia de segmentación resulta casi imposible. Respecto a la similitud, la aparición del ruido, presente en todas las imágenes, la falta de iluminación uniforme sobre el escenario, la creación de sombras de unos objetos sobre otros, produce que algo que parecía sencillo de definir como es alguna regla sobre similitud, resulte impracticable de acotar. Todas ellas fracasan sobre escenas más o menos complejas.

El procesamiento de imágenes proporciona algoritmos con los que se pueden identificar líneas, mediante técnicas de detección de bordes y la transformada de Hough.

4.2.- Adquisición de imágenes

Para leer imágenes contenidas en un archivo con extensión permitida en Matlab se utiliza la función ***'imread'***, cuya sintaxis es

>> *imread* ('nombre del archivo')

Donde nombre del archivo es una cadena de caracteres conteniendo el nombre completo de la imagen con su respectiva extensión. Los formatos de imágenes que soporta Matlab son los mostrados en la tabla 4.1.

Formato	Extensión
TIFF	.tiff
JPEG	.jpg
GIF	.gif
BMP	.bmp
PNG	.png
XWD	.xwd

Tabla 4.1: Formatos y extensiones soportadas por Matlab.

Para introducir una imagen guardada en un archivo con alguno de los formatos especificados en la tabla anterior solo tiene que usarse la función *imread* y asignar su resultado a una variable que representará a la imagen.

Si la imagen es en escala de grises (*grayscale*) entonces *imread* devuelve una matriz bidimensional. Si la imagen es RGB entonces *imread* devuelve un arreglo tridimensional.

Para desplegarla en pantalla se puede usar el comando ***'imshow'***:

```
[nombre direc]=
uigetfile({'*.jpg;*.jpeg;*.bmp;*.gif;*.png;*.tiff'},'Abrir Imagen');
img = imread(fullfile(direc,nombre))
imshow(img)
```

4.3.- Procesamiento de la imagen

El número de funciones que implementa el *toolbox* para el procesamiento de imagenes es muy diverso, sin contar la múltiple oferta de funciones ya generada por otros usuarios y disponibles a través del Internet, sin embargo en este capítulo serán tratadas solo aquellas que se han empleado para la detección de las paredes del plano en el simulador que se está tratando en este Proyecto.

La imagen tomada en el proceso de adquisición estará compuesta por líneas negra sobre un fondo blanco, las cuales son parte de la trayectoria a seguir. Por lo tanto se necesita extraer la línea (umbralización) y obtener sus características (detección de bordes), para realizar la descripción de ésta de una manera más simple. El primer paso es convertir a escala de grises la imagen almacenada.

4.3.1.- Funciones para la conversión de formatos de color

El formato de representación de color ofrecido por las imágenes RGB resulta no apropiado para aplicaciones en las cuales el cambio de iluminación es problema. Otro

tipo de formatos de color menos sensibles al cambio de iluminación han sido propuestos, tales como el modelo HSV. Matlab dispone de funciones especiales para realizar cambios entre modelos de color y para convertir imágenes de color a escala de grises (figura 4.2 (a) y (b)); algunas de esas funciones serán tratadas en este apartado.

La función ***'rgb2gray'*** cambia una imagen en formato RGB a escala de grises, el formato de dicha función es:

>> imagegray = *rgb2gray* (imageRGB);

Por su parte la función ***'rgb2hsv'*** cambia del modelo de color RGB al modelo HSV, esta función toma como entrada una imagen RGB compuesta de tres planos y devuelve la imagen convertida al modelo HSV compuesta a su vez de tres planos correspondientes al H, S y V. El formato de esta función es:

>> imagehsv = *rgb2hsv* (imageRGB);

La conversión contraria la realiza la función hsv2rgb.

Figura 4.2 (a): Imagen RGB

Figura 4.2 (b): Imagen resultante de aplicar la función rgb2gray a la imagen (a)

4.3.2.- Detección de bordes

4.3.2.1.- Introducción

En el área de procesamiento de imágenes, la detección de los bordes de una imagen es de suma importancia y utilidad, pues facilita muchas tareas, entre ellas, el reconocimiento de objetos o la segmentación de regiones, entre otras.

Un contorno es el límite entre dos regiones con niveles de gris distintos, de manera que éstos sean los suficientemente homogéneos para que la transición entre dichas regiones se pueda mirar como un cambio abrupto. La idea básica para la detección de bordes es el cálculo de una derivada, teniendo en cuenta que para una constante su valor es cero y para un cambio será diferente de cero. Es decir, en las regiones donde no exista un cambio brusco, el valor de la derivada es cero y para la transición entre regiones su valor es una constante. La primera derivada de un punto de una imagen se obtiene utilizando el módulo del gradiente de ese punto y, la segunda derivada, se obtiene de forma similar utilizando el laplaciano.

Se han desarrollado variedad de algoritmos que ayudan a esta tarea. El algoritmo de Canny es usado para detectar todos los bordes existentes en una imagen. Este algoritmo está considerado como uno de los mejores métodos de detección de contornos mediante el empleo de máscaras de convolución y basado en la primera derivada. Los puntos de contorno son zonas de píxeles en las que existe un cambio brusco de nivel de gris. En el tratamiento de imágenes, se trabaja con píxeles, y en un ambiente discreto, es así que en el algoritmo de Canny se utilizan máscaras, las cuales representan aproximaciones en diferencias finitas.

En visión computacional es de utilidad para hacer reconocimiento de objetos o bien para segmentar regiones, extraer los bordes de objetos (que en teoría delimitan sus tamaños y regiones).

La detección de contornos se encuentra implementada en Matlab en el comando *'**edge**'*.

Se pueden detectar todos los bordes del fondo de imagen mediante el uso de esta función (disponible en el *toolbox* 'images\images' de Matlab). La salida de este comando se corresponde con una matriz binaria "que llamaremos BW", de tal manera que en cada posición de la matriz, se encontrará un 1 o un 0 dependiendo de si se ha detectado un borde o no:

1 = se ha detectado un borde (sobre el fondo de imagen el píxel se representa de color blanco).

0 = no se ha detectado ningún borde (sobre el fondo de imagen el píxel se representa de color negro).

La función edge permite encontrar los bordes a partir de diferentes algoritmos que pueden ser elegidos, como son Canny y Sobel, entre otros. El formato de esta función es:

>> ImageT=*edge* (ImageS, algoritmo);

Donde ImageT es la imagen obtenida con los bordes extraídos, ImageS es la variable que contiene la imagen en escala de grises de la cual se pretende recuperar sus bordes, mientras que algoritmo puede ser canny, sobel, prewitt, roberts, log o zerocross.

De tal forma que si a la imagen en escala de grises contenida en la variable imagegray se le quieren recuperar sus bordes utilizando el algoritmo de Canny se escribiría en línea de comandos:

> >ImageR=*edge* (imagegray,canny);

La figura 4.3 muestra un ejemplo práctico del uso de esta función con el algoritmo de Canny.

Figura 4.3(a): Imagen grayscale

Figura 4.3(b): Imagen resultante de aplicar la función edge a la imagen (a)

La detección de contornos es un paso intermedio en el reconocimiento de patrones en imágenes digitales. En una imagen, los contornos corresponden a los límites de los objetos presentes en la imagen. Para hallar los contornos se buscan los lugares en la imagen en los que la intensidad del píxel cambia rápidamente, generalmente usando alguno de los siguientes criterios:

- Lugares donde la primera derivada (Gradiente) de la intensidad es de magnitud mayor que la de un umbral predefinido

- Lugares donde la segunda derivada (Laplaciano) de la intensidad tiene un cruce por cero.

En el primer caso se buscarán grandes picos y en el segundo cambios de signo, tal como se muestra en la figura 4.4.

4.3.2.2.- Técnicas basadas en el gradiente

Estas técnicas se basan en una aproximación al concepto de la derivada para espacios discretos. Esta generalización se basa en el cálculo de diferencias entre píxeles

vecinos; estas diferencias, según la relación de píxeles considerados, pueden dar lugar a derivadas unidimensionales o bidimensionales, así como aplicarse en una dirección determinada de la imagen o en todas direcciones. Otras aproximaciones diferenciales de gran utilidad son la de Roberts y la de Sobel.

