



**Universidad Veracruzana**

Maestría en Inteligencia Artificial

**Visión por Computadora**

**Tarea 4. Implementación del operador de Sobel  
para detección de bordes en Julia y Matlab.**

*Ángel García Báez*

Profesor: Dr. Héctor Acosta Mesa

March 22, 2025

## **Contents**

<b>1</b>	<b>Objetivo de la práctica</b>	<b>2</b>
<b>2</b>	<b>Metodología</b>	<b>3</b>
<b>3</b>	<b>Resultados</b>	<b>5</b>
3.1	Resultados para G <sub>x</sub> . . . . .	5
3.2	Resultados para G <sub>y</sub> . . . . .	6
3.3	Resultados para el gradiente G . . . . .	7
3.4	Resultados para el angulo del gradiente . . . . .	8
3.5	Resultados de la función quiver . . . . .	9
<b>4</b>	<b>Conclusiones</b>	<b>10</b>
<b>5</b>	<b>Referencias</b>	<b>11</b>
<b>6</b>	<b>Anexos</b>	<b>12</b>
6.1	Implementación de la ecualización por histograma . . . . .	12
6.2	Implementación del operador de Sobel en Matlab . . . . .	15

## 1 Objetivo de la práctica

Se tiene la siguiente imagen en escala de grises:

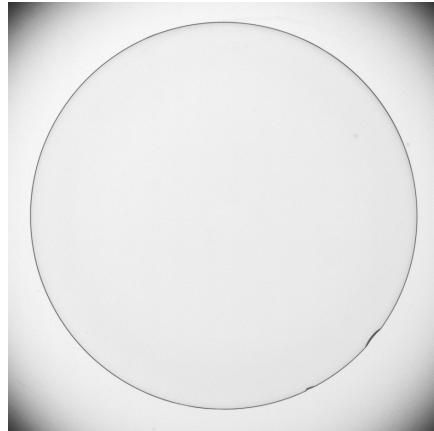


Figure 1: Figura 3.45(a). Extraída del libro Digital Image Processing

El objetivo de la presente practica es implementar un algoritmo que aplique el operador de Sobel sobre la imagen para la correcta detección de bordes en la imagen.

Para ello, se pide que:

1. Escribir un programa que haga el de los vectores gradientes usando el operador de Sobel sobre la figura 3.45.
2. Mostrar el resultado de del computo sobre  $G_x$ ,  $G_y$ ,  $\nabla f$  y  $\Theta$ .
3. Mostrar los vectores como un flujo óptico de flechas perpendiculares al eje usando el comando de matlab  $quiver(Gx, Gy)$ .
4. Explique brevemente los resultados.

## 2 Metodología

Para la correcta implementación del operador de Sobel es necesario definirlo en sus componentes más básicas, por lo que se recurrió al uso de las definiciones que vienen Gonzalez and Woods (2018):

El gradiente (primera derivada) de la una imagen, puede ser visto como una derivada bidimensional que tiene la siguiente forma:

$$\nabla f \equiv \text{grad}(f) = \begin{bmatrix} g_x \\ g_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}$$

Se observa que para la construcción del gradiente  $\nabla f$  se necesitan obtener los correspondientes  $g_x$  y  $g_y$  que corresponden a las derivadas parciales en  $x$  y en  $y$ .

Dado que los pixeles en una imagen no siguen el comportamiento de una función continua como tal, bajo ciertos arreglos y simplificaciones de muestreos locales, se puede construir un operador que calcula la derivada localmente mediante una mascara de 3x3 para la región que esta siendo muestreada bajo la siguiente estructura:

$$\begin{bmatrix} z_1 & z_2 & z_3 \\ z_4 & z_5 & z_6 \\ z_7 & z_8 & z_9 \end{bmatrix}$$

Haciendo dichas simplificaciones, es posible obtener la derivada parcial en  $x$  como sigue:

$$g_x = \frac{\partial f}{\partial x} = (z_7 + 2z_8 + z_9) - (z_1 + 2z_2 + z_3)$$

Al visualizar los coeficientes como matriz, salta a la vista el objetivo el filtro, detectar los cambios en la derivada en la componente  $x$ :

