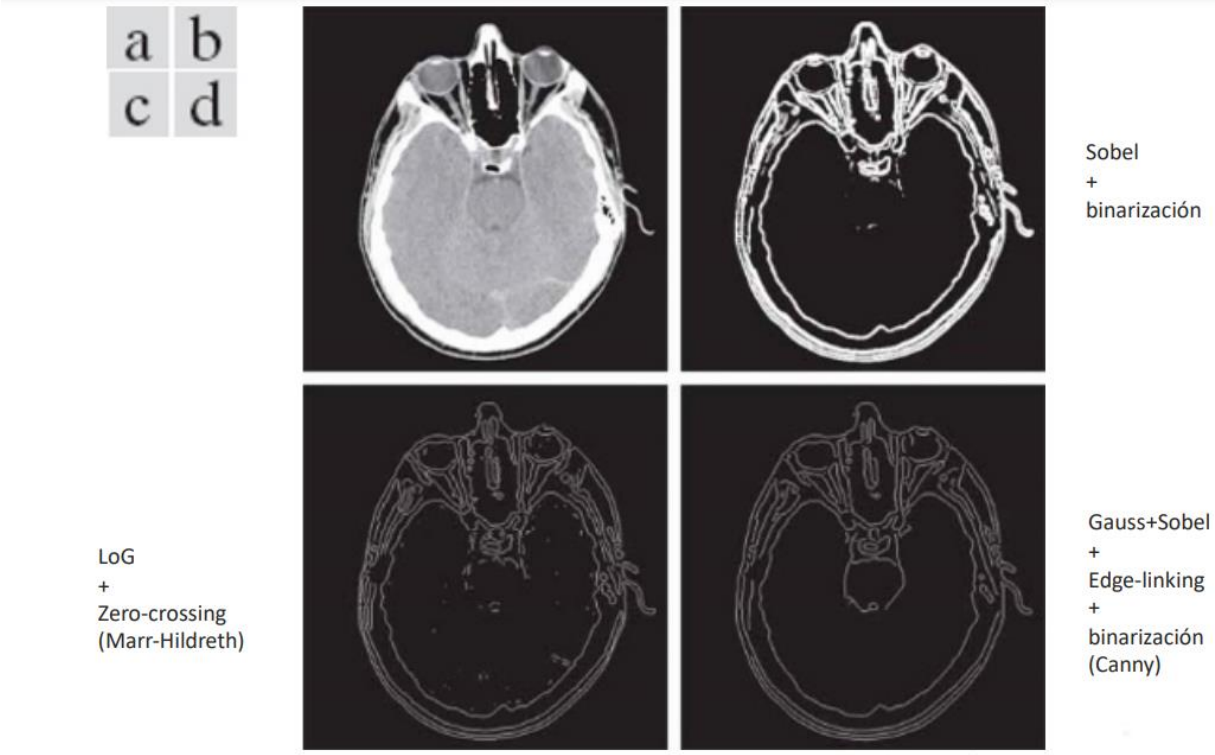
Universidad de Málaga

Materia: Imágenes biomédicas

Profesor: Ignacio Rodríguez Rodríguez

Practica2: Filtrado espacial de imágenes biomédicas



Diego De Pablo

Málaga, noviembre de 2023

índice:

Introducción:

Las imágenes biomédicas son una fuente valiosa de información para el diagnóstico y tratamiento de enfermedades. Sin embargo, estas imágenes pueden estar afectadas por ruido y artefactos, lo que dificulta su interpretación.

Los filtros lineales espaciales son una herramienta eficaz para mejorar la visualización de la información de interés en imágenes biomédicas. Estos filtros operan sobre los valores de píxel de una imagen para suavizar, realzar o resaltar características específicas.

En esta práctica, se utilizarán filtros lineales espaciales para mejorar la visualización de la información de interés en una imagen biomédica. Se utilizarán seis filtros diferentes:

- Filtro paso bajo 5x5
- Filtro de Sobel
- Filtro de Prewitt
- Filtro gaussiano de 5x5
- Filtro laplaciano de 3x3
- Filtro logaritmo de una gaussiana (LoG) de 5x5

Para cada filtro, se evaluará el resultado visual y se seleccionará el que mejor muestre la información de interés. Además, se utilizarán técnicas de gradiente para detectar los bordes de los objetos de interés en la imagen biomédica. Se utilizarán dos filtros de gradiente:

- Filtro de Sobel
- Filtro de Canny

A continuación, se trabajara esta práctica en orden cronológico para no perder la continuidad dando el apartado, codigo, output y su explicación.

También es importante recordar que esta practica consta de distintas imágenes dependiendo del alumno, en mi caso es la imagen 5 relacionada con la masa cerebral y tumor.

```
%Practica2 imagenesiMAGENES BIOMEDICAS
%Borramos las variables cargadas y cargamos la imagen
clear variables
clc
close all


imagen=dicomread('im5');
imagen_ajustada=imadjust(imagen)
figure(1)
imshow(imagen_ajustada)
```

Contenido de la practica:

Contenido de la práctica

1. Realizar un filtrado espacial de la imagen y visualizar y valorar el resultado. Para el filtrado, deberá utilizar dos algoritmos:
    1. Utilizando la función imfilter(…) de matlab. Los algoritmos a emplear deberán ser:
        1. Filtro paso bajo 5x5, todos los coeficientes del filtro valen 1.
        2. Filtro de Sobel.
        3. Filtro de Prewitt.
        4. Filtro gaussiano de 5x5, con sigma=1.
        5. Filtro laplaciano de 3x3.
        6. Filtro logaritmo de una gaussiana (LoG) de 5x5, con sigma=1.

Primero tenemos que crear un filtro, Para generar matrices manualmente podemos usar ones (3) una matriz con puro 1 3x3 o escribiendo el filtro sobel= [-1 0 1, -2 0 -2, 1 0 0]. Aunque existe una función capaz de darnos los filtros automáticamente siendo **H = fspecial (TYPE).** Donde type es el tipo de filtro (en esta práctica exploraremos su mayoría a continuación) y también se puede especificar el tamaño de la matriz.

Para aplicar los filtros mencionados, utilizaremos la función **imfilter(imagen, filtro, 'replicate')** en MATLAB. Como su nombre indica, esta función aplica un filtro dado a una imagen. En esta práctica, dejaremos los demás campos vacíos (por defectos, pero es importante mencionar que el tercer argumento se refiere al manejo de los bordes al aplicar un filtro a una imagen. En particular, el valor 'replicate' indica que los valores de los píxeles en el borde de la imagen se replicarán cuando se aplique el filtro. Esto es útil para evitar errores por dimensiones de matrices o falta de valores, específicamente cuando el filtro se aplica a los píxeles en el borde de la imagen.
También se comprobara que tanto afecta el orden a la hora de aplicar los filtros comparando ajustar la imagen antes y despúes de aplicarle el filtro.

1) Filtro paso bajo 5x5, todos los coeficientes del filtro valen 1.

Codigo de Matlab para crear el filtro paso bajo 5x5 *y ejemplo de uso:*

```
%%
h1=fspecial("average",5); %Filtro paso bajo 5x5
figure(2)
imshow(imfilter(imagen_ajustada,h1)) %
```

Explicación:

Este filtro permite el paso de frecuencias bajas, y atenúa las altas, devolviendo una imagen suavizada, consiguiendo corregir el ruido al mitigar los cambios bruscos de intensidad. Se divide entre 25 para evitar que el resultado supere el rango de niveles de grises.

El filtro h1=fspecial("average",5); en MATLAB crea un filtro paso bajo de tamaño 5x5 donde todos los coeficientes del filtro son iguales y suman 1. Este filtro se puede considerar como un filtro de media o promedio. Cuando se aplica a una imagen, tiene el efecto de suavizarla, reduciendo la variación de intensidad entre los píxeles adyacentes.

En nuestro caso. Podemos predecir que este filtro no es el ideal para la imagen, ya que en la imagen 5 no cuenta con ruido ni necesidad de suavizarla, más bien nos interesa conservar ciertos detalles finos que pueden verse afectados con este filtro.

Codigo de Matlab para comparar la imagen ajustada sin filtro, la imagen con filtro y posteriormente ajustada o la imagen ajustada a la cual se le aplico el filtro:

```
h1=fspecial("average",5); %Filtro paso bajo 5x5
f_paso_bajo=imfilter(imagen,h1);
f_paso_bajo=imadjust(f_paso_bajo);
f_paso_bajo2=imfilter(imagen_ajustada,h1);

figure('Name', 'Visualización paso bajo 5x5')
subplot(1,3,1);
imshow(imagen_ajustada)
title('Imagen Ajustada sin filtro')
subplot(1,3,2);
imshow(f_paso_bajo)
title('Filtro paso bajo y luego ajustada')
subplot(1,3,3);
imshow(f_paso_bajo2)
title('Imagen ajustada con filtro paso bajo')
```

Resultado:

Explicación:

En este caso, aunque apenas es perceptible porque vemos que el filtro no es el ideal para esta imagen. El orden en el que aplicas el ajuste de imagen y el filtro puede afectar el resultado final. Aquí está el por qué:

- **Caso 1: Ajustar primero, luego filtrar** En este caso, estás ajustando la imagen primero para mejorar su contraste. Esto puede hacer que los detalles de la imagen sean más visibles. Luego, cuando aplicas el filtro paso bajo, estás suavizando la imagen ajustada. Esto puede reducir el ruido y los detalles finos en la imagen.
- **Caso 2: Filtrar primero, luego ajustar** En este caso, estás aplicando el filtro paso bajo primero. Esto suaviza la imagen original, lo que puede reducir el ruido y los detalles finos. Luego, cuando ajustas la imagen, estás mejorando el contraste de la imagen suavizada. Esto puede hacer que los detalles restantes en la imagen sean más visibles.

Decido quedarme **con el caso 1 de la imagen ajustada y luego filtrada** porque se distingue mejor el tumor y la masa cerebral, aunque no es un buen filtro para este caso.

2) Filtro de Sobel.