El operador gradiente G aplicado a una imagen *f(x,y)* está definido como:

$$\nabla f(x,y) = \begin{bmatrix} G_x & G_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} & \frac{\partial f}{\partial y} \end{bmatrix} \tag{4.1}$$

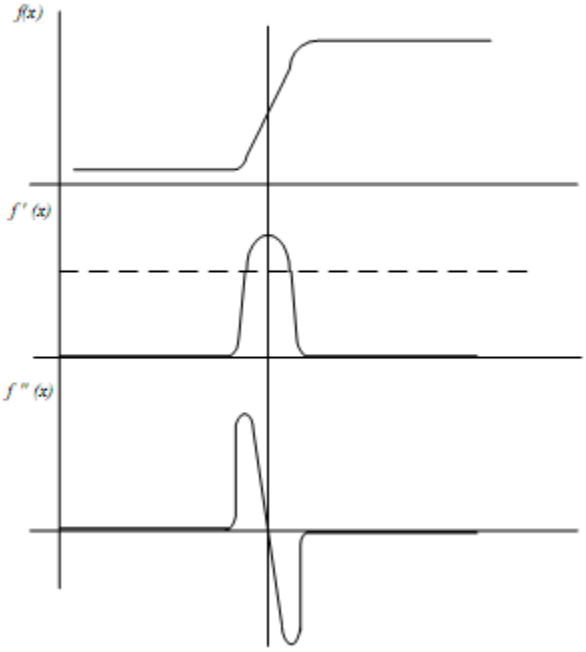

Figura 4.4: Detección de contornos mediante la primera y segunda derivada

El vector gradiente representa el cambio máximo de intensidad para el punto *(x,y);* su magnitud y dirección están dados por:

$$\begin{aligned} |\nabla f| &= \sqrt{G_{xx}^2 + G_y^2} \\ \angle \nabla f &= \arctan\left(\frac{G_y}{G_x}\right) \end{aligned} \tag{4.2}$$

siendo la dirección del gradiente perpendicular al borde. Para reducir el costo computacional, generalmente se aplica:

$$|\nabla f| = |G_x| + |G_y| \tag{4.3}$$

Debido a que las imágenes digitales no son señales continuas, se tiene:

$$\nabla f(x,y) = \begin{bmatrix} G_x & G_y \end{bmatrix} = \begin{bmatrix} \frac{\Delta f}{\Delta x} & \frac{\Delta f}{\Delta y} \end{bmatrix} \tag{4.4}$$

que se puede representar mediante las máscaras:

$$G_x = \frac{\Delta f}{\Delta x} \qquad \begin{array}{|c|c|} \hline -1 & 1 \\ \hline \end{array} \; * \; f(x,y)$$

$$G_y = \frac{\Delta f}{\Delta y} \qquad \begin{array}{|c|} \hline -1 \\ \hline 1 \\ \hline \end{array} \; * \; f(x,y) \tag{4.5}$$

Estas máscaras generalmente no se utilizan debido a que son muy poco sensibles al ruido al tomar en cuenta solamente la información de dos píxeles. Entre los filtros (operadores) más usados, que además permiten obtener un gradiente suavizado, se encuentran: Roberts, Prewitt, Sobel e Isotrópico. En la figura 4.5 se muestran las máscaras referentes a estos operadores.

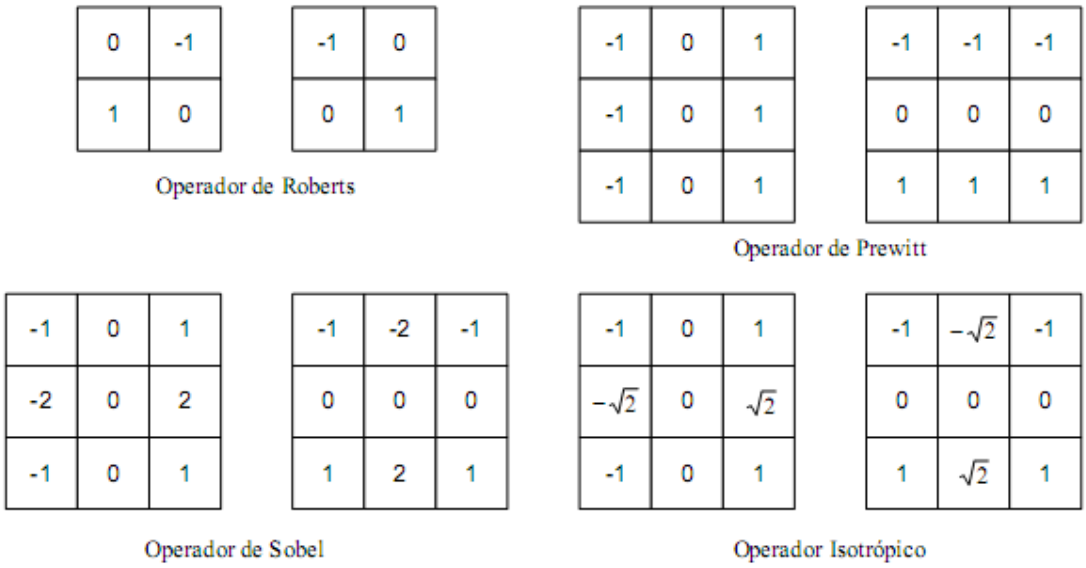

Figura 4.5: Máscaras para los operadores Roberts, Prewitt, Sobel e Isotrópico

En general, la función *'edge.m'* presenta diversos métodos para la detección de bordes, tales como el operador de Sobel, el algoritmo de Canny, etc.

En el artículo tomado como referencia para la realización de este Proyecto se utilizó el operador de Sobel, el cual se basa en el cálculo de la magnitud y dirección del

vector gradiente de cada uno de los píxeles. El vector gradiente de un píxel determinado se define como un vector orientado en la dirección de máxima variación de intensidad de dicho píxel (de negro a blanco), cuya magnitud depende de esa misma variación de intensidad. De este modo, si un píxel presenta una intensidad de brillo constante, la magnitud del vector gradiente será 0. Una vez calculada la magnitud y dirección del vector gradiente de cada uno de los píxeles, se aplicará el método de Otsu para hallar el umbral óptimo con el que se comparará la magnitud del vector gradiente de cada píxel. En el caso de que dicha magnitud sea superior al umbral, el píxel se considerará como 'borde'; de lo contrario, se considerará que ese determinado píxel no pertenece a ningún borde. El método de Otsu se puede implementar con Matlab a partir de la función 'graythresh.m' (disponible en el *toolbox* 'images\images').

A diferencia de lo anterior, en este Proyecto se decidió utilizar el algoritmo de Canny, debido a las imprecisiones en la detección de bordes que conllevaba la utilización del operador de Sobel. De hecho, se podría afirmar que el algoritmo de Canny es uno de los métodos más precisos para la detección de bordes, gracias a su gran capacidad para detectar bordes tanto de mucha como de poca intensidad.

4.3.2.2.1.- Algoritmo de Canny

En 1986, Canny propuso un método para la detección de bordes, el cual se basaba en tres criterios, estos son:

- Un criterio de detección expresa el hecho de evitar la eliminación de bordes importantes y no suministrar falsos bordes.
- El criterio de localización establece que la distancia entre la posición real y la localizada del borde se debe minimizar.
- El criterio de una respuesta que integre las respuestas múltiples correspondientes a un único borde.

El algoritmo de Canny sigue una secuencia de tres grandes pasos:

1. Obtención del gradiente
2. Supresión no máxima
3. Histéresis de umbral

4.3.2.2.1.1.- Obtención del gradiente

En este paso se calcula la magnitud y la orientación del vector gradiente en cada uno de los píxeles del fotograma.

Para la obtención del gradiente, lo primero que se realiza es la aplicación de un filtro *gaussiano* a la imagen original con el objetivo de suavizar la imagen y tratar de eliminar el posible ruido existente. Sin embargo, se debe tener cuidado de no realizar un suavizado excesivo, pues se podrían perder detalles de la imagen y provocar un pésimo resultado final. Este suavizado se obtiene promediando los valores de intensidad de los píxeles en el entorno de vecindad con una máscara de convolución de media cero y desviación estándar σ. En la figura 4.6 se muestran dos ejemplos de máscaras que se pueden usar para realizar el filtrado *gaussiano*.

