CENTRO DE ENSEÑANZA TECNICA INDUSTRIAL
PRACTICAS DE LABORATORIO

INGENIERÍA CARRERA	PLAN DE ESTUDIO	CLAVE ASIGNATURA	NOMBRE DE LA ASIGNATURA
INGENIERÍA DE DESARROLLO DE SOFTWARE	2025-B	19SDS32	Procesamiento de Imágenes
ALUMNO		**FECHA**	**EVALUACIÓN**

PRÁCTICA No.	LABORATORIO DE COMPUTACIÓN No	NOMBRE DE LA PRÁCTICA	DURACIÓN (HORAS)
7	LCS	Filtros canny & laplaciano	2

INTRODUCCIÓN

La detección de bordes es una técnica fundamental en el procesamiento de imágenes y visión por computadora. Permite identificar los límites de los objetos dentro de una imagen, lo que es crucial para tareas como el reconocimiento de objetos, segmentación y análisis de imágenes. Existen múltiples técnicas para la detección de bordes, entre las más populares están el Filtro de Canny y el Filtro Laplaciano. En esta práctica, se implementan ambos filtros utilizando MATLAB para analizar sus resultados en una imagen y comparar su efectividad en la detección de bordes.

OBJETIVO (COMPETENCIA)

El objetivo de esta práctica es aplicar los filtros de detección de bordes Canny y Laplaciano en una imagen digital utilizando MATLAB, y analizar las diferencias en los resultados obtenidos por ambos métodos, evaluando su eficacia para resaltar los contornos de los objetos en la imagen.

FUNDAMENTO

Filtro de Canny

El algoritmo de Canny es uno de los métodos más utilizados para la detección de bordes debido a su capacidad para detectar bordes de manera eficiente y con pocos falsos positivos. Sus etapas principales son:

- Suavizado de la imagen: Se aplica un filtro Gaussiano para reducir el ruido y las variaciones menores en la imagen.
- Cálculo del gradiente: Se utilizan operadores como Sobel para calcular la magnitud y dirección del gradiente en cada píxel.
- Supresión de no máximos: Se eliminan los píxeles que no son máximos locales en la dirección del gradiente, preservando solo los bordes más fuertes.

- Umbralización con histéresis: Se aplican dos umbrales (alto y bajo) para clasificar los bordes como fuertes, débiles o no bordes, conectando bordes débiles a fuertes si están en vecindad.

Filtro Laplaciano

El filtro Laplaciano es un operador de segundo orden que detecta bordes resaltando las regiones de cambio rápido en la intensidad de la imagen. Su fórmula matemática es:

$$L(x,y) = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2}$$

Donde I es la intensidad de la imagen en el punto (x,y). El filtro Laplaciano es isotrópico, lo que significa que responde igual en todas las direcciones, a diferencia de los operadores de primer orden como Sobel.

METODOLOGÍA (DESARROLLO DE LA PRACTICA)

El código consta de dos funciones principales que implementan los filtros Canny y Laplaciano para detectar bordes en una imagen de ejemplo llamada carro.jpg.

Función Canny(Im_original, TH, TL):

- La imagen se convierte primero a escala de grises y se suaviza usando un filtro gaussiano.
- Luego, se calculan los gradientes horizontales y verticales con el filtro de Sobel para detectar los cambios de intensidad en las direcciones horizontal y vertical.
- Se calcula la magnitud y la dirección del gradiente, y se realiza la supresión de no máximos para eliminar los bordes falsos.
- Finalmente, se aplica umbralización con histéresis para clasificar los píxeles en bordes fuertes y débiles, conectando los bordes débiles a los fuertes si están cercanos.
- El resultado se muestra en un gráfico en el paso final, con el título "Filtro Canny".

Función Laplaciano(Im_original):

- La imagen se convierte en escala de grises y se calcula el operador Laplaciano en cada píxel de la imagen. Este operador es una segunda derivada que identifica áreas de cambio rápido en la intensidad.