Codigo de Matlab para crear el filtro y ejemplo de uso:

```
%%
%Filtro sobel
h2=fspecial('sobel');
imshow(imfilter(imagen_ajustada, h2))
%Filtro sobel doble barrido
B = double(imagen_ajustada);
fs = fspecial('sobel'); %vertical
%matriz transpuesta'
fs2 = fs'; %Filtro Sobel, tenemos que hacer filtrado horizontal y vertical
sobel1 = imfilter(B, fs);
sobel2 = imfilter(B, fs2);
%Calculamos el módulo
sobel = sqrt((sobel1.^2) + (sobel2.^2));
sobel = uint16(sobel);
figure(3)
imshow(sobel)
```

Explicación:
El filtro Sobel es un operador de detección de bordes que se utiliza en el procesamiento de imágenes. Calcula la aproximación del gradiente de la intensidad de una imagen, lo que puede ser útil para detectar cambios en la intensidad, o “bordes”, en una imagen.

El filtro Sobel es ideal para imágenes donde los bordes son importantes características que se deben resaltar. Sin embargo, es sensible al ruido en la imagen. Esto significa que, si la imagen tiene mucho ruido, los resultados pueden no ser óptimos ya que el ruido puede ser interpretado como bordes.

El filtro Sobel se puede aplicar en varias direcciones porque los bordes en una imagen pueden ocurrir en cualquier dirección. Por ejemplo, existen bordes verticales (donde la intensidad cambia horizontalmente) y bordes horizontales (donde la intensidad cambia verticalmente). Al aplicar el filtro Sobel en diferentes direcciones, puedes detectar bordes en esas direcciones.

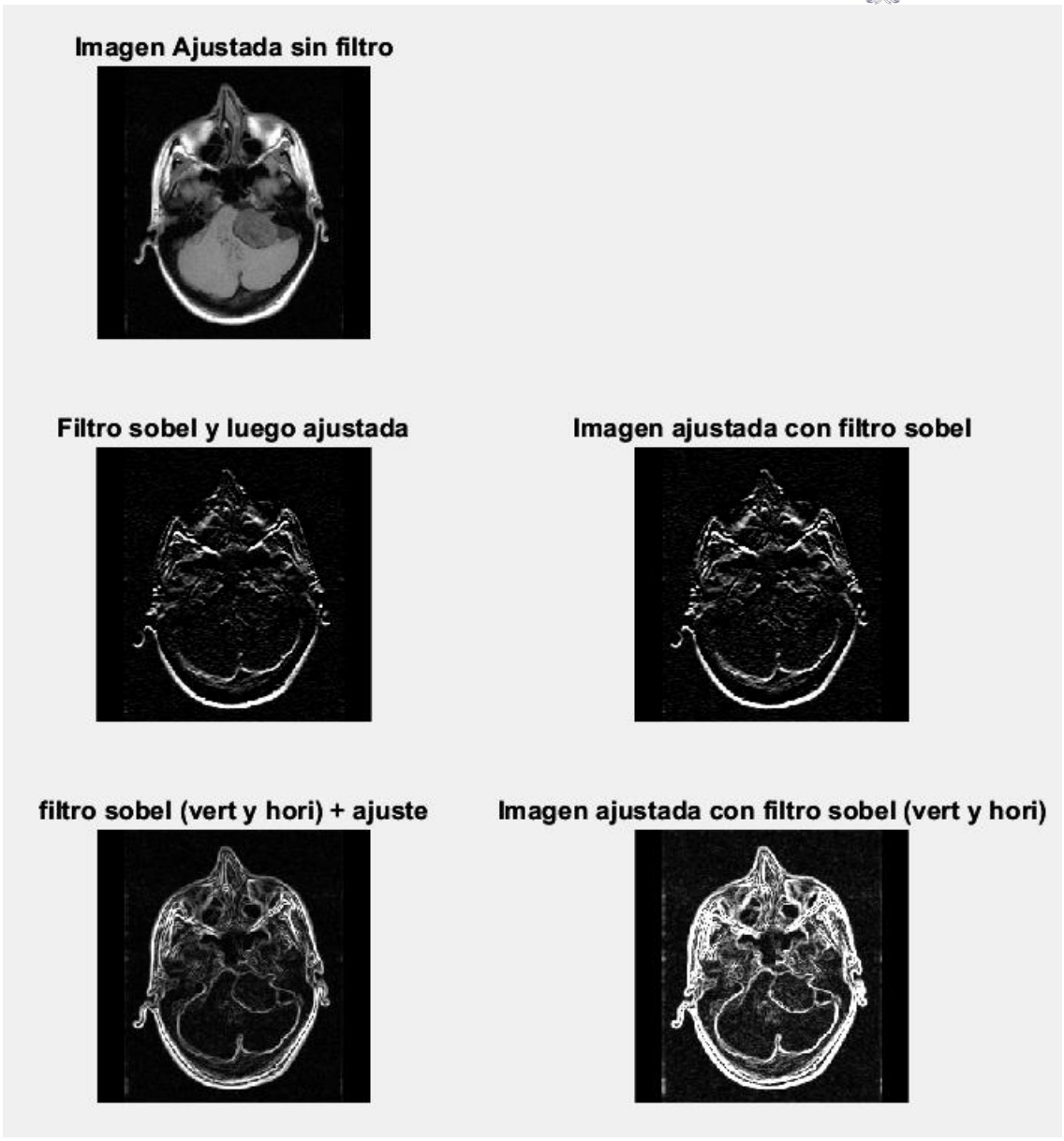
Su principal objetivo es aumentar la nitidez de la imagen y detectar bordes.

Codigo de Matlab para comparar la imagen ajustada sin filtro, la imagen con filtro (vertical), posteriormente ajustada o la imagen ajustada a la cual se le aplica el filtro (vertical), la imagen con filtro sobel en vertical y horizontal posteriormente ajustada y ajustada que se le aplica filtro vertical y horizontal:

- Imagen ajustada que no se le aplico un filtro sobel
- Imagen sin ajustar que se le aplico el filtro sobel (solamente vertical) y posteriormente ajustada.
- Imagen ajustada que se le aplico el filtro sobel (solamente vertical)
- Imagen sin ajustar que se le aplico el filtro sobel (vertical y horizontal) y posteriormente ajustada.
- Imagen ajustada que se le aplico el filtro sobel (vertical y horizontal)

```
figure('Name', 'Visualización filtro sobel')
subplot(3,2,1);
imshow(imagen_ajustada)
title('Imagen Ajustada sin filtro')
subplot(3,2,3);
imshow(sobel_simple)
title('Filtro sobel y luego ajustada')
subplot(3,2,4);
imshow(imfilter(imagen_ajustada, h2))
title('Imagen ajustada con filtro sobel')
subplot(3,2,5);
imshow(sobel_Ajustado)
title('filtro sobel (vert y hori) + ajuste') %vertical y horizontal
subplot(3,2,6);
imshow(sobel)
title('Imagen ajustada con filtro sobel (vert y hori)')
```

Resultado:



Explicación:

Ahora, en cuanto a las diferencias entre las imágenes:

1. **Imagen sin aplicar el filtro Sobel**: Esta es la imagen original solamente ajustada para tener de referencia.
2. **Imagen sin ajustar con filtro Sobel aplicado (solo vertical) y luego ajustada**: Al aplicar el filtro Sobel verticalmente a una imagen sin ajustar, se resaltan los bordes horizontales (cambios de intensidad de izquierda a derecha). Luego, al ajustar la imagen, se normalizan los valores de los píxeles para que se ajusten a un rango específico, lo que puede mejorar la visualización de los bordes detectados.
3. **Imagen ajustada con filtro Sobel aplicado (solo vertical)**: En este caso, se ajusta la imagen antes de aplicar el filtro Sobel. Esto puede afectar cómo el filtro Sobel resalta los bordes, ya que el ajuste puede cambiar los valores de los píxeles de la imagen.

4. **Imagen sin ajustar con filtro Sobel aplicado (vertical y horizontal) y luego ajustada**: Al aplicar el filtro Sobel tanto vertical como horizontalmente, se resaltan los bordes en **todas** las direcciones. Luego, al ajustar la imagen, se normalizan los valores de los píxeles, lo que puede mejorar la visualización de los bordes detectados.
5. **Imagen ajustada con filtro Sobel aplicado (vertical y horizontal)**: Al igual que en el tercer caso, ajustar la imagen antes de aplicar el filtro Sobel puede afectar cómo se resaltan los bordes.

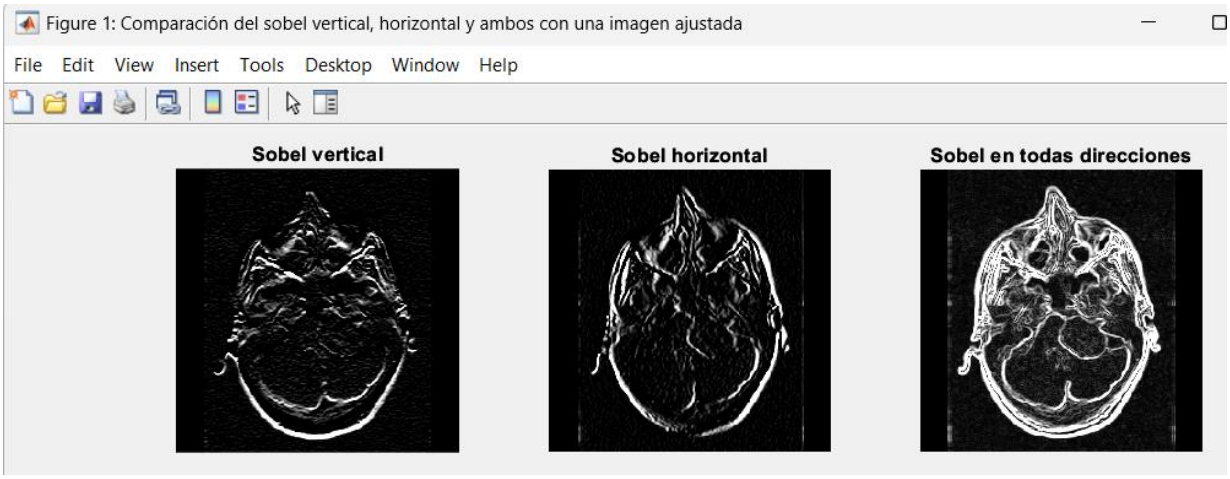
En cuanto al mejor caso considero que claramente debe ser entre la 4 y 5, el filtrado solo vertical prácticamente no podemos diferenciar el tumor de la masa cerebral, en cuanto al barrido vertical y horizontal, podemos ver el borde tanto de la masa cerebral como del tumor, sin embargo, perdemos bastante información, ya que no se ve diferencia del tumor al cerebro, **En la imagen 5 lo mejor será aplicar el filtro y posteriormente aplicar el ajuste**. Aunque no es el mejor filtro para esta imagen.