$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Por otro lado, se tiene la correspondiente operación para el calculo de la derivada parcial en  $y$  como sigue:

$$g_y = \frac{\partial f}{\partial y} = (z_3 + 2z_6 + z_9) - (z_1 + 2z_4 + z_7)$$

Al visualizar los coeficientes como matriz, salta a la vista el objetivo el filtro, detectar los cambios en la derivada en la componente  $y$ :

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

Una vez que se cuentan con los elementos primarios se puede proceder con la construcción del gradiente. Cabe mencionar que el gradiente como tal, lo que hace es señalar la dirección de mayor cambio en la función en el punto  $(x, y)$ , por lo que se utiliza la norma del vector gradiente para conocer su norma.

$$M(x, y) = ||\nabla f|| = \text{mag}(\nabla f) = \sqrt{G_x^2 + G_y^2}$$

Finalmente, para calcular el angulo que se forma entre los vectores de las derivadas parciales  $G_x$  y  $G_y$  se utilizo la tangente inversa entre los vectores como se explica en Pérez and Valente (2018):

$$\theta(G_x, G_y) = \arctan(G_x, G_y)$$

Se realizo la implementación en Julia por su simpleza y velocidad en el procesamiento de operaciones con matrices así como con matlab para el uso de la función `quiver()`.

Cabe mencionar que para el caso del procesamiento de las imágenes como matrices en Julia, fue necesario aplicar una normalización a los valores producidos por el operador de Sobel para que se encuentre en un rango entre 0 y 1. Dicha normalización para el caso de los resultados obtenidos para  $G_x$ ,  $G_y$  y  $M(x, y)$  viene dada por:

$$\text{normalizacion}(M) = \frac{M - \min(M)}{\max(M) - \min(M)}$$

Donde  $M$  es la matriz de valores resultantes para los 3 casos.

Para el caso del angulo  $\theta$  se aplico la normalización de forma distinta debido a que los valores del angulo que devuelve Julia están dados en radianes:

$$\text{normalizacion}(\theta) = \frac{\theta + \pi}{2\pi}$$

Por ultimo, se hizo uso de a función ya implementada en matlab `quiver()` con las matrices de  $G_x$  y  $G_y$  obtenidas del proceso de aplicar el operador de Sobel.

### 3 Resultados

A continuación se muestran los resultados obtenidos por la implementación en Julia y en matlab

#### 3.1 Resultados para Gx

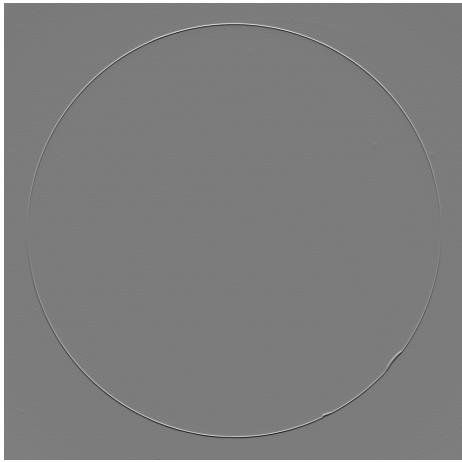


Figure 2: Gx en Julia.

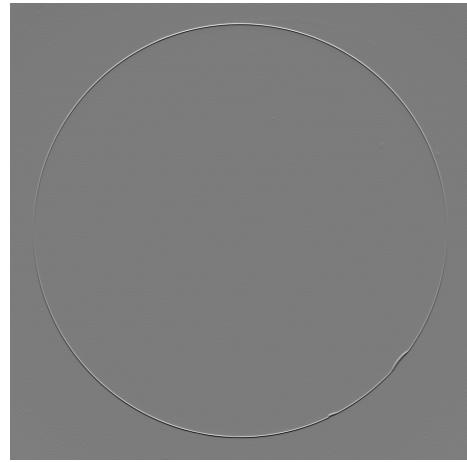


Figure 3: Gx en Matlab.

Figure 4: Comparación en Gx.

Se observa como ambos filtros logran detectar con mayor precisión los bordes que se generan en la curvatura del circulo que están en los extremos superiores e inferiores, dejando con una menor intensidad de detección la curvatura de los laterales y mostrando que el circulo presenta una pequeña deformidad en la parte inferior derecha.

Los resultados obtenidos entre ambas implementaciones son prácticamente el mismo.