Una vez que se suaviza la imagen, para cada píxel se obtiene la magnitud y módulo (orientación) del gradiente, obteniendo así dos imágenes. El algoritmo para este primer paso se describe a continuación.

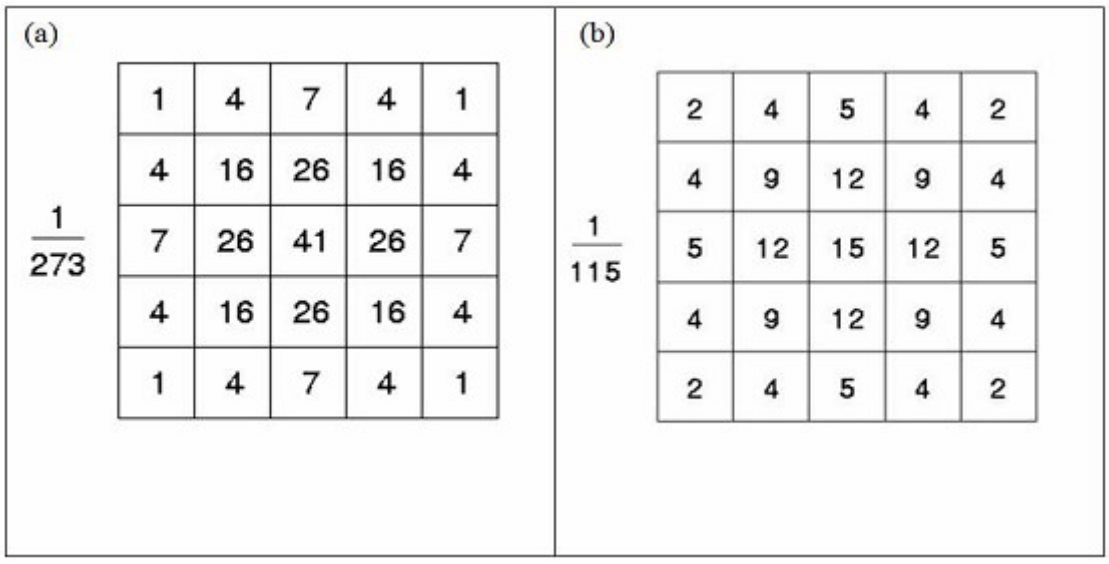

Figura 4.6: Máscaras de convolución recomendadas para obtener el filtro gaussiano. La máscara (a) fue obtenida de [26], mientras que la máscara (b) fue obtenida de [27]

Algoritmo: Obtención de Gradiente

Entrada: imagen *I*
máscara de convolución *H*, con media cero y desviación estándar σ .

Salida: imagen E_m de la magnitud del gradiente
imagen E_o de la orientación del gradiente

1. Suavizar la imagen *I* con *H* mediante un filtro gaussiano y obtener *J* como imagen de salida.

2. Para cada píxel *(i, j)* en *J*, obtener la magnitud y orientación del gradiente basándose en las siguientes expresiones.

Tal y como se muestra a continuación, los elementos de la máscara de convolución se calculan mediante la siguiente ecuación:

$$G_\sigma(i,j) = c \cdot e^{\frac{-(i^2+j^2)}{2\sigma^2}} \tag{4.6}$$

donde:

i,j: Fila y columna de cada elemento de la máscara considerando que el origen se encuentra en el centro de dicha máscara.

c: Constante por la que se debe multiplicar cada elemento para que la suma de todos ellos sea igual a 1.

σ: Desviación estándar del filtro cuyo valor está comprendido entre 0 y 3 (cuánto más cercano a 3 sea el valor de *σ*, más suavizada quedará la imagen original).

Un ejemplo de máscara de convolución de 5x5 elementos sería el siguiente (se considera *σ*=1):

$$G_\sigma(i,j) = \frac{1}{6.1689} \cdot \begin{bmatrix} 0.0183 & 0.0821 & 0.1353 & 0.0821 & 0.0183 \\ 0.0821 & 0.3679 & 0.6065 & 0.3679 & 0.0821 \\ 0.1353 & 0.6065 & 1.0000 & 0.6065 & 0.1353 \\ 0.0821 & 0.3679 & 0.6065 & 0.3679 & 0.0821 \\ 0.0183 & 0.0821 & 0.1353 & 0.0821 & 0.0183 \end{bmatrix} \tag{4.7}$$

Una vez se suaviza la imagen, para cada píxel se obtiene la magnitud y la orientación del vector gradiente.

El gradiente de una imagen *f(x, y)* en un punto *(x, y)* se define como un vector bidimensional dado por la ecuación:

$$G[f(x,y)] = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \frac{\partial}{\partial_x} f(x,y) \\ \frac{\partial}{\partial_y} f(x,y) \end{bmatrix} \tag{4.8}$$

siendo un vector perpendicular al borde que apunta en la dirección de variación máxima intensidad, *f,* en el punto *(x, y)* por unidad de distancia. Su magnitud $|G[f(x,y)]|$ y dirección $\phi(x,y)$ vienen dadas por:

$$|G| = \sqrt{{G_x}^2 + {G_y}^2} = |G_x| + |G_y|$$

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

(4.9)

Cuanto mayor sea esta variación de intensidad, mayor será la magnitud del vector gradiente.

3. Obtener E_m a partir de la magnitud de gradiente y E_o a partir de la orientación, de acuerdo a las expresiones anteriores.

4.3.2.2.1.2.- Supresión no máxima

En este segundo paso se realiza una supresión de los píxeles que no presentan una máxima variación de intensidad (pequeña magnitud del vector gradiente).

Las dos imágenes generadas en el paso anterior sirven de entrada para generar una imagen con los bordes adelgazados. El procedimiento es el siguiente:

Se consideran cuatro direcciones identificadas por las orientaciones de 0º, 45º, 90º y 135º con respecto al eje horizontal y a cada píxel se le asigna una de estas direcciones, la que más se aproxime a la dirección del ángulo de gradiente calculado anteriormente.

A continuación, se compara la magnitud del vector gradiente en la dirección asignada con la de al menos uno de sus dos píxeles vecinos en esa dirección. En el caso de que esta magnitud sea menor, se le asigna un 0 de magnitud a dicho píxel; de lo contrario, se le asigna al píxel el valor que tenga la magnitud del vector gradiente.

Siguiendo este mismo procedimiento para cada uno de los píxeles, se consigue suprimir los píxeles que no presentan una elevada variación de intensidad y se obtienen los píxeles que se podrían considerar como posible 'borde'.

La salida de este segundo paso es la imagen I_n con los bordes adelgazados, es decir, $E_m\ (i,j)$ después de la supresión no máxima de puntos de borde.

Un dato importante a añadir sería la introducción de ruido durante este procedimiento, el cual podría ocasionar una confusión en la asignación de valores de magnitud del vector gradiente de cada uno de los píxeles. De ser así, los píxeles con valor de magnitud 0 podrían pasar a tener un valor diferente y a considerarse como 'borde'.

Algoritmo: Supresión no máxima

Entrada: imagen E_m de la magnitud del gradiente
imagen E_o de la orientación del gradiente

Salida: imagen I_n

Considerar: cuatro direcciones d_1, d_2, d_3, d_4 identificadas por las direcciones de 0º, 45º, 90º y 135º con respecto al eje horizontal.

1. Para cada píxel (i, j):

1.1. Encontrar la dirección d_k que mejor se aproxima a la dirección $E_o\ (i, j)$, que viene a ser la perpendicular al borde.

1.2. Si $E_m\ (i, j)$ es más pequeño que al menos uno de sus dos vecinos en la dirección d_k , al píxel (i, j) de I_n se le asigna el valor 0, $I_n(i, j)=0$ (supresión), de otro modo $I_n(i,j)=E_m(i,j)$.

2. Devolver I_n

4.3.2.2.1.3.- Histéresis de umbral

La imagen obtenida en el paso anterior suele contener máximos locales creados por el ruido.