Agregado:

Considero curioso comparar el barrido solo vertical con el barrido solo horizontal, es decir:

```
h2=fspecial('sobel');
sobel_vertical=imfilter(imagen_ajustada, h2);
sobel_horizontal=imfilter(imagen_ajustada, h2');

figure('Name', 'Comparación del sobel vertical, horizontal y ambos con una imagen ajustada')
subplot(1,3,1);
imshow(sobel_vertical)
title('Sobel vertical')
subplot(1,3,2);
imshow(sobel_horizontal)
title('Sobel horizontal')
subplot(1,3,3);
imshow(sobel_todas)
title('Sobel en todas direcciones')
```



Podemos comparar como el sobel de barrido vertical muestra los bordes horizontales, pero no tan bien los bordes verticales, mientras que por el contrario el sobel de barrido muestra los bordes verticales, pero no tan bien los horizontales

3) Filtro de Prewitt

Codigo de Matlab para crear el filtro y ejemplo de uso:

```
%3Filtro de Prewitt
h3=fspecial('prewitt');
prewitt_simple=imfilter(imagen, h3);
prewitt_simple=imadjust(prewitt_simple);
prewitt_simple2=imfilter(imagen_ajustada, h3);

%prewitt en todas las direcciones
%B = double(imagen_ajustada);  esto ya fue cargado anteriormente
fp = fspecial('prewitt');
fp2 = fp';
prewitt1 = imfilter(B, fp);
prewitt2 = imfilter(B, fp2);
prewitt = sqrt((prewitt1.^2) + (prewitt2.^2));
prewitt = uint16(prewitt);
%imshow(prewitt)
```

Explicación:

El filtro Prewitt es un operador utilizado en el procesamiento de imágenes, particularmente dentro de los algoritmos de detección de bordes. Técnicamente, es un operador de diferenciación discreta, que calcula una aproximación del gradiente de la función de intensidad de la imagen. En cada punto de la imagen, el resultado del operador Prewitt es el vector de gradiente correspondiente o la norma de este vector.

El operador Prewitt es ideal para imágenes donde los bordes son características importantes que se deben resaltar. Sin embargo, la aproximación del gradiente que produce es relativamente cruda, en particular para las variaciones de alta frecuencia en la imagen.

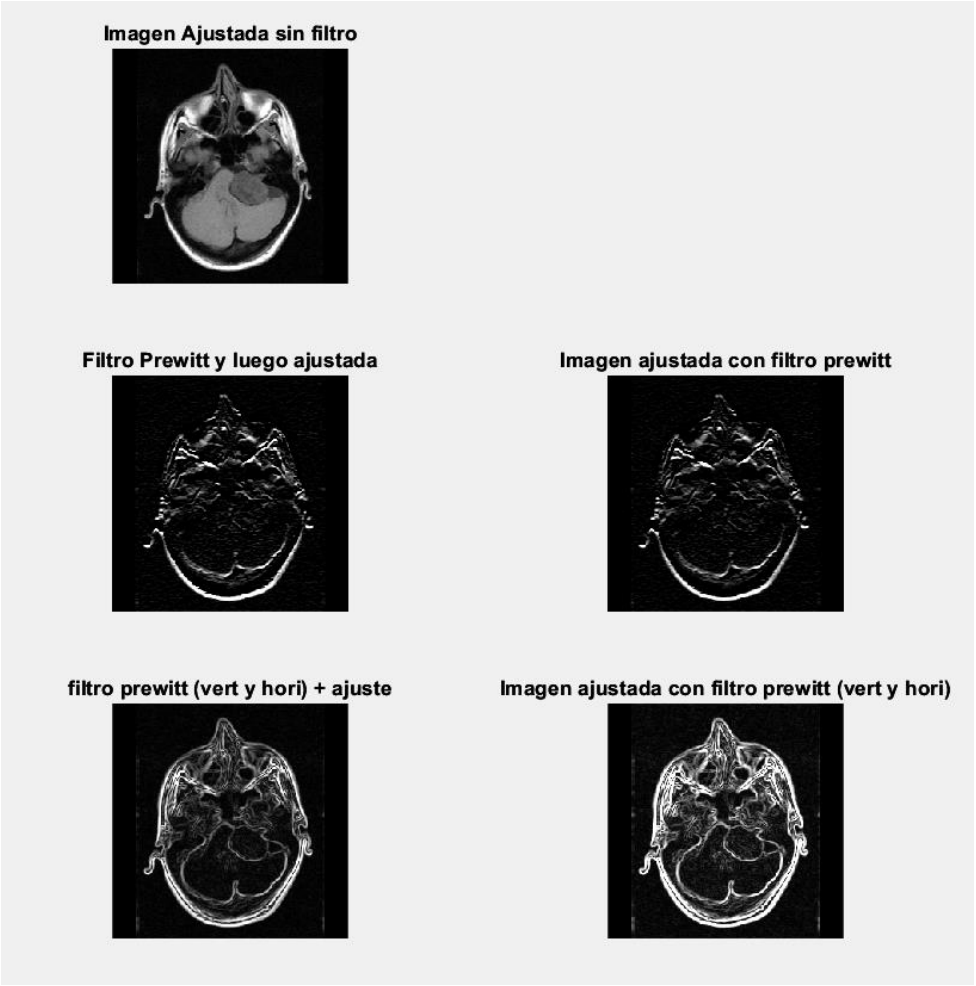
El operador Prewitt se puede aplicar en varias direcciones al igual que el filtro sobel porque los bordes en una imagen pueden ocurrir en cualquier dirección. Por ejemplo, puedes tener bordes verticales (donde la intensidad cambia horizontalmente) y bordes horizontales (donde la intensidad cambia verticalmente). Al aplicar el operador Prewitt en diferentes direcciones, puedes detectar bordes en esas direcciones.

El filtro Sobel y el filtro Prewitt son ambos operadores de detección de bordes utilizados en el procesamiento de imágenes. Ambos filtros funcionan de manera similar y utilizan dos kernels 3x3 para calcular la aproximación del gradiente de la intensidad de una imagen, sin embargo, los coeficientes de los kernels son diferentes el sobel le da mayor importancia a los pixeles cercanos al centro de la máscara mientras que prewitt trata a todos los pixeles dentro de la máscara igual.

Codigo de Matlab para comparar el uso del filtro prewitt:

```
%comparacion
figure('Name', 'Visualización filtro Prewitt')
subplot(3,2,1);
imshow(imagen_ajustada)
title('Imagen Ajustada sin filtro')
subplot(3,2,3);
imshow(prewitt_simple)
title('Filtro Prewitt y luego ajustada')
subplot(3,2,4);
imshow(prewitt_simple2)
title('Imagen ajustada con filtro prewitt')
subplot(3,2,5);
imshow(prewitt_ajustada)
title('filtro prewitt (vert y hori) + ajuste') %vertical y horizontal
subplot(3,2,6);
imshow(prewitt)
title('Imagen ajustada con filtro prewitt (vert y hori)')
```

Resultado:



Explicación:

Ya se explicó que el filtro sobel y prewitt son muy parecidos, solo cambiando el peso que le dan a los pixeles de la máscara, sería redundante volver a tocar cada caso por separado, siendo el barrido en una sola dirección una imagen que no muestra todos los bordes y se pierde mucha información. Pero al igual que en el sobel se muestra el borde de la masa cerebral y el tumor, pero se pierden muchos detalles en la imagen, creo que en este caso

la imagen mejor es la imagen a la que se le aplica prewitt y luego se ajusta, por el simple hecho de mostrar mejor todos los bordes del tumor y de la masa cerebral.

4) Filtro gaussiano de 5x5, con sigma=1.

Codigo de Matlab para crear el filtro y ejemplo de uso:

```
%4Filtro gaussiano de 5x5, con sigma=1.
h4 = fspecial('gaussian', [5 5], 1);
fg = imfilter(imagen_ajustada, h4);

fg_sinajustar = imfilter(imagen, h4);
fg_ajustada = imadjust(fg_sinajustar);
```

Explicación:
Un filtro gaussiano es un tipo de filtro de paso bajo que se utiliza para reducir el ruido y suavizar una imagen. El filtro gaussiano se implementa como una matriz (o kernel) que se pasa a través de cada píxel de la imagen para obtener el efecto deseado.

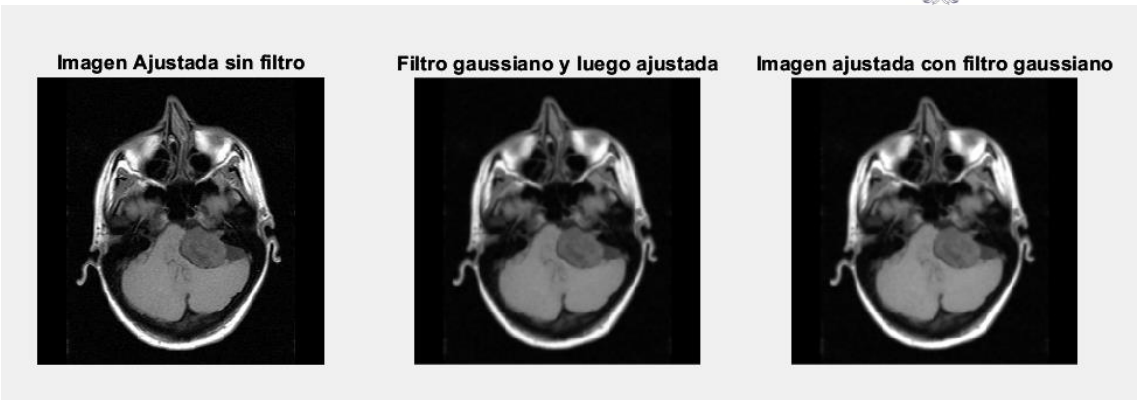
El codigo, h4 = fspecial('gaussian', [5 5], 1); crea un filtro gaussiano de tamaño 5x5 con una desviación estándar (sigma) de 1 como es requerido.

El término “sigma” se refiere a la desviación estándar de la distribución gaussiana. En el contexto de un filtro gaussiano, sigma controla el *ancho* de la función gaussiana, es decir, cuánto se extiende la función gaussiana desde su punto central. Un valor de sigma más grande dará como resultado un efecto de suavizado más fuerte, mientras que un valor de sigma más pequeño dará como resultado un efecto de suavizado más débil.