### 3.2 Resultados para Gy

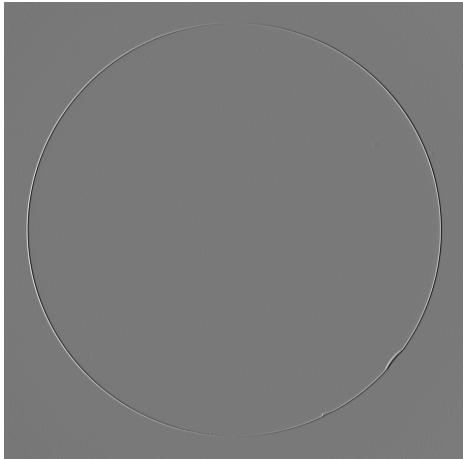


Figure 5: Gy en Julia.

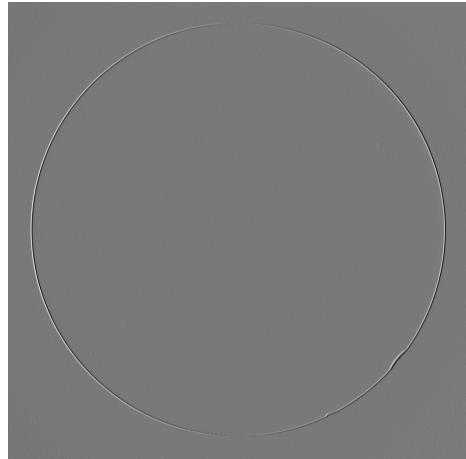


Figure 6: Gx en Matlab.

Figure 7: Comparación en Gy.

Se observa como ambos filtros detectan mejor la curvatura de los laterales del círculo, dándole menos intensidad a la curvatura superior e inferior.

Los resultados obtenidos entre ambas implementaciones son prácticamente el mismo.

### 3.3 Resultados para el gradiente G

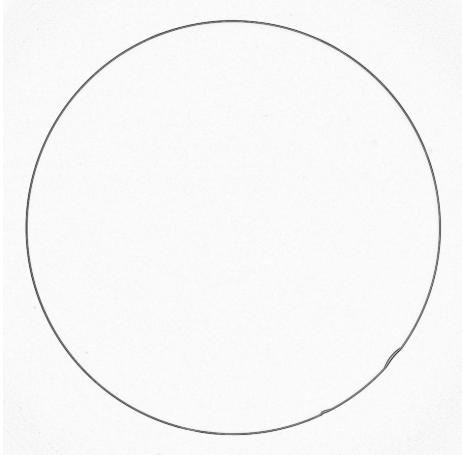


Figure 8: Gradiente en Julia.

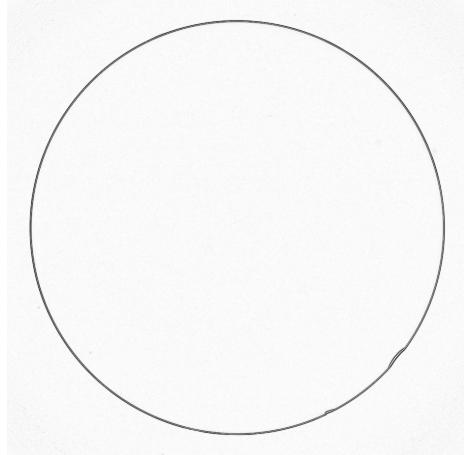


Figure 9: Gradiente en Matlab.

Figure 10: Comparación en G.

Lo primero que salta a la vista es como al combinar la información obtenida de  $G_x$  y de  $G_y$  para el gradiente de la imagen, la imagen mejora bastante al punto que resalta únicamente lo que detecta como borde, que en ambos casos es la completitud de la curvatura del círculo, eliminando a su paso todo aquello que no es borde según el operador.

### 3.4 Resultados para el angulo del gradiente

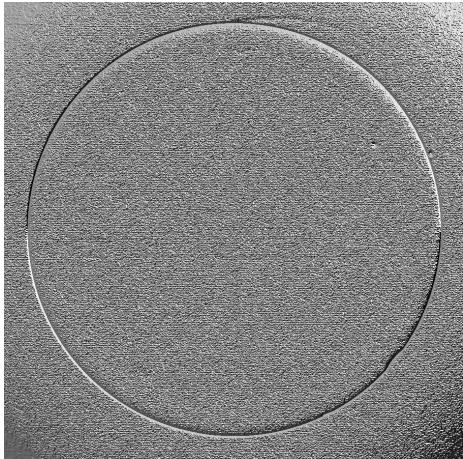


Figure 11: Angulo del gradiente en Julia.