Por tanto, un último paso en la aplicación del algoritmo de Canny es la realización de una histéresis de umbral con el objetivo de eliminar el posible ruido detectado en el paso anterior.

El proceso consiste en tomar la imagen obtenida del paso anterior, tomar la orientación de los puntos de borde de la imagen y tomar dos umbrales, el primero más pequeño que el segundo.

Para cada pixel de la imagen (con valor diferente de 0) se observa si la magnitud del vector gradiente es superior al segundo umbral; de ser así, se asignará un 1 a dicho píxel. De lo contrario, se tendrá que comparar si ese valor de magnitud es superior al primer umbral. En el caso de que sea superior, también se asignará un 1 a ese píxel, pero si se trata de un valor inferior, se le asignará un 0 (es decir, por debajo del primer umbral no se detectará ningún píxel como 'borde').

A partir de dicho punto, seguirán las cadenas de máximos locales conectados en ambas direcciones perpendiculares a la normal del borde, siempre que sean mayores al primer umbral. Así se marcan todos los puntos explorados y se almacena la lista de todos los puntos en el contorno conectado. Es así como en este paso se logra eliminar las uniones en forma de Y de los segmentos que confluyan en un punto.

Como resultado, se obtiene una imagen binaria (salida de la función) que en adelante se representará como una matriz.

Algoritmo: Histéresis de umbral a la supresión no máxima

Entrada: imagen I_n obtenida del paso anterior
imagen E_o de la orientación del gradiente
umbral t_1
umbral t_2, donde $t_1 > t_2$

Salida: imagen *G* con los bordes conectados de contornos

1. Para todos los puntos de I_n y explorando I_n en orden fijo:

1.1. Localizar el siguiente punto de borde no explorado previamente, $I_n(i,j)$, tal que $I_n(i,j) > t_2$

1.2. Comenzar a partir de $I_n(i,j)$, seguir las cadenas de máximos locales conectados en ambas direcciones perpendiculares a la normal de borde, siempre que $I_n > t_2$

1.3. Marcar todos los puntos explorados y, salvar la lista de todos los puntos en el entorno conectado encontrado.

2. Devolver G formada por el conjunto de bordes conectados de contornos de la imagen, así como la magnitud y orientación, describiendo las propiedades de los puntos de borde.

4.3.2.2.1.4.- Resultados

A continuación se muestra un ejemplo visual de aplicar el algoritmo de *Canny* a una imagen.

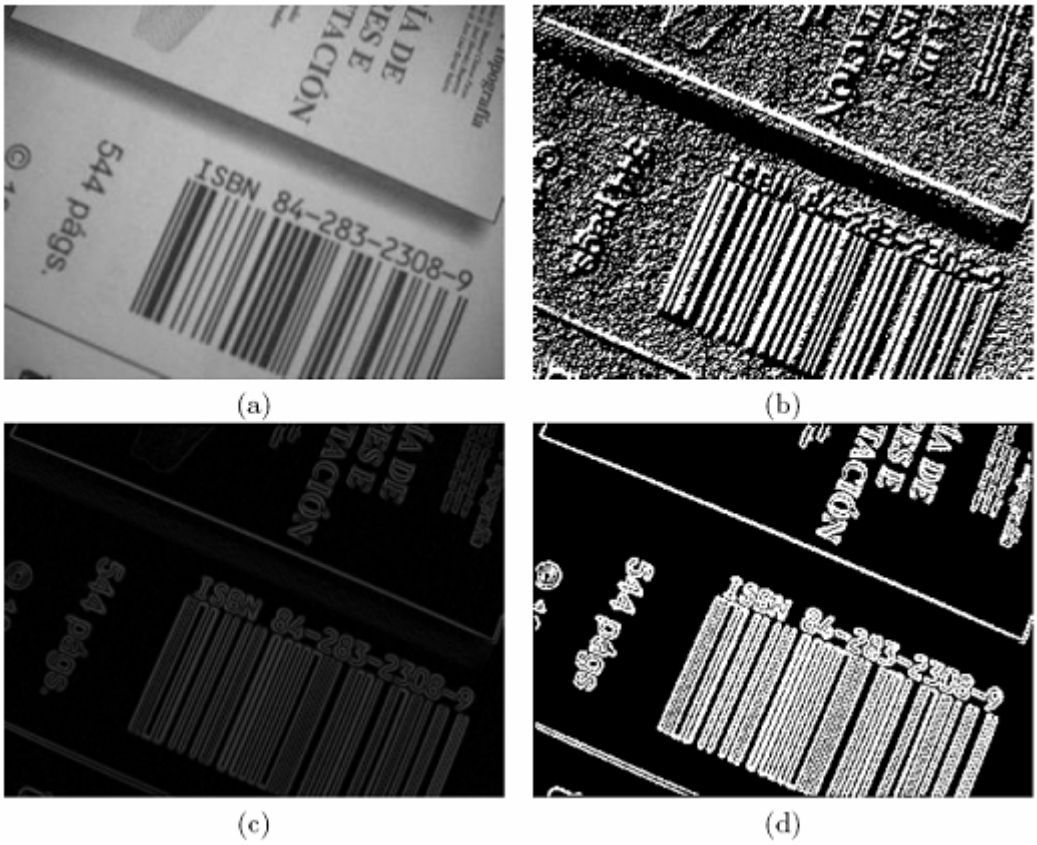

Figura 4.7: Resultado de aplicar el detector de bordes de Canny: (a) imagen original; (b) orientación; (c) supresión no máxima; (d) histéresis de umbral.

4.3.2.2.2.- Problemas en los operadores para la detección de bordes

Existen ciertos problemas comunes en todos los operadores que utilizan el gradiente para la detección de bordes.

- Se deben realizar elecciones en valores umbral (corte) y tamaño de las máscaras a utilizar (el tamaño condiciona el grado de suavizado, el cual puede afectar a las

detecciones por cruce por cero y al máximo gradiente sobre una imagen). La posición estimada de un borde debería ser independiente del tamaño de la máscara de convolución.

- Las esquinas son a menudo omitidas a causa de que el gradiente (*1D*) sobre las esquinas es normalmente pequeño. Esto puede causar considerables dificultades para el etiquetado de líneas ya que éstas pueden aparecer discontinuas.

- Los operadores de primera derivada detectan solamente *step-like*. Si uno quiere encontrar líneas se necesita utilizar operadores diferentes. (por ejemplo Canny).

- Procesos diferenciales aplicados en la detección de bordes generan falsos positivos y falsos negativos.

4.3.2.3.- Técnicas basadas en el Laplaciano

El Laplaciano es la segunda derivada de una función y representa la derivada de esta respecto a todas las direcciones, y esta dado por:

$$\nabla^2 f(x,y) = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \tag{4.10}$$

Generalmente para el Laplaciano se utilizan las máscaras mostradas en la figura 4.8. Nótese que el píxel central toma el valor negativo de la suma de todos los que lo rodean, de tal forma que la suma aritmética de todos los píxeles sea cero.

-1	-1	-1
-1	8	-1
-1	-1	-1

0	-1	0
-1	0	-1
0	-1	0

Figura 4.8: Máscaras utilizadas para el operador Laplaciano

4.3.3.- Transformada de Hough

La Transformada de *Hough* es un algoritmo empleado en reconocimiento de patrones en imágenes que permite encontrar ciertas formas (líneas, círculos o curvas) dentro de una imagen a la que previamente se le ha aplicado algún detector de bordes, quedando la imagen binaria. La versión más simple consiste en encontrar líneas, que es de lo que se encarga este programa.

Su modo de operación es principalmente estadístico y consiste en que para cada punto que se desea averiguar si es parte de una línea se aplica una operación dentro de cierto rango, con lo que se averiguan las posibles líneas de las que puede ser parte el punto. Esto se continúa para todas los puntos en la imagen, al final se determina qué líneas fueron las que más puntos posibles tuvieron y esas son las líneas en la imagen.