Codigo de Matlab para comparar la imagen ajustada sin filtro, la imagen con filtro y posteriormente ajustada o la imagen ajustada a la cual se le aplica el filtro:

```
%comparacion
figure('Name', 'Visualización filtro gaussiano 5x5')
subplot(1,3,1);
imshow(imagen_ajustada)
title('Imagen Ajustada sin filtro')
subplot(1,3,2);
imshow(fg)
title('Filtro gaussiano y luego ajustada')
subplot(1,3,3);
imshow(fg_ajustada)
title('Imagen ajustada con filtro gaussiano')
```

Resultado:



Explicación:

El filtro gaussiano fue aplicado vemos una leve reducción de la nitidez y también una reducción del ruido de esta (notable alrededor de la masa cerebral), aunque el filtro no hace un gran cambio en la imagen tampoco perjudica gravemente la imagen.

Extra:

Podemos probar distintos valores de sigma, en nuestro caso una sigma cercana a 0,5 dará un valor más parecido a la imagen ajustada sin filtro pero sin tantas machas blancas alrededor del cerebro, en el caso de usar un sigma = 2 la imagen empezara a verse más borrosa.

5) Filtro laplaciano de 3x3.

Codigo de Matlab para crear el filtro y ejemplo de uso:

```
%5Filtro Laplaciano
h5 = fspecial ('laplacian');
fl = imfilter(imagen_ajustada, h5);

fl_sinajustar = imfilter(imagen, h5);
fl_ajustada = imadjust(fl_sinajustar);
```

Explicación:
El filtro Laplaciano es un filtro de detección de bordes. Es un operador de diferenciación de segundo orden que calcula una aproximación del gradiente de la función de intensidad de una imagen. En cada punto de la imagen, el resultado del operador Laplaciano es el vector de gradiente correspondiente o la norma de este vector.

El filtro Laplaciano es ideal para imágenes donde los bordes son características importantes que se deben resaltar. Sin embargo, la aproximación del gradiente que produce es relativamente cruda, en particular para las variaciones de alta frecuencia en la imagen.

A diferencia de los filtros de derivación de primer orden, el filtro Laplaciano detecta los bordes en toda la imagen a la vez. No es necesario aplicarlo por separado para detectar los bordes en las direcciones horizontal y vertical.

Codigo de Matlab para comparar la imagen ajustada sin filtro, la imagen con filtro y posteriormente ajustada o la imagen ajustada a la cual se le aplica el filtro:

```
%comparacion
figure('Name', 'Visualización filtro laplaciano')
subplot(1,3,1);
imshow(imagen_ajustada)
title('Imagen Ajustada sin filtro')
subplot(1,3,2);
imshow(fg_ajustada)
title('Filtro laplaciano y luego ajustada')
subplot(1,3,3);
imshow(fg)
title('Imagen ajustada con filtro laplaciano')
```

Resultado:

Explicación:

En este caso vemos un claro ejemplo como ajustar la imagen antes de aplicar un filtro puede hacer que pierda todo el sentido, la imagen ajustada pierde muchísima información al aplicarle el filtro. Aún así el filtro laplaciano muestra mucho ruido en el caso de usarlo y luego ajustar la imagen.

6) Filtro logaritmo de una gaussiana (LoG) de 5x5, con sigma=1.

Codigo de Matlab para crear el filtro y ejemplo de uso:

```
% 6Filtro logaritmo de una gaussiana (LoG) de 5x5, con sigma=1.
h6 = fspecial("log",[5 5], 1);
flog = imfilter(imagen_ajustada, h6);

flog_sinajustar = imfilter(imagen, h6);
flog_ajustada = imadjust(flog_sinajustar);
```

Explicación:
El filtro Laplaciano de una Gaussiana (LoG, por sus siglas en inglés) es un operador de detección de bordes. Este filtro combina dos operaciones: suavizado con un filtro Gaussiano y cálculo del Laplaciano.

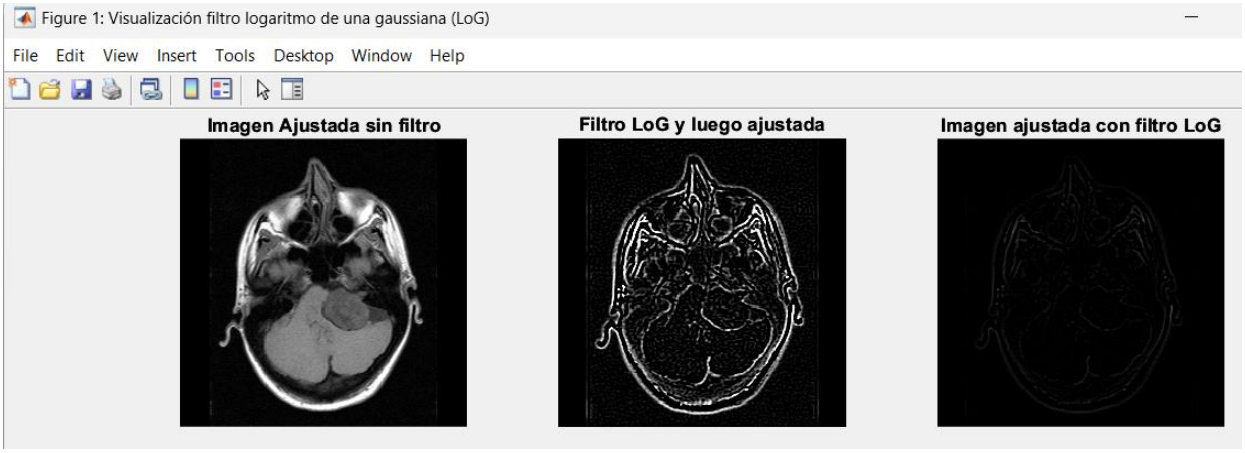
Este filtro se puede utilizar para suavizar tu imagen y detectar bordes en ella. El filtro LoG es útil para detectar bordes que aparecen en varias escalas de imagen o grados de

enfoque de imagen. Sin embargo, debido a que estos kernels están aproximando una medida de segundo orden en la imagen, son muy sensibles al ruido. Para contrarrestar esto, la imagen a menudo se suaviza con un filtro Gaussiano antes de aplicar el filtro Laplaciano.

Codigo de Matlab para comparar la imagen ajustada sin filtro, la imagen con filtro y posteriormente ajustada o la imagen ajustada a la cual se le aplica el filtro:

```
%comparacion
figure('Name', 'Visualización filtro logaritmo de una gaussiana (LoG)')
subplot(1,3,1);
imshow(imagen_ajustada)
title('Imagen Ajustada sin filtro')
subplot(1,3,2);
imshow(flog_ajustada)
title('Filtro LoG y luego ajustada')
subplot(1,3,3);
imshow(flog)
title('Imagen ajustada con filtro LoG')
```

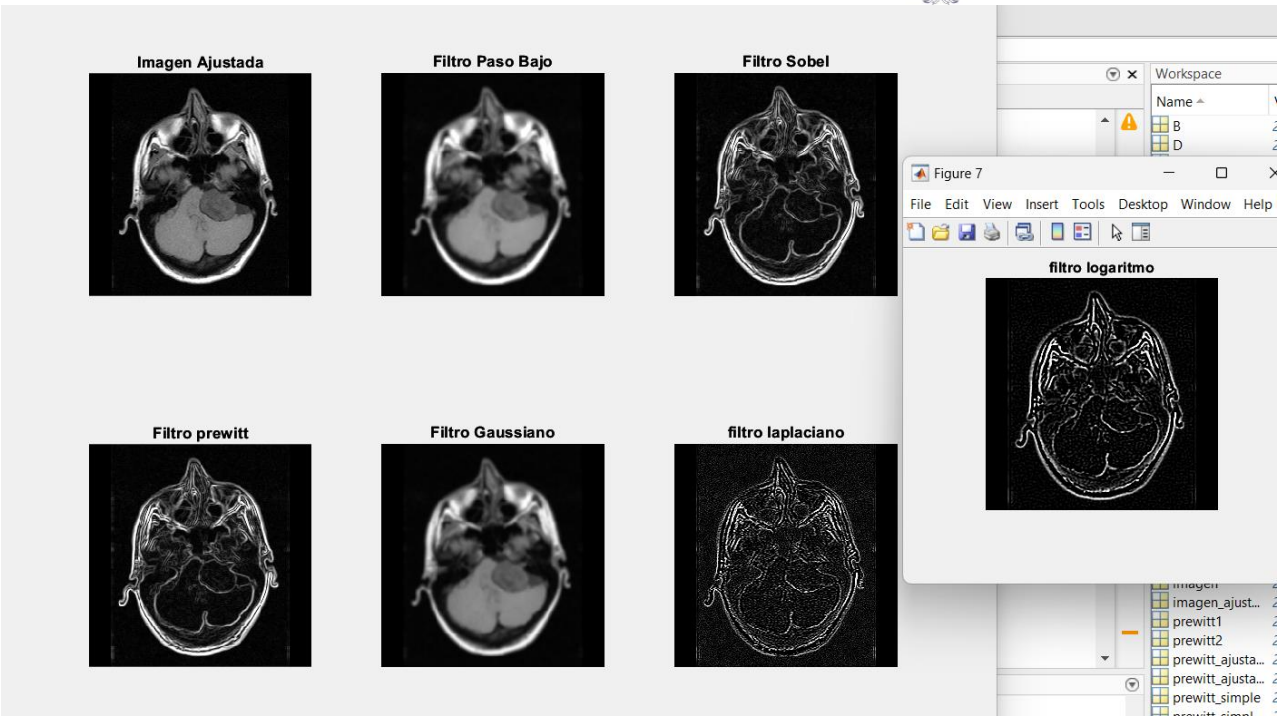
Resultado:



Explicación:

Este es un claro ejemplo de como el orden al tratar con filtros es de gran importancia.

- **Filtrar primero, luego ajustar** En este caso, estás aplicando el filtro LoG primero. Esto suaviza la imagen original y detecta los bordes. Luego, cuando ajustas la imagen, estás mejorando el contraste de la imagen suavizada. Esto puede hacer que los detalles restantes en la imagen sean más visibles.
- **Ajustar primero, luego filtrar** En este caso, estás ajustando la imagen primero para mejorar su contraste. Esto puede hacer que los detalles de la imagen sean más visibles. Luego, cuando aplicas el filtro LoG, estás suavizando la imagen ajustada y detectando los bordes.