- Se aplica un factor de nitidez w para ajustar la intensidad del realce de los bordes y se normaliza la imagen resultante para que sus valores estén entre 0 y 255.
- El resultado final muestra los bordes detectados por el filtro Laplaciano, y se despliega en un gráfico titulado "Filtro Laplaciano".

Código:

```
%FILTRO DE CANNY
Im = imread('carro.jpg');
Im_bordes = Canny(Im, 100, 40);
Im_bordes2 = Laplaciano(Im);
function [Im_bordes] = Canny(Im_original,TH,TL)
   Im_original=rgb2gray(Im_original);
   Im_mues=Im_original(1:1:end,1:1:end);
   Im_mues=double(Im_mues);
   %1. FILTRADO.-------------------------------------------
   H = fspecial('gaussian',[5 5],1.4); %Mascara gaussiana con std=1.4
   filtro = imfilter(Im_mues, H); %Filtrado y suavizado de la imagen
   subplot(2, 3, 1);
   imshow(mat2gray(filtro));
   title("Imagen suavizada")

   %2. CÁLCULO DEL GRADIENTE.------------------------------
   %Definir kernel horizontal y vertical
   Hx=[-1 0 1; -2 0 2; -1 0 1]; %Horizontal
   Hy=[-1 -2 -1 ; 0 0 0; 1 2 1]; %Vertical
   %Cálculo del gradiente en ambas direcciones
   Gx=abs(imfilter(Im_mues,Hx));
   Gy=abs(imfilter(Im_mues,Hy));
   %3. CÁLCULO DE LA MAGINITUD Y LA DIRECCIÓN.---------------
   Im=sqrt(Gx.^2 + Gy.^2); %Magnitudes
   teta=atand(Gy./ Gx); %Direcciónes. resultado en grados
   subplot(2, 3, 2);
   imshow(mat2gray(Im));


   %4. REDONDEO A DIRECCIONES VÁLIDAS.----------------------
   in = teta>=0 & teta<22.5;
   teta(in)= 0 ; %Dirección horizontal
   in = teta>=157.5 & teta<=180;
   teta(in) = 0; %Dirección horizontal
   in = teta>=22.5 & teta<67.5;
   teta(in) = 45; %Direccion Diagonal
   in = teta>=67.5 & teta<112.5;
   teta(in) = 90; %Dirección vertical
   in = teta>=112.5 & teta<157.5;
   teta(in) = 135; %Direccion Diagonal

   %5. ELIMINACIÓN DE NO MAXIMOS
   [M, N]=size(Im); %Dimensión de la imagen con magnitudes de los pixeles


   In=zeros(M-2,N-2); %Imagen no maximos
   for i=2:M-1 %Barrido de la imagen
       for j=2:N-1
           switch(teta(i,j))
               case 0 %Comparar vecinos en direccion horizontal
```

```
                    [~,m]=max([Im(i,j-1) Im(i,j) Im(i,j+1)]);
                    if (m==2) %Si gradiente mayor que los vecinos
                        In(i-1,j-1)=Im(i,j); %Se considera borde
                    end
                case 45 %Comparar vecinos diagonales
                    [~,m]=max([Im(i+1,j-1) Im(i,j) Im(i-1,j+1)]);
                    if (m==2) %Si gradiente mayor que los vecinos
                        In(i-1,j-1)=Im(i,j);%Se considera borde
                    end
                case 90 %Comparar vecinos en direccion vertical
                    [~,m]=max([Im(i-1,j) Im(i,j) Im(i+1,j)]);
                    if (m==2)%Si gradiente mayor que los vecinos
                        In(i-1,j-1)=Im(i,j); %Se considera borde
                    end

                case 135 %Comparar vecinos diagonales
                    [~,m]=max([Im(i-1,j-1) Im(i,j) Im(i+1,j+1)]);
                    if (m==2)%Si gradiente mayor que los vecinos
                        In(i-1,j-1)=Im(i,j); %Se considera borde