4.3.3.1.- Detección de líneas rectas

Considerando un píxel de coordenadas (x_i, y_i), seleccionado como elemento que representa un borde; sobre éste pasarán infinitas rectas que lo contienen:

$$y_i = ax_i + b \tag{4.11}$$

El modelo de todas estas posibilidades quedan definidas por los infinitos valores de *a* (pendiente) y *b* (ordenada al origen). Examinando el espacio paramétrico de *a* y *b* (plano *ab*) se puede observar que un punto (a_i, b_i) representa una determinada recta en el plano xy:

$$b = y_i - ax_i \tag{4.12}$$

Por lo que para cada punto (x_i, y_i) perteneciente a una recta se obtiene otra recta en el espacio paramétrico *ab*.

Todas éstas se intersecan en un punto (*a'*, *b'*). Dicho punto es la representación de la recta en el plano *xy* que contiene todos los puntos (x_i, y_i) colineales (que tienen pendiente *a'* y ordenada al origen *b'*).

El problema que surge de esta transformación es para aquellos casos donde la recta en el plano ab es vertical, ya que la pendiente de la recta es infinita.

Para solucionar este problema *Hough* propuso un método alternativo que consiste en expresar la ecuación de la recta en coordenadas polares. Ésta es la llamada '*Transformada de Hough'*.

Al variar el parámetro *a* desde -∞ a +∞ se obtendrá los infinitos valores de b. La representación geométrica de $b = y_i - ax_i$, en el espacio paramétrico, será una recta. La característica interesante de esta método consiste que si dos píxeles que pertenezcan a una misma línea son representados en el espacio paramétrico, el modelo de la recta está definido en la intersección de las dos rectas del espacio paramétrico.

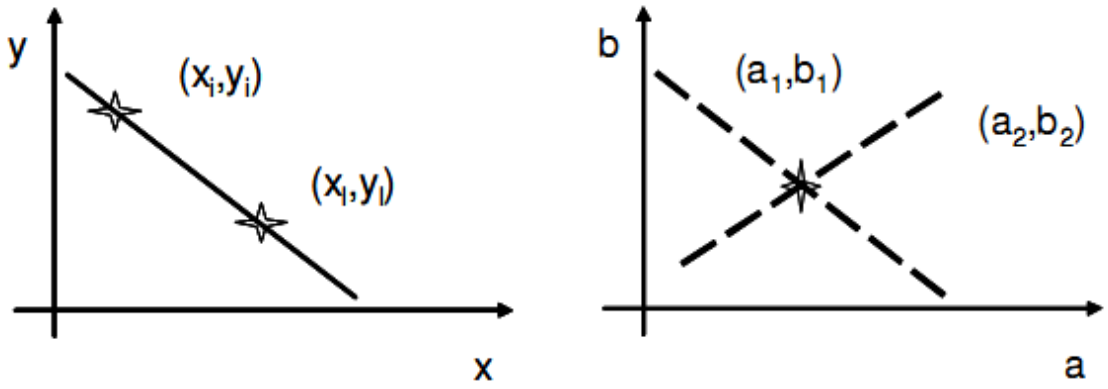

Figura 4.9: Ejemplo de conversión hacia el espacio paramétrico

La transformada de *Hough* (1962) aplica este concepto para la localización de líneas rectas en la imagen. Discretiza el espacio paramétrico en intervalos de [*amin, amax*] y [*bmin, bmax*], creando una rejilla de celdas de acumulación. Por cada píxel, considerado como borde, se hace recorrer el rango dinámico de a obteniendo los valores de b. Por cada valor de a y b se le pone un voto en la celda correspondiente. Esta operación se hace con todos los píxeles etiquetados como bordes. Al finalizar, aquellas celdas con más votos indicarán la presencia de rectas en la imagen, cuyos modelos corresponderán con las coordenadas de la celda.

Sin embargo, el espacio paramétrico elegido no es el más correcto, ya que los rangos dinámicos de a y b no están limitados. En cambio, si se hace una representación en coordenadas polares, el ángulo de la normal de la recta, θ, está limitado al rango de $[0\ \pi]$:

$$x_i \cos\theta + y_i \sin\theta = \xi \qquad (4.13)$$

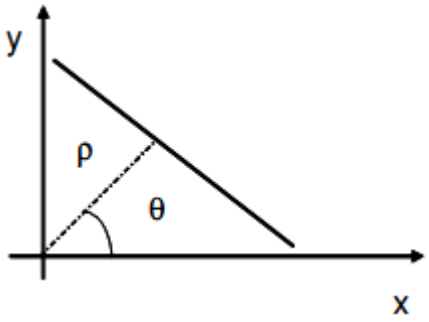

Figura 4.10: Espacio paramétrico en coordenadas polares

La transformada de *Hough* emplea una representación paramétrica de formas geométricas. Una recta, por ejemplo se representa por un módulo *ρ(rho)* (perpendicular a la recta y que pasa por el origen (0,0) y un ángulo *θ(theta)* (formado por el módulo y el eje positivo de las *x's*).

La ventaja de este método es que evita singularidades, como por ejemplo rectas de pendiente infinita. Si se representa ρ y θ en un plano cartesiano, una recta queda determinada mediante un punto con coordenadas (*rho(recta), theta(recta)*), mientras que un punto, se representa como una función senoidal. Si por ejemplo tenemos dos puntos, tendremos dos senoides desfasadas alfa grados dependiendo de las coordenadas de los puntos. Dichas senoides se irán cruzando cada 180º. La interpretación geométrica de este hecho, es que la función seno de cada punto, representa las infinitas rectas que pasan por cada punto, cuando dos puntos comparten la misma recta, sus representaciones senoidales se cruzan, se obtiene un punto. Cada vez que se da media vuelta ($\theta = 180º$) se vuelve a repetir la misma recta, por lo que volvemos a obtener otro punto, que de hecho es la misma recta.

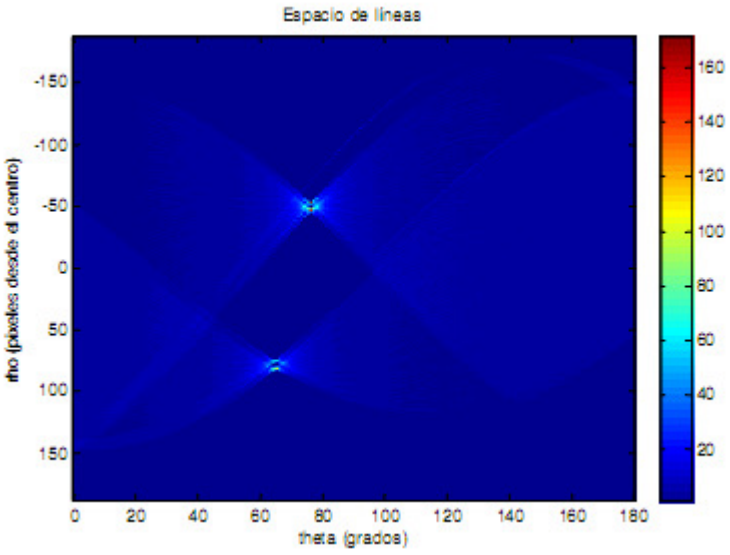

Figura 4.11: Votaciones en el espacio paramétrico de coordenadas polares

Una forma de mejorar el coste computacional, es aprovechar el ángulo del gradiente de cada píxel, pues éste, aunque es de carácter local, dará una primera aproximación del ángulo de la normal. De esta forma, se disminuye el rango dinámico de θ y por tanto del número de operaciones y de votos en la rejilla de acumulación.

Una vez obtenida la matriz binaria (*BW*) en la función anterior, se aplicará la Transformada de *Hough* a dicha matriz para poder detectar las líneas principales que conforman el plano. A partir de la función ***'hough.m'***, se obtendrá la matriz *H* y los vectores ρ y θ.

La transformada de *Hough* está orientada a la detección de contornos cuya forma básica es conocida y que puede ser representada como una curva paramétrica, tales como líneas, círculos, cónicas, etc.

Consideremos el caso de una línea recta que es el que nos interesa en este caso. Se tienen varios puntos (orillas) que tienen una alta probabilidad de pertenecer a una línea, pero existen algunas orillas faltantes y otras fuera de línea. El objetivo es encontrar la ecuación de la línea que "mejor" explique los puntos existentes, ver figura 4.12.