Consideor que la **prewitt** es el mejor filtro para la imagen, aunque no es un filtro perfecto es el que da otra perspectiva donde se ven muy notoriamente los bordes del cerebro y del tumor sin ruido en lugares cercanos.

Al final del documento (luego del código) podrá encontrar el valor exacto de cada filtro y más información.

> Una vez obtenida la imagen filtrada, busque la manera de visualizar de forma óptima el resultado obtenido para cada una de los seis imágenes resultado (use los comandos imshow() o imadjust() que ha aprendido a usar anteriormente) y seleccione el que ofrece una mejor visualización subjetiva de la información de interés. Nota: El histograma de la imagen le puede resultar de utilidad, así como las herramientas aprendidas en la práctica primera.

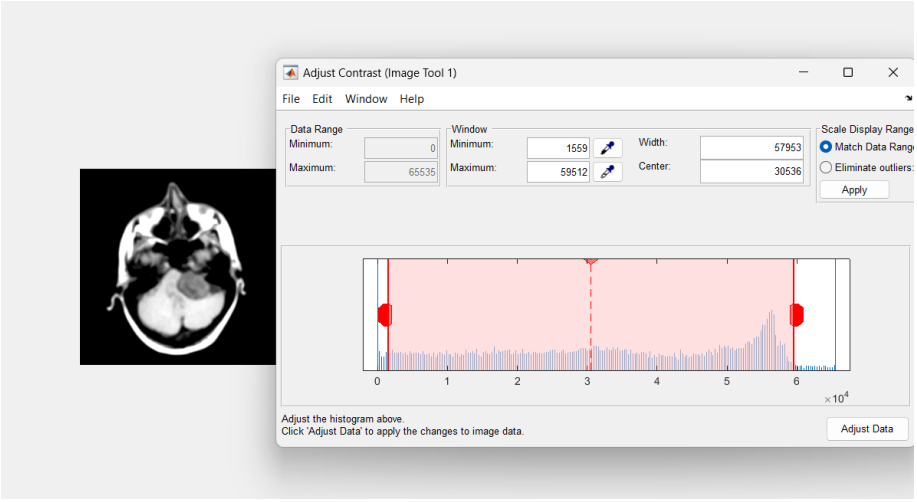
Ya se ha comparado el ajuste imadjust en el apartado anterior, a continuación, se da un codigo de ejemplo donde se podría ajustar manualmente

```
%1buscar la forma óptima manualmente
%f_paso_sinajuste=imfilter(imagen,h1);
imtool(f_paso_sinajuste)
figure(10);
f_paso_ajustado=imadjust(f_paso_sinajuste, [50000/65536 60000/65536],[0,1])
imshow(f_paso_ajustado)
```

U

```
%segundo ejemplo de buscar la forma óptima manualmente
imhist(sobel,65536)
axis([0 500 0 10000])
sobel_ajusadom= imadjust(sobel, [0 1000/65536], [0,1]);
imshow(sobel_ajustadom)
```

Ejemplo de la pantalla de imtool:



En este apartado se considera que con el imadjust se hace una buena visualización, ver las diferencias con imtool es interesante, pero son demasiado parecidas a las ya visto para que valga la pena anexarlas en la memoria, igualmente estan los códigos donde se deja probar el imtool o con el histograma, pero no logro ver cambios que sean notoriamente mejores a los ya elegidos. A su vez que en el apartado 1 no se muestran fotos de las imágenes sin ajustar con los filtros, porque las imágenes son mayormente negras y no daban material para explicar o analizar realmente la imagen.

primera.

2. Para el filtro que considere que permite una mejor visualización de la zona de interés, codifique un algoritmo propio que obtenga un resultado similar al de la rutina de Matlab.

Nota: Para encontrar descripciones de estos algoritmos, use la ayuda de Matlab para la función fspecial(…). Tenga en cuenta que las imágenes de salida deben tener el mismo tamaño que la de entrada y que debe usarse la técnica de replicación para el filtrado en los bordes exteriores de la imagen.

Programando Prewitt

Entiendo que no sé refiere a programar los filtros literalmente que sería tan facil como crear manualmente los siguientes filtros (vertical y horizontal)

h3 =

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

h3`=

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

La siguiente función al introducir una imagen le aplicara el algoritmo prewitt sin usar la función fspecial o en sí los códigos usados en el primer apartado:

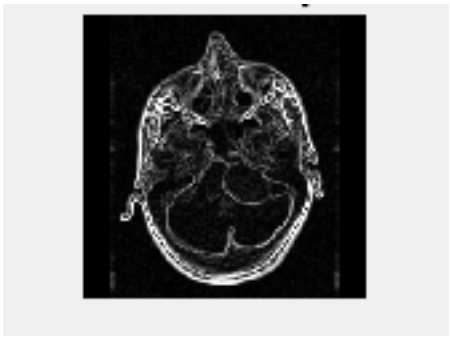
Código de matlab

```
%Funciones
function [Gx, Gy, im_mag] = prewitt_filter(im)
%lee la imagen
imagen=double(im);
[filas, columnas]=size(imagen);
Gx=zeros(filas,columnas);
Gy=zeros(filas,columnas);
% Define los kernels Prewitt
prewitt_x = [-1 0 1; -1 0 1; -1 0 1];
prewitt_y = [-1 -1 -1; 0 0 0; 1 1 1];
for i=2:filas-1
    for j=2: columnas-1
        Gx(i,j)=abs(imagen(i,j)* prewitt_x (2,2)+ imagen(i,j+1)* prewitt_x (2,3) + ...
        imagen(i-1,j+1)* prewitt_x (1,3)+ imagen(i-1,j)* prewitt_x (1,2) + ...
        imagen(i-1,j-1)* prewitt_x (1,1)+ imagen(i,j-1)* prewitt_x (2,1) + ...
        imagen(i+1,j-1)* prewitt_x (3,1)+ imagen(i+1,j)* prewitt_x (3,2)+  ...
        imagen(i+1,j+1)*prewitt_x(3,3));
        %con el eje y
        Gy(i,j)=abs(imagen(i,j)* prewitt_y (2,2)+ imagen(i,j+1)* prewitt_y (2,3) + ...
        imagen(i-1,j+1)* prewitt_y (1,3)+ imagen(i-1,j)* prewitt_y (1,2) + ...
        imagen(i-1,j-1)* prewitt_y (1,1)+ imagen(i,j-1)* prewitt_y (2,1) + ...
        imagen(i+1,j-1)* prewitt_y (3,1)+ imagen(i+1,j)* prewitt_y (3,2)+  ...
        imagen(i+1,j+1)*prewitt_y(3,3));

    end
end
% Calcula la magnitud del gradiente
im_mag = sqrt(double(Gx).^2 + double(Gy).^2);

end
```

Resultado:



Explicación:

1. Primero, se lee la imagen y obtiene su tamaño con imagen=double(im); [filas, columnas]=size(imagen);. También inicializa las matrices Gx y Gy para almacenar los resultados de la convolución con los kernels Prewitt.

2. Luego, estás definiendo los kernels Prewitt con prewitt_x = [-1 0 1; -1 0 1; -1 0 1]; y prewitt_y = [-1 -1 -1; 0 0 0; 1 1 1]; (Esto sería como usar el h3=fspecial('prewitt') y h3’ (matriz transpuesta de h3))

3. Después de eso, el algoritmo queda recorriendo cada píxel de la imagen y aplicando los kernels Prewitt en las direcciones x e y, excepto el borde por temas de facilidad, se podría tomar funciones como replicate , symmetric y circular, pero se decidió trabajar con zero. Que viene a ser el margen no se ve afectado por la máscara, puesto que tomara dichos valores como 0. Esto se hace con los

bucles for y las líneas de código especificando al empezar en el borde de la imagen.

4. Finalmente, se calcula la magnitud del gradiente con im_mag = sqrt(double(Gx).^2 + double(Gy).^2);.

La imagen obtenida es muy parecida a la esperada, esto es porque la imagen 5 no cuenta con elementos en los márgenes, son pixeles negros de valor 0

2. Realizar una detección de los bordes de los objetos de interés en la imagen utilizando técnicas de gradiente.
    1. Utilice la función edge(), en los siguientes algoritmos:
        i. Filtro de Sobel, con un umbral que deberá optimizar. Se recomienda probar varios umbrales y valorar el resultado obtenido.

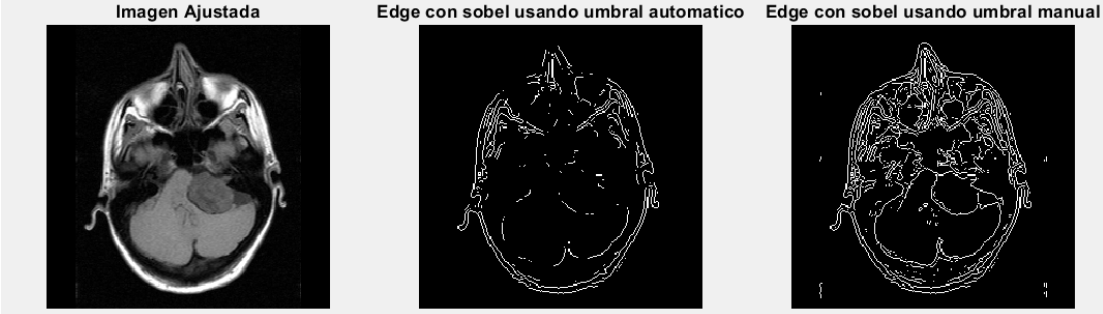
Código de Matlab:

```
%% Apartado 2
%Utilice la funcion edge en los siguientes algoritmos
[imagen_binaria,thresh]= edge(imagen, 'sobel'); %usa sobel por defecto
thresh %ver umbral automatico
figure(1)
%imshow(imagen_binaria)

[imagen_manual]= edge(imagen, 'sobel',0.00035); %usa sobel por defecto
figure(2)
%imshow(imagen_manual)

%comparacion
figure('Name', 'Visualización para función edge')
subplot(1,3,1);
imshow(imagen_ajustada)
title('Imagen Ajustada')
subplot(1,3,2);
imshow(imagen_binaria)
title('Edge con sobel usando umbral automatico')
subplot(1,3,3);
imshow(imagen_manual)
title('Edge con sobel usando umbral manual')
```

Resultado:



Explicación:

La función edge en MATLAB es una función de detección de bordes que se utiliza para identificar los puntos en una imagen donde la intensidad de los píxeles cambia de manera abrupta. La función edge puede utilizar varios métodos de detección de bordes, incluyendo el operador Sobel, que es el método predeterminado.