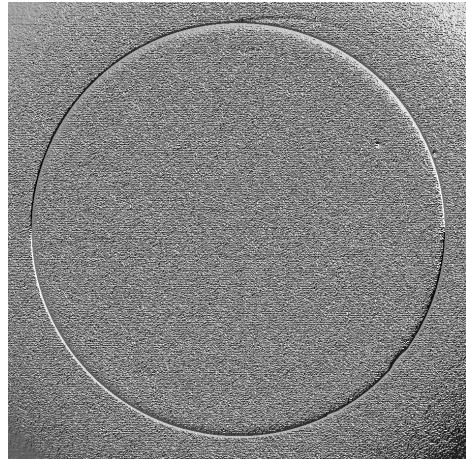


Figure 12: Angulo del gradiente en Matlab.

Figure 13: Comparación del Angulo del gradiente.

Se observa como la representación del angulo genera una imagen con lo que pareciera ser algún patrón de textura como si fuera tela estirada o papel aluminio doblado en donde resalta principalmente los bordes de la figura.

El resultado de graficar el angulo de los gradientes  $Gx$  y  $Gy$  es prácticamente el mismo en ambos casos.

### 3.5 Resultados de la función quiver

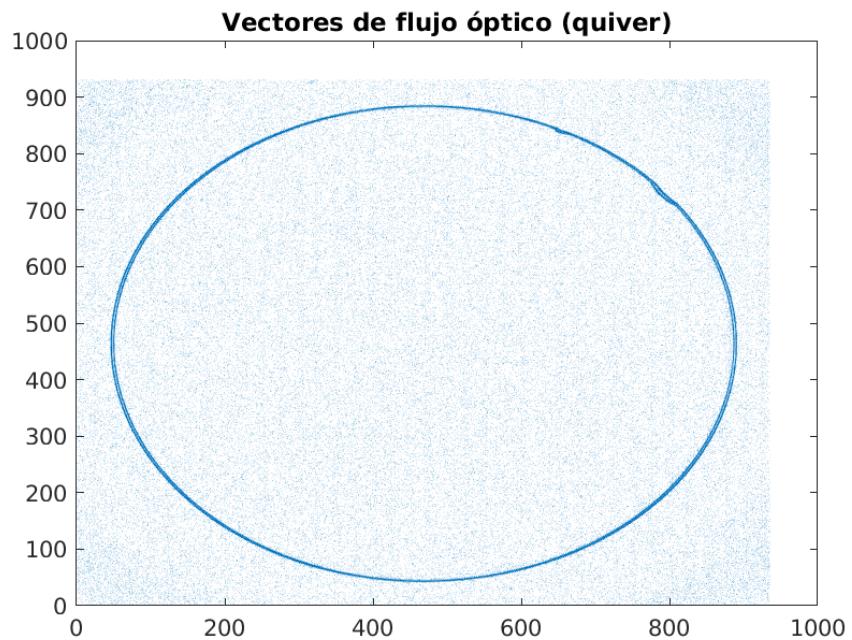


Figure 14: Resultado de quiver sobre la imagen.

El resultado de aplicar la función quiver en matlab da como resultado el mapeo de los bordes de la imagen con su representación como vectores (flechas). Salta a la vista que pese a que no se observan directamente los vectores, estos se encuentran acomodados alrededor del borde del circulo, formando a su vez el contorno de la figura.

## **4 Conclusiones**

Tras la implementación del algoritmo en ambos lenguajes de programación, se observó como no difieren prácticamente en nada los resultados de las implementaciones entre ambos lenguajes, esto se debe a que en esencia, están computando lo mismo ambos lenguajes.

Por otro lado, los resultados obtenidos por el operador de sobel resultan impresionantes debido a la precisión con la que logra extraer el borde de la imagen y eliminar todo lo que no representa un borde para el operador.