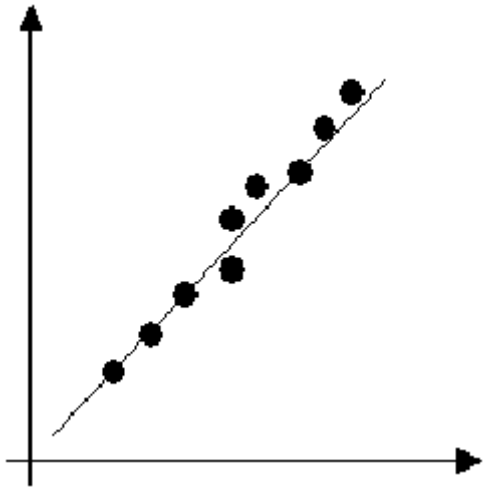

Figura 4.12: Detección de líneas. Se ilustra la recta que mejor aproxima el borde descrito por los puntos que representan orillas.

A continuación, se explicará de forma muy analítica los pasos que sigue la Transformada de *Hough* mediante el siguiente ejemplo:

Supóngase la siguiente **matriz binaria** (*BW*) de la figura 4.13, donde los píxeles de interés se encuentran en las siguientes coordenadas (X, Y): (5,1), (6,2), (7,3), (8,4), (9,5).

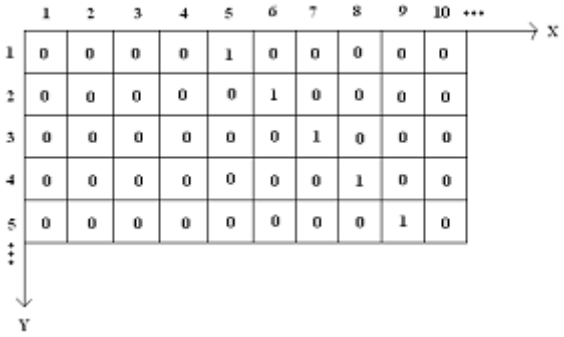

Figura 4.13: Matriz binaria (BW)

Para representar la línea que definen estos píxeles de valor 1, se considera la ecuación de una recta a partir de los parámetros ρ y θ (coordenadas polares):

$$y = \left(\frac{-\cos\theta}{\sin\theta}\right)x + \left(\frac{\rho}{\sin\theta}\right) \tag{4.14}$$

Simplificando la ecuación,

$$\rho(\theta) = x \cdot \cos\theta + y \cdot \sin\theta \tag{4.15}$$

$\theta \in$ [-90, 90] en grados

Tal y como se muestra en la siguiente figura (Fig. 4.14), θ se corresponde con la dirección (medida con respecto al eje x) en la que se evaluará cada uno de los píxeles de interés (para cada píxel se trazará una recta que corte perpendicularmente con esa dirección) y ρ con el módulo del vector o de cada uno de los vectores que se formen en esa dirección θ (distancia desde el origen de coordenadas a la recta que atraviesa el píxel de interés). Un dato importante a tener en cuenta sería la ubicación de este origen de coordenadas, ya que dependiendo de donde se sitúe, se obtendrán diferentes resultados para los valores de ρ y θ.

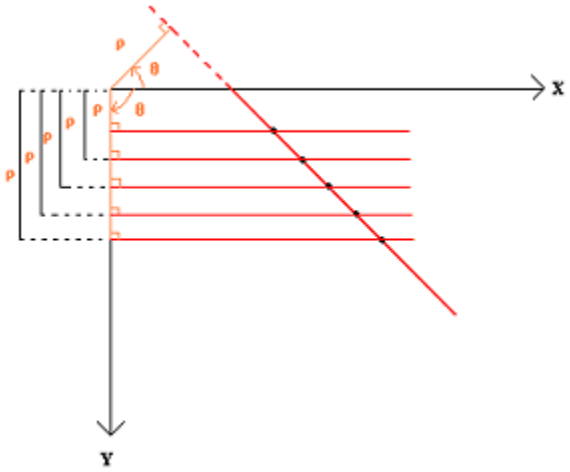

Figura 4.14: Representación de la línea

En este caso, se considera que el origen se encuentra en la parte superior izquierda de la matriz binaria (BW).

Partiendo de la ecuación de la recta en coordenadas polares ($\rho(\theta) = x \cdot \cos\theta + y \cdot \sin\theta$), para cada θ se calcula la ρ correspondiente de solamente los píxeles de interés (los píxeles de interés son los que están a '1' en la matriz binaria BW) y se almacenan los resultados en un vector (Fig. 4.15).

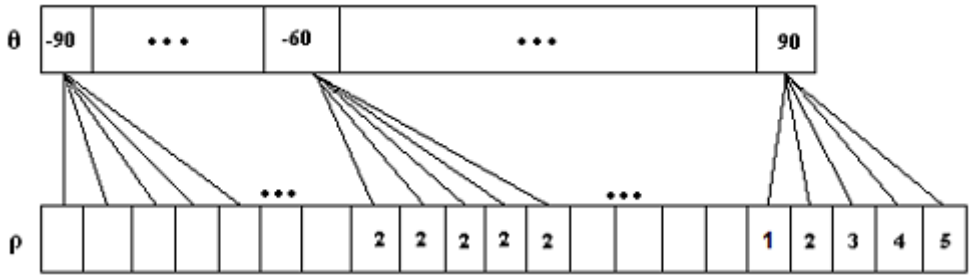

Figura 4.15: Vectores ϑ y ρ

En este ejemplo, la matriz binaria (BW) sólo consta de 5 píxeles con valor '1'; por lo tanto, para cada ángulo θ sólo se obtendrán 5 posibles valores de ρ.

En cuanto a estos valores de ρ, se observa que para θ=-60º, la ρ es la misma para los 5 píxeles; es decir, los 5 píxeles están alineados y, por lo tanto, eso significa que se ha encontrado una línea principal. A diferencia de eso, para θ=90º no hay ningún valor de ρ igual y, por lo tanto, se deduce que los píxeles de interés no se encuentran alineados horizontalmente.

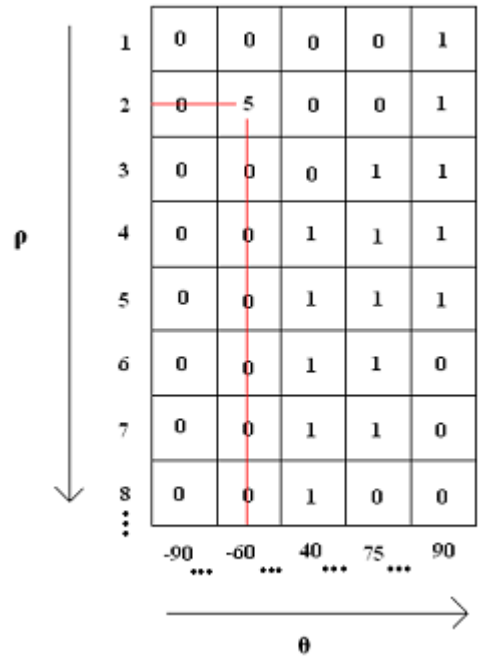

Figura 4.16: Matriz *H*

A partir de los vectores θ y ρ de la Fig. 4.15 se rellena la matriz *H* (Fig. 4.16) que devuelve esta función, de manera que el valor de cada una de las posiciones de esta matriz se corresponde con el número de píxeles que comparten un determinado valor de ρ, alineados en la dirección perpendicular a θ. Por ejemplo, para el caso anterior, se obtendría lo siguiente:

La línea definida por θ=-60º y ρ=2 atraviesa 5 píxeles de interés. El resto de líneas posibles tan solo atraviesan 1 píxel o ninguno.

En general, los valores máximos de la matriz *H* se corresponden con las líneas detectadas. En este ejemplo se puede afirmar que se ha detectado una única línea.