La función devuelve una imagen binaria que contiene 1s donde la función encuentra bordes en la imagen y 0s en otros lugares. Esto estos bordes se determinan según un umbral que puede ser generado automáticamente o dado manualmente por el usuario (en el código se ven ambos casos), donde se recomienda siempre intentar encontrar el valor óptimo de manera manual.

Se podría decir que es una forma más rápida de aplicar los filtros de detección de bordes. la principal diferencia entre usar edge y imfilter con un operador Sobel es que edge realiza algunas operaciones adicionales además de aplicar el operador Sobel. Estas operaciones adicionales pueden incluir el adelgazamiento de los bordes detectados y la eliminación de los bordes débiles.

En cuanto a los parámetros que puedes pasar a la función edge, aquí están algunos de los más comunes:

- **I**: la imagen
- **method**: Este parámetro especifica el método de detección de bordes que quieres usar. Puede ser 'sobel', 'prewitt', 'roberts', 'log' (Laplaciano de una Gaussiana), 'canny', 'approxcanny' and zero cross
- **thresh**: Este parámetro especifica el valor de umbral que se utiliza para determinar qué píxeles en la imagen constituyen un borde.
- **direction**: Este parámetro especifica la orientación de los bordes a detectar.
- **sigma:** Este parámetro especifica la desviación estándar del filtro, y sólo es válido cuando method es 'log' o 'canny'.

La función edge puede devolver varios valores:

- **BW:** Una imagen binaria que contiene 1s donde la función encuentra bordes en la imagen y 0s en otros lugares.
- **threshout:** El valor de umbral utilizado por la función edge.
- **Gx, Gy:** Los gradientes direccionales.

Cabe acotar que el umbral manual deja ver todo el borde entre la masa cerebral y el tumor, cosa que se pierde al dejar este valor automatizado, ya que cierta parte del borde no logra superar el umbral que se establece automáticamente de 8.7621e-04.

ii. Filtro de Canny, con 3 parámetros que deberá optimizar.

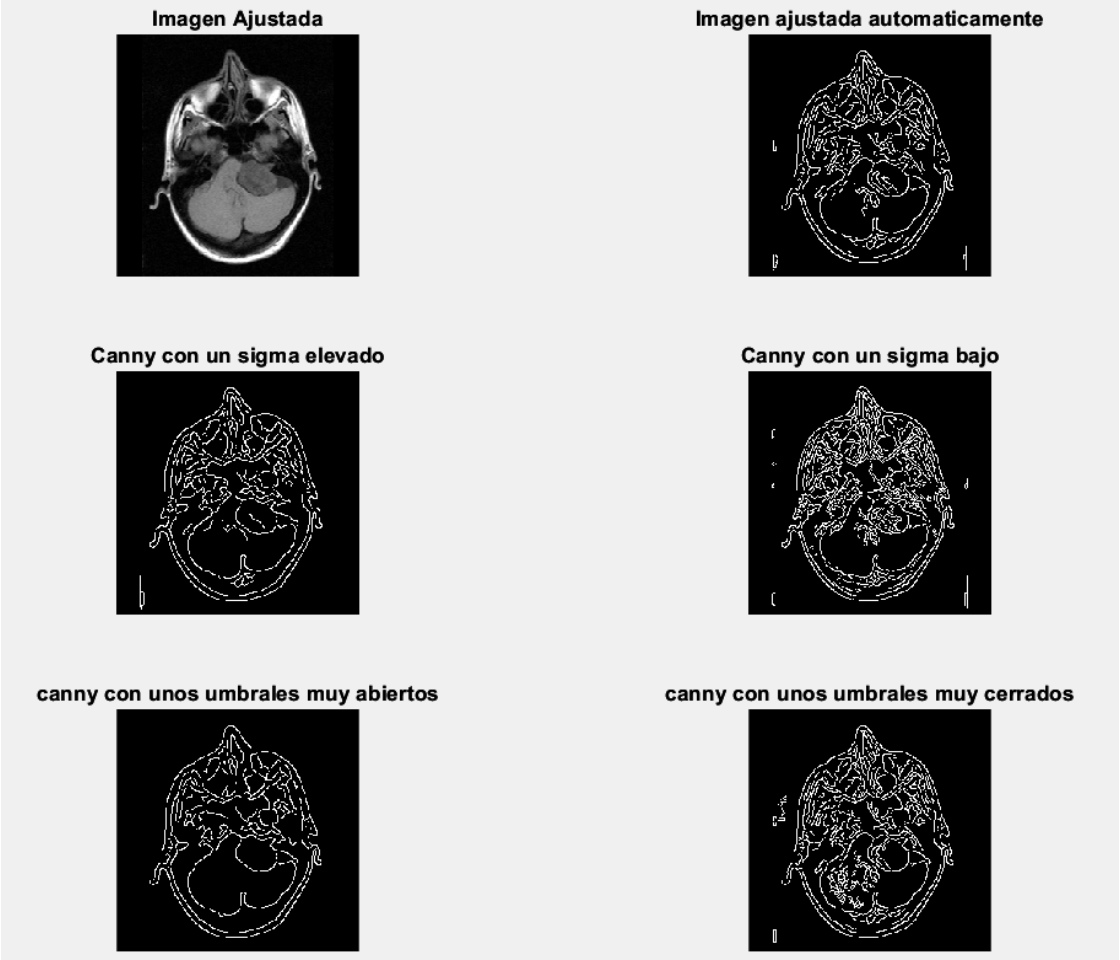
Codigo de matlab:

```
[imagen_automatica,thresh]= edge(imagen, 'canny');
thresh %ver umbral automatico
%imshow(imagen_automatica)

canny1= edge(imagen, 'canny', [0.0300 0.0800],sqrt(3));
canny2= edge(imagen, 'canny', [0.0300 0.0800],sqrt(0.5));
canny3= edge(imagen, 'canny', [0 0.1800],sqrt(3));
canny4= edge(imagen, 'canny', [0.006 0.106],sqrt(1.2));

figure('Name', 'Visualización canny')
subplot(3,2,1);
imshow(imagen_ajustada)
title('Imagen Ajustada')
subplot(3,2,2);
imshow(imagen_automatica)
title('Imagen ajustada automaticamente')
subplot(3,2,3);
imshow(canny1)
title('Canny con un sigma elevado')
subplot(3,2,4);
imshow(canny2)
title('Canny con un sigma bajo')
subplot(3,2,5);
imshow(canny3)
title('canny con unos umbrales muy abiertos')
subplot(3,2,6);
imshow(canny4)
title('canny con unos umbrales muy cerrados')
```

Resultado:



Explicación:

El método de Canny nombrado así por su creador, es un popular algoritmo de detección de bordes. Este algoritmo consta de múltiples etapas:

1. **Reducción de ruido**: Debido a que la detección de bordes es susceptible al ruido en la imagen, el primer paso es eliminarlo o reducirlo lo más que se pueda. Para esto, el algoritmo utiliza un filtro Gaussiano 5×5.
2. **Encontrando el gradiente de intensidad de la imagen**: Una vez que la imagen ha sido alisada con el filtro Gaussiano, se calcula el gradiente de la misma.
3. **Supresión de falsos máximos**: Esta técnica es utilizada para afinar los bordes encontrados en el paso anterior.
4. **Umbral de histéresis**: En esta cuarta etapa se decide cuáles píxeles pertenecen realmente a bordes y cuáles no. Para ello, se deben fijar dos valores de umbral, minVal y maxVal.

El método de Canny difiere de otros en que utiliza dos umbrales diferentes (para detectar bordes intensos y débiles), e incluye los bordes débiles en el resultado solo si están conectados a bordes intensos

Los valores ideales para los umbrales en el algoritmo de Canny pueden variar dependiendo de la imagen y lo que se quiera lograr. Sin embargo, algunas recomendaciones generales:

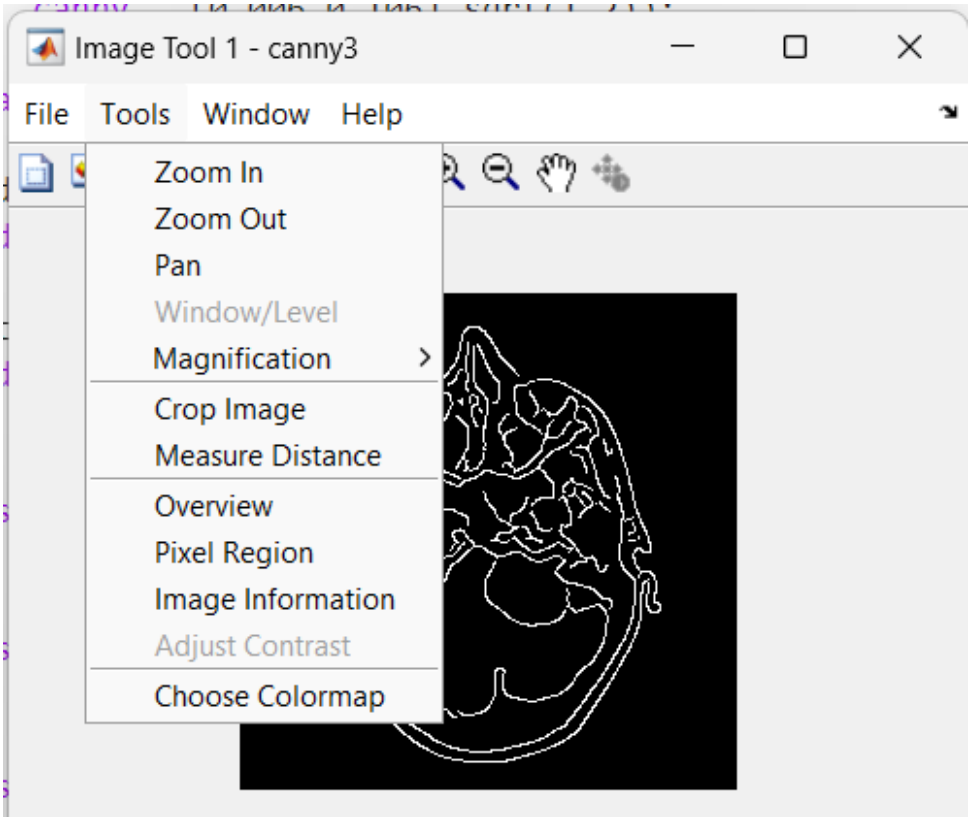
- **Relación de umbrales:** Según la documentación de OpenCV, una práctica común es usar una relación de 1:3 entre el umbral bajo y el umbral alto.
- **Uso de la mediana de la imagen:** Otra técnica consiste en calcular la mediana de los valores de intensidad de la imagen y usarla para definir los umbrales. Por ejemplo, puedes definir el umbral bajo como el 50% de la mediana y el umbral alto como el 150% de la mediana.
- **Método de Otsu:** Este método calcula de manera automática un umbral óptimo basándose en los histogramas de intensidad de la imagen.
- **Prueba y error:** Puedes experimentar con diferentes valores de umbrales y ver cuál produce los mejores resultados para tu caso específico.