Finalmente, cabe mencionar que ya sea que se haga en un lenguaje u otro, es necesario tomar en cuenta la forma en que se hacen los cálculos internamente de las convoluciones, debido a que a medida que se pueda procesar una gran serie de imágenes o de altas resoluciones, la cantidad de operaciones que debe realizar el algoritmo aumenta rápidamente por lo que implicaría un mayor coste de tiempo y recurso computacional.

## **5 Referencias**

### **References**

Gonzalez, R. C. and Woods, R. E. (2018). *Digital Image Processing*. Pearson.

Pérez, P. and Valente, M. (2018). Procesamiento de imágenes con derivadas - detección de esquinas y bordes. Accedido: 19 de marzo de 2025.

## 6 Anexos

### 6.1 Implementación de la ecualización por histograma

```
1 ##### Paquetes necesarios #####
2 using Plots
3 using Images
4 using StatsBase
5
6 # Función del gradiente de SOBEL #
7 function imsobel(img_ruta::String)
8     # Cargar la imagen
9     img = load(img_ruta)
10    # Convertir la imagen a escala de grises y multiplicar por 255
11    img1 = Gray.(img) .* 255
12    # Generar la matriz que detecta en Gx
13    gxk = [-1 0 1; -2 0 2; -1 0 1]
14    # Generar la matriz que detecta en Gy
15    gyk = gxk' # Usamos la transpuesta de gxk
16    # Obtener las dimensiones de la imagen y el kernel
17    imdim = size(img1)
18    kerdim = size(gxk)
19    # Calcular las dimensiones de la imagen resultante
20    rf = imdim[1] - kerdim[1] + 1
21    rc = imdim[2] - kerdim[2] + 1
22    # Crear un array para guardar los resultados
23    resultado = zeros(rf, rc, 4)
24    # Aplicar la convolución
25    for i in 1:rf
26        for j in 1:rc
27            # Extraer una submatriz de nxn de la imagen
28            subim = img1[i:i+kerdim[1]-1, j:j+kerdim[2]-1]
29
30            # Aplicar la convolución sobre GX
31            resultado[i, j, 1] = sum(subim .* gxk)
32
33            # Aplicar la convolución sobre GY
34            resultado[i, j, 2] = sum(subim .* gyk)
35
36            # Calcular el operador de Sobel
37            sobel = sqrt(resultado[i, j, 1]^2 + resultado[i, j, 2]^2)
```

```

38
39                      # Guardar el valor del gradiente
40                      resultado[i, j, 3] = sobel
41                  end
42              end
43          ##### Reportar las salidas #####
44          # Reportar Gx
45          rgx = resultado[:, :, 1]
46          # Estandarizar
47          minr = minimum(rgx)
48          maxr = maximum(rgx)
49          p1 = (rgx .- minr) ./ (maxr - minr)
50          q1 = 1 .- p1
51          # Reportar Gy
52          rgy = resultado[:, :, 2]
53          # Estandarizar
54          minr = minimum(rgy)
55          maxr = maximum(rgy)
56          p2 = (rgy .- minr) ./ (maxr - minr)
57          q2 = 1 .- p2
58          # Reportar Sobel
59          rg = resultado[:, :, 3]
60          # Estandarizar
61          minr = minimum(rg)
62          maxr = maximum(rg)
63          p3 = (rg .- minr) ./ (maxr - minr)
64          q3 = 1 .- p3
65          # Operador de la dirección
66          resultado[:, :, 4] = atan.(resultado[:, :, 2], resultado[:, :, 1])
67          q4 = resultado[:, :, 4]
68          # Normalizar el ángulo a [0, 1]
69          q4 = (q4 .+ pi) ./ (2 * pi)
70          # Exportar salidas
71          return Gray.(q1), Gray.(q2), Gray.(q3), Gray.(q4)
72      end
73
74      # Obtener resultados #
75      # Ejemplo de uso
76      ruta = "Fig3.45(a).jpg"
77      q1, q2, q3, q4 = imsobel(ruta)

```

```
78 # Puedes guardar las imágenes o mostrarlas usando imshow
79 save("q1.png", q1)
80 save("q2.png", q2)
81 save("q3.png", q3)
82 save("q4.png", q4)
```