Para acabar de ilustrar la técnica de *Hough* para la detección de líneas, se presenta la siguiente imagen binaria (Fig. 4.11) de dimensiones conocidas:

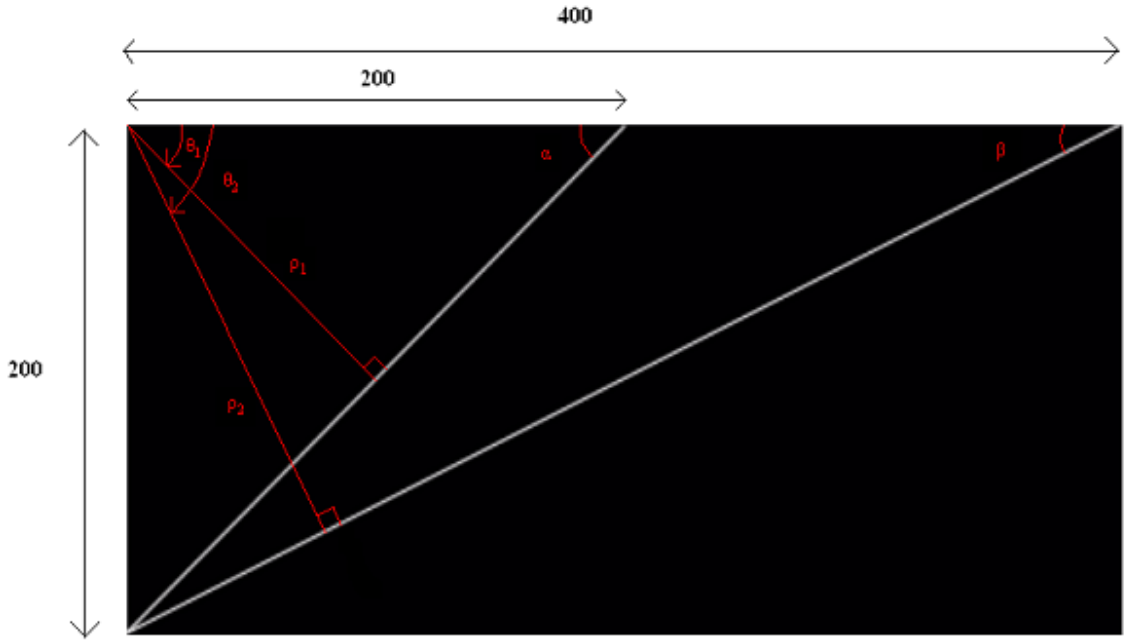

Figura 4.17: Imagen binaria

Previamente se realizarán los cálculos reales de θ y ρ de cada una de las líneas, con el objetivo de verificar que los resultados que se obtengan al aplicar la *Transformada de Hough* sean los correctos. Los cálculos se muestran en las siguientes tablas:

α	θ_1	ρ_1
$\arctan\left(\frac{200}{200}\right) = 45^{\circ}$	$90^{\circ} - \alpha = 45^{\circ}$	$200 \cdot \sin\alpha = 141{,}42$

Tabla 4.2: Cálculo de los ángulos α, ϑ_1 y ρ_1

β	θ_2	ρ_2
$\arctan\left(\frac{200}{400}\right) = 26{,}565^{\circ}$	$90^{\circ} - \beta = 63{,}43^{\circ}$	$400 \cdot \sin\beta = 178{,}88$

Tabla 4.3: Cálculo de los ángulos θ, ϑ_2 y ρ_2

Una vez realizados estos cálculos trigonométricos, se aplicará la *Transformada de Hough* a la imagen binaria y se representará con Matlab la matriz *H* obtenida. Los resultados se muestran en la Fig. 4.18 y 4.19:

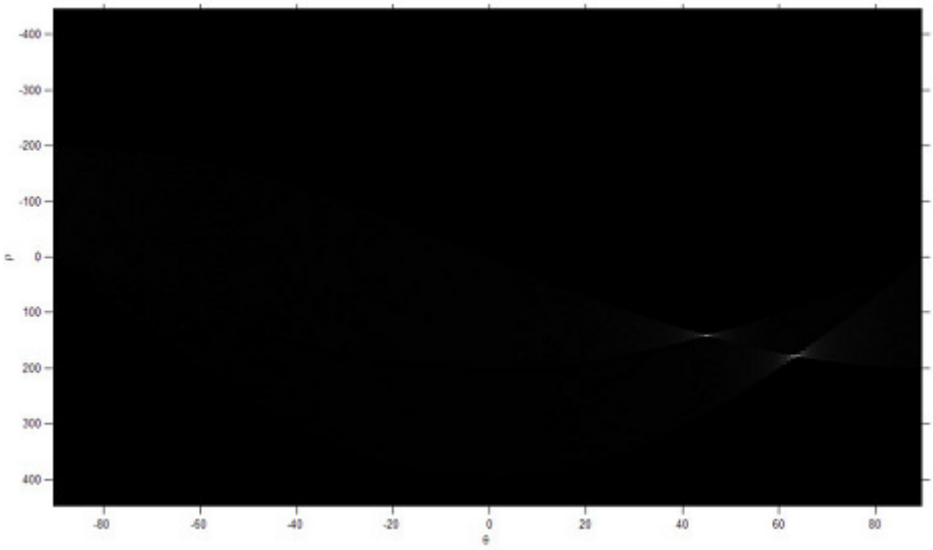

Figura 4.18: Representación gráfica de la matriz H

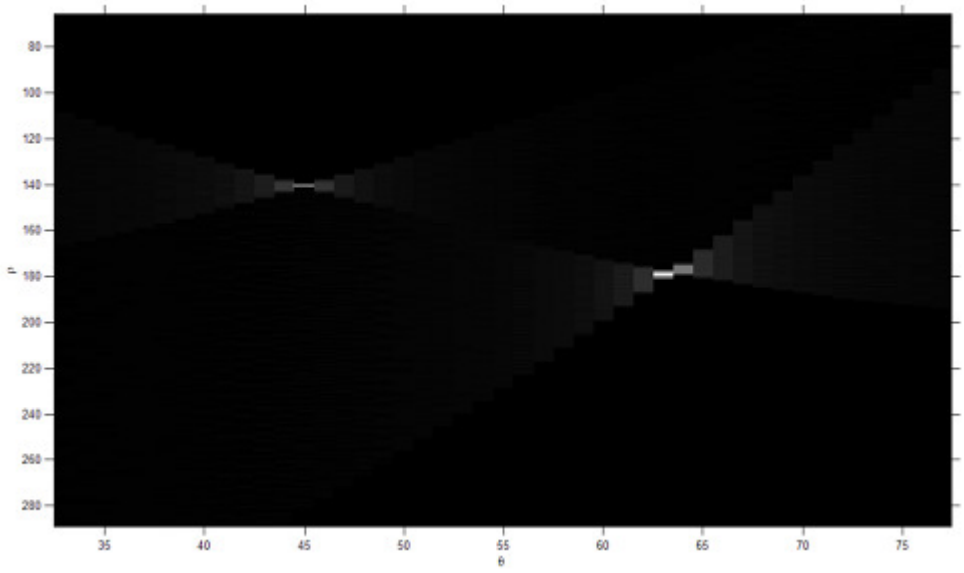

Figura 4.19: Máximos de la matriz H (ampliación de la Fig. 4.16)

Los resultados obtenidos se aproximan mucho a los cálculos trigonométricos realizados: encontramos un máximo en θ_1=45º y ρ_1=140,31 y un segundo máximo en θ_2=63º y ρ_2=179,40. Se podría comentar que estas pequeñas diferencias son debidas a la falta de precisión en la realización del trazado de las dos líneas, ya que éste se realizó manualmente.

Tal y como se ha comentado anteriormente, los valores máximos de la matriz *H* indican que en esas posiciones de la matriz se han detectado líneas principales, cuyo número de píxeles es igual al valor encontrado en dicha posición. En la representación gráfica adquieren una elevada intensidad, debido a la concentración de píxeles en esas coordenadas.