En la imagen 5 hay que tomar en cuenta que nos interesa solamente el área del cerebro y el tumor, por esto un umbral amplio de máximo y mínimo nos asegura que todos los bordes queden en la imagen final, además que con este método podemos eliminar ruido, por eso considero que la mejor imagen es la quinta, siendo canny con **umbrales muy abiertos**.

2. Visualice los resultados obtenidos con los dos filtros de forma óptima, aplicando la técnica de modificación del contraste (práctica 1) que le parezca más adecuada.

Considero que al recibir del método Edge una imagen binaria utilizar imadjust o modificar el contraste desde imtool no generaran cambios óptimos, esto es porque al trabajar con una imagen binaria hace referencia a que todos los valores son 0 o 1, dejando que el modificar el contraste sea imposible sin arruinar la imagen.

Igualmente se intento modificar la imagen desde imtool, pero efectivamente no tiene lógica modificar el contraste de una imagen binaria, ya que esto solo podría cambiar todos los valores a 1 o 0, lo cual no tiene sentido.



3. Indique el método que obtenga mejor la información deseada, en el sentido de que la zona de interés está definida con claridad, desde su punto de vista subjetivo.

Considero que el método **canny** muestra muy bien los bordes sin dejar manchas/marcas en medio de la masa cerebral o el tumor, dejando más claro el borde y dándonos una perspectiva diferente a la imagen inicial, donde se pierde un poco el contraste que diferenciaba el tumor de la masa cerebral, pero ahora se pueden apreciar totalmente sus bordes muy claramente.

Creo que cada filtro será más útil dependiendo la imagen, el interés en dicha imagen, etc. Pero considero que una parte importante que nos puede proporcionar los filtros es otro punto de vista/simplificación de la imagen, ya que esto nos deja mucha flexibilidad a poder automatizar observaciones que normalmente serían un dolor de cabeza para un ser humano.

3. Guardar todos los resultados de imágenes finales obtenidas para la memoria de prácticas. Use la función imwrite(…).
    1. En formato png, con el mismo tipo de datos que la imagen original.
    2. En formato jpg, tras reescalar la imagen al tipo uint8 (256 niveles de gris).
    3. Adjuntar un fichero comprimido a la memoria con los ficheros de imagen obtenidos.

```
%% 3 guardar las imagenes
imwrite(f_paso_bajo,"Filtro_Paso_Bajo.png");
imwrite(sobel_Ajustado,"sobel_Ajustado.png");
imwrite(prewitt_ajustada2,"Filtro_prewitt.png");
imwrite(fg,"Filtro_Gaussiano.png");
imwrite(fl_ajustada,"filtro_laplaciano.png");
imwrite(flog_ajustada,"filtro_logaritmico.png");
imwrite(imagen_manual,"Edge_sobel_manual.png");
imwrite(canny4,"Canny_umbrales_amplios.png");
```

```
%% 3 uint8( y jpg
imwrite(f_paso_bajo,"Filtro_Paso_Bajo.jpg");
imwrite(uint8(sobel_Ajustado),"sobel_Ajustado.jpg");
imwrite(uint8(prewitt_ajustada2),"Filtro_prewitt.jpg");
imwrite(uint8(fg),"Filtro_Gaussiano.jpg");
imwrite(uint8(fl_ajustada),"filtro_laplaciano.jpg");
imwrite(uint8(flog_ajustada),"filtro_logaritmico.jpg");
imwrite(uint8(imagen_manual),"Edge_sobel_manual.jpg");
imwrite(uint8(canny4),"Edge_Canny_umbrales_amplios.jpg");
```

Las imágenes serán enviadas en una carpeta zip junto a este documento, igual se recomienda usar el codigo de Matlab donde en verdad se puede probar y ver mejor TODAS las imágenes, no solamente las mejores, también otras muy importantes que sirvan para comparar.

Conclusiones:

En esta práctica de laboratorio, hemos explorado la utilización y programación de filtros lineales espaciales para imágenes biomédicas. Hemos trabajado con una gran variedad de algoritmos y técnicas, incluyendo el uso de filtros paso bajo, Sobel, Prewitt, gaussiano, laplaciano y logaritmo de una gaussiana (LoG).

Hemos aprendido a utilizar la función imfilter() de Matlab y a seleccionar los parámetros adecuados para obtener los mejores resultados. Además, hemos programado nuestro propio algoritmo de filtrado, lo que nos ha permitido entender en profundidad cómo funcionan estos procesos.

En la segunda parte de la práctica, hemos aplicado técnicas de detección de bordes utilizando filtros de Sobel y Canny. Hemos aprendido a optimizar los parámetros de estos algoritmos y a visualizar los resultados de manera efectiva.

En general, esta práctica nos ha proporcionado una comprensión sólida de cómo funcionan los filtros lineales espaciales y cómo se pueden utilizar para mejorar la visualización y el análisis de imágenes biomédicas. Hemos adquirido habilidades valiosas que serán de gran utilidad en nuestra futura carrera en el campo de la imagen biomédica.

Sin embargo, también hemos aprendido que no existe un enfoque único para todos los casos. Los parámetros óptimos pueden variar dependiendo de la imagen y de lo que se quiera lograr. Por lo tanto, es esencial experimentar y ajustar estos parámetros para obtener los mejores resultados.

En resumen, esta práctica ha sido una experiencia de aprendizaje valiosa y enriquecedora. Estamos deseando aplicar lo que hemos aprendido en futuros proyectos y explorar aún más el fascinante campo de la imagen biomédica. . Así mismo, tras esta práctica he llegado a ser consciente de que el tratamiento de imágenes es más complicado de lo que esperaba, y que realmente desarrollar más tecnología respecto a esta rama de la bioingeniería va a ser crucial para detectar un montón de enfermedades con mucha mayor facilidad.

Bibliografía:

R.C. Gonzalez, R.E. Woods; Digital Image Processing, 3ed, Prentice-Hall 2007. R.C. Gonzalez, R.E. Woods, S.L. Eddins; Digital Image Processing using Matlab, 2ed, PrenticeHall 2009. W.K. Pratt; Digital Image Processing, 4ed, Wiley 2007.

Practica de laboratorio y material de estudio de Ingeniería de la Salud en la materia de imágenes biomédicas, material proporcionado por Enrique Nava Baro y Ignacio Rodríguez Rodríguez