## 6.2 Implementación del operador de Sobel en Matlab

```
1 function [q1, q2, q3, q4, Gx, Gy] = imsobel(img_ruta)
2 % Cargar la imagen
3 img = imread(img_ruta);
4 % Convertir la imagen a escala de grises y a double
5 if size(img, 3) == 3
6 img1 = double(rgb2gray(img));
7 else
8 img1 = double(img);
9 end
10
11 % Generar la matriz que detecta en Gx
12 gxk = [-1 0 1; -2 0 2; -1 0 1];
13
14 % Generar la matriz que detecta en Gy
15 gyk = gxk'; % Usamos la transpuesta de gxk
16
17 % Obtener las dimensiones de la imagen y el kernel
18 [imdim_rows, imdim_cols] = size(img1);
19 [kerdim_rows, kerdim_cols] = size(gxk);
20
21 % Calcular las dimensiones de la imagen resultante
22 rf = imdim_rows - kerdim_rows + 1;
23 rc = imdim_cols - kerdim_cols + 1;
24
25 % Crear un array para guardar los resultados
26 resultado = zeros(rf, rc, 4);
27
28 % Aplicar la convolución
29 for i = 1:rf
30     for j = 1:rc
31         % Extraer una submatriz de nñn de la imagen
32         subim = img1(i:i+kerdim_rows-1, j:j+kerdim_cols-1);
33
34         % Aplicar la convolución sobre GX
35         resultado(i, j, 1) = sum(subim .* gxk, 'all');
36
37         % Aplicar la convolución sobre GY
38         resultado(i, j, 2) = sum(subim .* gyk, 'all');
```

```

40      % Calcular el operador de Sobel
41      sobel = sqrt(resultado(i, j, 1)^2 + resultado(i, j, 2)^2);
42
43      % Guardar el valor del gradiente
44      resultado(i, j, 3) = sobel;
45      end
46
47
48      % Operador de la dirección
49      resultado(:,:,4) = atan2(resultado(:,:,2), resultado(:,:,1));
50
51      % Reportar Gx
52      Gx = resultado(:,:,1);
53      % Estandarizar
54      minr = min(Gx(:));
55      maxr = max(Gx(:));
56      p1 = (Gx - minr) / (maxr - minr);
57      q1 = 1 - p1;
58
59      % Reportar Gy
60      Gy = resultado(:,:,2);
61      % Estandarizar
62      minr = min(Gy(:));
63      maxr = max(Gy(:));
64      p2 = (Gy - minr) / (maxr - minr);
65      q2 = 1 - p2;
66
67      % Reportar Sobel
68      rg = resultado(:,:,3);
69      % Estandarizar
70      minr = min(rg(:));
71      maxr = max(rg(:));
72      p3 = (rg - minr) / (maxr - minr);
73      q3 = 1 - p3;
74
75      % Operador de la dirección
76      q4 = resultado(:,:,4);
77      % Normalizar el ángulo a [0, 1]
78      q4 = (q4 + pi) / (2 * pi);
79

```

```

80      % Mostrar y guardar los vectores de flujo óptico
81      figure;
82      quiver(Gx, -Gy); % Se invierte Gy para la visualización correcta
83      title('Vectores de flujo óptico (quiver)');
84      saveas(gcf, 'quiver.png'); % Guardar la figura en la carpeta actual
85
86      % Exportar salidas
87      q1 = uint8(q1 * 255);
88      q2 = uint8(q2 * 255);
89      q3 = uint8(q3 * 255);
90      q4 = q4; % q4 ya está normalizado a [0, 1]
91
92      % Guardar las imágenes resultantes en la carpeta actual
93      imwrite(q1, 'q1.png');
94      imwrite(q2, 'q2.png');
95      imwrite(q3, 'q3.png');
96      imwrite(q4, 'q4.png');
97  end
98
99  % Ejemplo de uso
100 img_ruta = 'Fig3.45(a).jpg';
101 [q1, q2, q3, q4, Gx, Gy] = imsobel(img_ruta);
102
103
104 % Mostrar las imágenes resultantes
105 figure; imshow(q1); title('q1 (1 - Gx)');
106 figure; imshow(q2); title('q2 (1 - Gy)');
107 figure; imshow(q3); title('q3 (1 - Sobel)');
108 figure; imshow(q4); title('q4 (Dirección)')

```