                    end
            end
        end
    end
    subplot(2, 3, 3);
    imshow(mat2gray(In));
    %6. UMBRALIZACIÓN CON HISTÉRESIS
    [M,N]=size(In); %Dimensión de imagen no maximos
    Binaria=zeros(M,N);%Imagen de bordes
    for i=1:M
        for j=1:N
            if(In(i,j)>=TH) %Si In(i,j)>=TH = 1 (Fuerte)
                Binaria(i,j)=1;
            elseif(In(i,j)>TL && In(i,j)>TH) %Si In(i,j)>TL y In(i,j)<TH (Debil)
                Binaria(i,j)=0.5;
            end
       end
    end
    subplot(2, 3, 4);
    imshow(mat2gray(Binaria));
    Itemp=zeros(M+2,N+2); %Se crea una imagen con un marco de ceros
    Itemp(2:M+1,2:N+1)=Binaria;
    for i=2:M+1 %Se hace un barrido a la imagen para detectar valores debiles
        for j=2:N+1
            if(Itemp(i,j)==0.5) %Si se detectan valores debiles, se observa si
%en su vecindad (3X3) existen valores fuertes.
                if((Itemp(i-1,j-1)==1)||(Itemp(i-1,j)==1)||(Itemp(i-1,j+1)==1)
||(Itemp(i,j-1)==1)||(Itemp(i,j+1)==1)
```

```
||(Itemp(i+1,j-1)==1)||(Itemp(i+1,j)==1)||(Itemp(i+1,j+1)==1))
                    Binaria(i-1,j-1)=1; %Si dentro de sus vecindad existe un
valor fuerte,dicho pixel se considera como borde.
                end
            end
        end
    end

    Im_bordes=Binaria;
    subplot(2, 3, 5);
    imshow(Im_bordes);
    title("Filtro Canny")
end
function [GInitN] = Laplaciano(Im_original)
    %Se aplican el filtro Laplaciano
    %véase Figura 6.26
    Im_original = rgb2gray(Im_original);
    Im_original = double(Im_original);
    [M, N]=size(Im_original);
    L = zeros(M, N);
    for r=2:M-1
        for c=2:N-1
            L(r,c)=Im_original(r-1,c-1)+Im_original(r-1,c)+Im_original(r-
1,c+1)+Im_original(r,c-1)-
8*Im_original(r,c)+Im_original(r,c+1)+Im_original(r+1,c-
1)+Im_original(r+1,c)+Im_original(r+1,c+1);
        end
    end

    %SEGUNDA PARTE
    %Se calcula los nuevos píxeles de la imagen
    %cuya nitidez se pretende mejorar (véase ecuación 6.28)
    w = .3; % Factor de nitidez
    Init=Im_original-w*L;
    %Se obtienen el valor mínimo para la imagen Init.
    VminInit=min(min(Init));
    %Utilizando el valor mínimo se desplaza para evitar negativos
    GradOffL=Init-VminInit;
    %Se obtienen el valore máximo para normalizar a 1.
    VmaxInit=max(max(GradOffL));
    %Se normalizan los gradientes a 255
    InitN=(GradOffL/VmaxInit)*255;
    %Se convierte la imagen para el despliegue
    GInitN=uint8(InitN);
    subplot(2, 3, 6);
    imshow(GInitN);
    title("Filtro Laplaciano")
end
```

RESULTADOS Y CONCLUSIONES

REFERENCIAS

1. MathWorks. (s.f.). *Edge Detection*. MATLAB. Recuperado de https://www.mathworks.com/help/images/edge-detection.html
2. Gonzalez, R. C., & Woods, R. E. (2008). *Digital Image Processing (3rd Edition)*. Pearson Prentice Hall.
3. OpenCV. (s.f.). *Canny Edge Detection*. Recuperado de https://docs.opencv.org/3.4/da/d22/tutorial_py_canny.html
4. MathWorks. (s.f.). *Laplacian of Gaussian*. MATLAB. Recuperado de https://www.mathworks.com/help/images/ref/fspecial.html

Elavoro	Observaciones	Evaluacion
DR. Gerardo García Gil		