A partir de la función ***'houghpeaks.m'*** (disponible también en el toolbox 'images\images' de Matlab) se hallarán las coordenadas de los máximos de la matriz *H* detectados (Fig. 4.20). Para asegurarse de que sólo se detecten líneas principales, en esta misma función se fijará un umbral de detección de los máximos, de manera que sólo se considerarán máximos de *Hough* los valores de la matriz *H* que sean iguales o superiores a dicho umbral (el resto de valores harán referencia a líneas de pocos píxeles, líneas no principales de la imagen binaria). Además, mediante esta función también se podrá especificar el número máximo de máximos de *Hough* que se desea detectar.

```
P = houghpeaks(H,20,'threshold',ceil(0.3*max(H(:))));
[a, b] = size(P);
x = T(P(:,2)); y = R(P(:,1));
plot(x,y,'s','color','red');
```

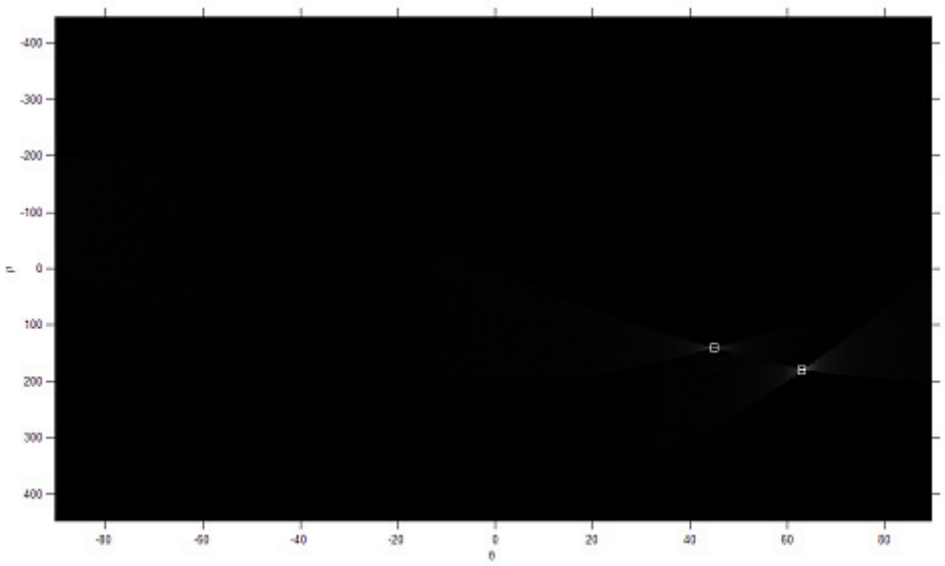

Figura 4.20: Detección de las coordenadas de los máximos de la matriz H

Una vez detectadas las coordenadas de los máximos de la matriz transformada de *Hough*, a partir de la función ***'houghlines.m'*** (disponible en el toolbox 'images\images' de Matlab) se representarán las líneas principales (Fig. 4.21).

Para ello, a esta función se le pasarán las coordenadas de los máximos como parámetro de entrada y, de este modo, se obtendrá como resultado un vector de líneas. Además, como parámetros de entrada, también se puede especificar la mínima longitud de la línea encontrada (despreciando las líneas que se encuentren por debajo de ese valor) y la mínima separación entre líneas, de tal manera que si se detectan dos líneas separadas una distancia menor que el valor especificado, tan solo se representará una única línea.

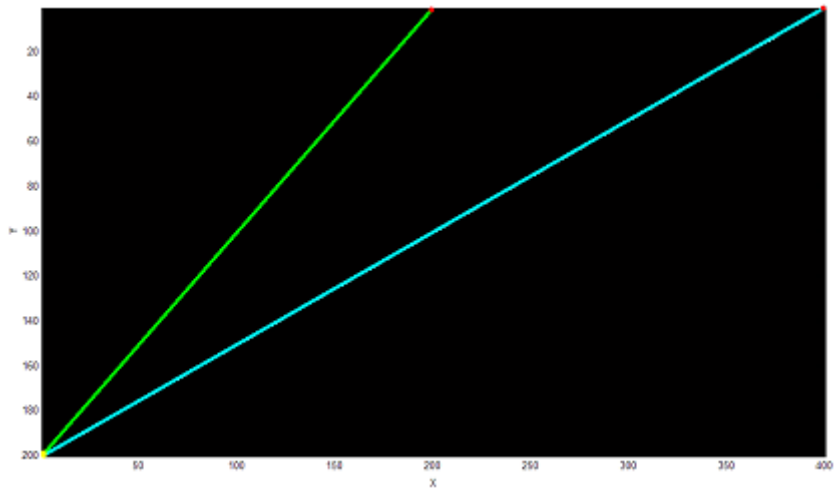

Figura 4.21: Líneas detectadas

4.3.3.2.- Algoritmo de Hough

Sea E una imagen binaria de tamaño *MxN*, en la cual cada píxel de la imagen es uno o cero. Sean ρ_d, θ_d vectores que contienen valores discretos del espacio paramétrico ρ, θ ($\rho \in [0, (M^2 + N^2)^{1/2}]$ y $\theta \in [0, 180]$). Donde R y T son el número de elementos de ρ y θ respectivamente. Ver [24].

1) Discretize el espacio parámetro ρ, θ.

2) Sea $A(R, T)$ una matriz de contadores inicializada en 0.

3) Para cada píxel $E(i, j)$ que pertenezca a la línea desde h = 1, 2,…T con k =1, 2, …,K.

a) Hacer $\rho = i \cos \theta_d(h) + j \sin \theta_d(h)$.

b) Halle el intervalo K que sea más cercano al ρ obtenido.

c) Incremente matriz de contadores $A(k, h)$.

4) Encuentre el máximo local (k_p, h_p) tal que sea mayor que un umbral.

5) Con el ρ y θ encontrados se describe la línea en forma polar.

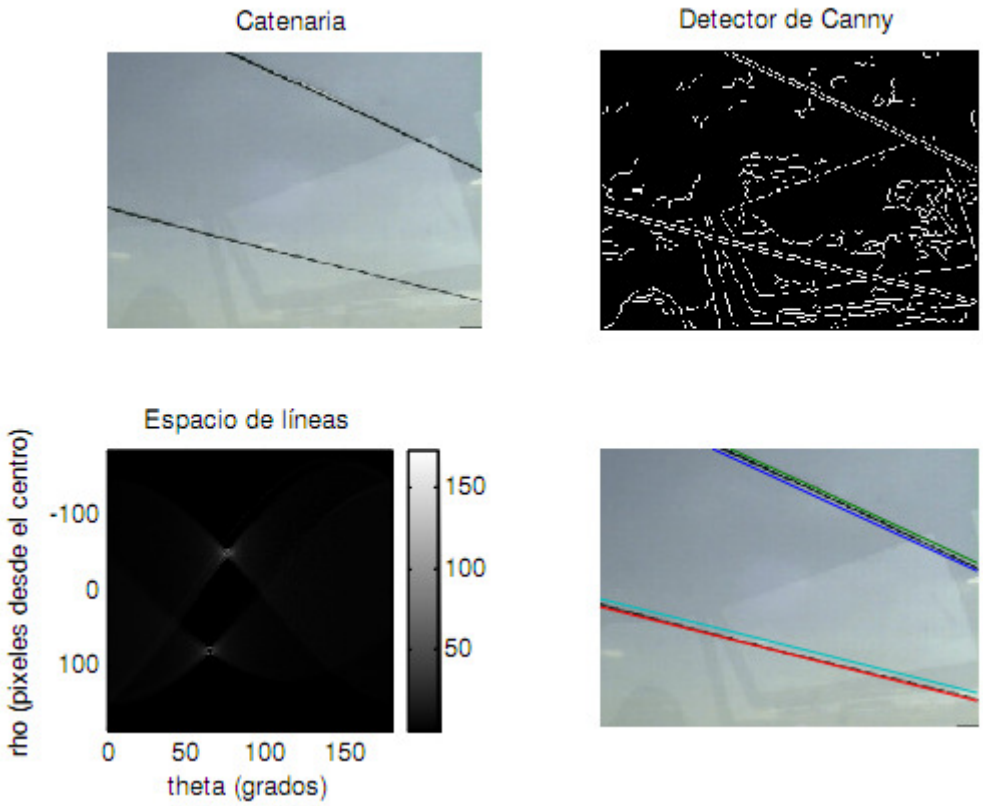

Figura 4.22: Procesamiento de imagen para detectar líneas rectas