Codigo:

```
%Practica2 imagenesiMAGENES BIOMEDICAS
%Borramos las variables cargadas y cargamos la imagen
clear variables
clc
close all

%Cargamos la imagen y la ajustamos automaticamente
imagen=dicomread('im5');
imagen_ajustada=imadjust(imagen);
figure(1)
imshow(imagen_ajustada) %Visualizacion de la imagen ajustada sin filtro
%%
h1=fspecial("average",5); %Filtro paso bajo 5x5
f_paso_sinajuste=imfilter(imagen,h1);
f_paso_bajo=imadjust(f_paso_sinajuste);
f_paso_bajo2=imfilter(imagen_ajustada,h1);

%comparacion
figure('Name', 'Visualización paso bajo 5x5')
subplot(1,3,1);
imshow(imagen_ajustada)
title('Imagen Ajustada sin filtro')
subplot(1,3,2);
imshow(f_paso_bajo)
title('Filtro paso bajo y luego ajustada')
subplot(1,3,3);
imshow(f_paso_bajo2)
title('Imagen ajustada con filtro paso bajo')

%1buscar la forma óptima manualmente
%f_paso_sinajuste=imfilter(imagen,h1);
%imtool(f_paso_sinajuste)
%f_paso_ajustado=imadjust(f_paso_sinajuste, [50000/65536 60000/65536],[0,1]);
%%
%Filtro sobel
h2=fspecial('sobel');
sobel_simple=imfilter(imagen, h2);
sobel_simple=imadjust(sobel_simple);
%imshow(imfilter(imagen_ajustada, h2))

%Filtro sobel doble barrido
B = double(imagen_ajustada);
fs = fspecial('sobel'); %vertical
%matriz transpuesta'
fs2 = fs'; %Filtro Sobel, tenemos que hacer filtrado horizontal y vertical
sobel1 = imfilter(B, fs);
sobel2 = imfilter(B, fs2);
%Calculamos el módulo
sobel = sqrt((sobel1.^2) + (sobel2.^2));
sobel = uint16(sobel);

%Filtro sobel doble barrido imagen sin ajustar (mejor hacer un metodo
%aparte en caso de querer seguir trabajando con el sobel)
D=double(imagen);
sobel11 = imfilter(D, fs);
sobel22 = imfilter(D, fs2);
%Calculamos el módulo
```

```
sobel1 = sqrt((sobel11.^2) + (sobel22.^2));
sobel1 = uint16(sobel1);
sobel_Ajustado=imadjust(sobel1);

%comparacion
figure('Name', 'Visualización filtro sobel')
subplot(3,2,1);
imshow(imagen_ajustada)
title('Imagen Ajustada sin filtro')
subplot(3,2,3);
imshow(sobel_simple)
title('Filtro sobel y luego ajustada')
subplot(3,2,4);
imshow(imfilter(imagen_ajustada, h2))
title('Imagen ajustada con filtro sobel')
subplot(3,2,5);
imshow(sobel_Ajustado)
title('filtro sobel (vert y hori) + ajuste') %vertical y horizontal
subplot(3,2,6);
imshow(sobel)
title('Imagen ajustada con filtro sobel (vert y hori)')

%%
%3Filtro de Prewitt
h3=fspecial('prewitt');
prewitt_simple=imfilter(imagen, h3);
prewitt_simple=imadjust(prewitt_simple);
prewitt_simple2=imfilter(imagen_ajustada, h3);

%prewitt en todas las direcciones
%B = double(imagen_ajustada);  esto ya fue cargado anteriormente
fp = fspecial('prewitt');
fp2 = fp';
prewitt1 = imfilter(B, fp);
prewitt2 = imfilter(B, fp2);
prewitt_to = sqrt((prewitt1.^2) + (prewitt2.^2));
prewitt_ajustada1 = uint16(prewitt_to);

%Recomiendo hacer una funcion pero por tiempo lo copie y pegue en otro
%lado, para subirlo lo traje aqui
%D=double(imagen);
prewitt1 = imfilter(D, fp);
prewitt2 = imfilter(D, fp2);
prewitt_sinajuste = sqrt((prewitt1.^2) + (prewitt2.^2));
prewitt_sinajuste = uint16(prewitt_sinajuste);
prewitt_ajustada2 = imadjust(prewitt_sinajuste);
%imshow(prewitt)

%comparacion
figure('Name', 'Visualización filtro Prewitt')
subplot(3,2,1);
imshow(imagen_ajustada)
title('Imagen Ajustada sin filtro')
subplot(3,2,3);
imshow(prewitt_simple)
title('Filtro Prewitt y luego ajustada')
subplot(3,2,4);
imshow(prewitt_simple2)
title('Imagen ajustada con filtro prewitt')
```

```
subplot(3,2,5);
imshow(prewitt_ajustada2)
title('filtro prewitt (vert y hori) + ajuste') %vertical y horizontal
subplot(3,2,6);
imshow(prewitt_ajustada1)
title('Imagen ajustada con filtro prewitt (vert y hori)')
%%
%4Filtro gaussiano de 5x5, con sigma=1.
h4 = fspecial('gaussian', [5 5], 2);
fg = imfilter(imagen_ajustada, h4);

fg_sinajustar = imfilter(imagen, h4);
fg_ajustada = imadjust(fg_sinajustar);

%comparacion
figure('Name', 'Visualización filtro gaussiano 5x5')
subplot(1,3,1);
imshow(imagen_ajustada)
title('Imagen Ajustada sin filtro')
subplot(1,3,2);
imshow(fg)
title('Filtro gaussiano y luego ajustada')
subplot(1,3,3);
imshow(fg_ajustada)
title('Imagen ajustada con filtro gaussiano')
%%
%5Filtro Laplaciano
h5 = fspecial ('laplacian');
fl = imfilter(imagen_ajustada, h5);

fl_sinajustar = imfilter(imagen, h5);
fl_ajustada = imadjust(fl_sinajustar);

%comparacion
figure('Name', 'Visualización filtro laplaciano')
subplot(1,3,1);
imshow(imagen_ajustada)
title('Imagen Ajustada sin filtro')
subplot(1,3,2);
imshow(fl_ajustada)
title('Filtro laplaciano y luego ajustada')
subplot(1,3,3);
imshow(fl)
title('Imagen ajustada con filtro laplaciano')
%%

% 6Filtro logaritmo de una gaussiana (LoG) de 5x5, con sigma=1.
h6 = fspecial("log",[5 5], 1);
flog = imfilter(imagen_ajustada, h6);

flog_sinajustar = imfilter(imagen, h6);
flog_ajustada = imadjust(flog_sinajustar);

%comparacion
figure('Name', 'Visualización filtro logaritmo de una gaussiana (LoG)')
subplot(1,3,1);
imshow(imagen_ajustada)
title('Imagen Ajustada sin filtro')
subplot(1,3,2);
```

```
imshow(flog_ajustada)
title('Filtro LoG y luego ajustada')
subplot(1,3,3);
imshow(flog)
title('Imagen ajustada con filtro LoG')
%%
figure('Name', 'Todos los Filtros')
subplot(2,3,1);
imshow(imagen_ajustada)
title('Imagen Ajustada')
subplot(2,3,2);
imshow(f_paso_bajo)
title('Filtro Paso Bajo')
subplot(2,3,3);
imshow(sobel_Ajustado)
title('Filtro Sobel')
subplot(2,3,4);
imshow(prewitt_ajustada2)
title('Filtro prewitt')
subplot(2,3,5);
imshow(fg)
title('Filtro Gaussiano')
subplot(2,3,6);
imshow(fl_ajustada)
title('filtro laplaciano')

figure(7)
imshow(flog_ajustada)
title('filtro logaritmo')

%% Apartado 2
%Utilice la funcion edge en los siguientes algoritmos
[imagen_binaria,thresh]= edge(imagen, 'sobel'); %usa sobel por defecto
thresh; %ver umbral automatico
%figure(1)
%imshow(imagen_binaria)

[imagen_manual]= edge(imagen, 'sobel',0.00035); %usa sobel por defecto
%figure(2)
%imshow(imagen_manual)

%comparacion
figure('Name', 'Visualización para función edge')
subplot(1,3,1);
imshow(imagen_ajustada)
title('Imagen Ajustada')
subplot(1,3,2);
imshow(imagen_binaria)
title('Edge con sobel usando umbral automatico')
subplot(1,3,3);
imshow(imagen_manual)
title('Edge con sobel usando umbral manual')
%%
[imagen_automatica,thresh]= edge(imagen, 'canny');
thresh %ver umbral automatico
%imshow(imagen_automatica)

canny1= edge(imagen, 'canny', [0.0300 0.0800],sqrt(3));
canny2= edge(imagen, 'canny', [0.0300 0.0800],sqrt(0.5));
```

```
canny3= edge(imagen, 'canny', [0 0.1800],sqrt(3));
canny4= edge(imagen, 'canny', [0.006 0.106],sqrt(1.2));

figure('Name', 'Visualización canny')
subplot(3,2,1);
imshow(imagen_ajustada)
title('Imagen Ajustada')
subplot(3,2,2);
imshow(imagen_automatica)
title('Imagen ajustada automaticamente')
subplot(3,2,3);
imshow(canny1)
title('Canny con un sigma elevado')
subplot(3,2,4);
imshow(canny2)
title('Canny con un sigma bajo')
subplot(3,2,5);
imshow(canny3)
title('canny con unos umbrales muy abiertos')
subplot(3,2,6);
imshow(canny4)
title('canny con unos umbrales muy cerrados')
imtool(canny5)

%% 3 guardar las imagenes
imwrite(f_paso_bajo,"Filtro_Paso_Bajo.png");
imwrite(sobel_Ajustado,"sobel_Ajustado.png");
imwrite(prewitt_ajustada2,"Filtro_prewitt.png");
imwrite(fg,"Filtro_Gaussiano.png");
imwrite(fl_ajustada,"filtro_laplaciano.png");
imwrite(flog_ajustada,"filtro_logaritmico.png");
imwrite(imagen_manual,"Edge_sobel_manual.png");
imwrite(canny4,"Canny_umbrales_amplios.png");
%% 3 uint8( y jpg
imwrite(f_paso_bajo,"Filtro_Paso_Bajo.jpg");
imwrite(uint8(sobel_Ajustado),"sobel_Ajustado.jpg");
imwrite(uint8(prewitt_ajustada2),"Filtro_prewitt.jpg");
imwrite(uint8(fg),"Filtro_Gaussiano.jpg");
imwrite(uint8(fl_ajustada),"filtro_laplaciano.jpg");
imwrite(uint8(flog_ajustada),"filtro_logaritmico.jpg");
imwrite(uint8(imagen_manual),"Edge_sobel_manual.jpg");
imwrite(uint8(canny4),"Edge_Canny_umbrales_amplios.jpg");
%%
imshow(imadjust(prewitt_filter(imagen)))
%Funciones
function [Gx, Gy, im_mag] = prewitt_filter(im)
%lee la imagen
imagen=double(im);
[filas, columnas]=size(imagen);
Gx=zeros(filas,columnas);
Gy=zeros(filas,columnas);
% Define los kernels Prewitt
prewitt_x = [-1 0 1; -1 0 1; -1 0 1];
prewitt_y = [-1 -1 -1; 0 0 0; 1 1 1];
for i=2:filas-1
    for j=2: columnas-1
        Gx(i,j) =imagen(i,j)* prewitt_x (2,2)+ imagen(i,j+1)* prewitt_x (2,3)
+ ...
```

```
        imagen(i-1,j+1)* prewitt_x (1,3)+ imagen(i-1,j)* prewitt_x (1,2) +
...
        imagen(i-1,j-1)* prewitt_x (1,1)+ imagen(i,j-1)* prewitt_x (2,1) +
...
        imagen(i+1,j-1)* prewitt_x (3,1)+ imagen(i+1,j)* prewitt_x (3,2)+
...
        imagen(i+1,j+1)*prewitt_x(3,3);
        %con el eje y
        Gy(i,j)=imagen(i,j)* prewitt_y (2,2)+ imagen(i,j+1)* prewitt_y (2,3)
+ ...
        imagen(i-1,j+1)* prewitt_y (1,3)+ imagen(i-1,j)* prewitt_y (1,2) +
...
        imagen(i-1,j-1)* prewitt_y (1,1)+ imagen(i,j-1)* prewitt_y (2,1) +
...
        imagen(i+1,j-1)* prewitt_y (3,1)+ imagen(i+1,j)* prewitt_y (3,2)+
...
        imagen(i+1,j+1)*prewitt_y(3,3);

    end
end
% Calcula la magnitud del gradiente
im_mag = sqrt(double(Gx).^2 + double(Gy).^2);
im_mag  = uint16(im_mag );
end

function r = fspecial_propio(imagen, filtro)
s = size(imagen);
r = zeros(s);
for i = 2:s(1)-1
    for j = 2:s(2)-1
        temp = imagen(i-1:i+1,j-1:j+1) .* filtro;
        r(i,j) = sum(temp(:));
    end
end
end
```

Filtros:

Valores de cada filtro:

h1 =

1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1

h2 =

1	2	1
0	0	0
-1	-2	-1

h3 =

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

h4 =

0.0232	0.0338	0.0383	0.0338	0.0232
0.0338	0.0492	0.0558	0.0492	0.0338
0.0383	0.0558	0.0632	0.0558	0.0383
0.0338	0.0492	0.0558	0.0492	0.0338
0.0232	0.0338	0.0383	0.0338	0.0232

h5 =

0.1667	0.6667	0.1667
0.6667	-3.3333	0.6667
0.1667	0.6667	0.1667

h6 =

0.0239	0.0460	0.0499	0.0460	0.0239
0.0460	0.0061	-0.0923	0.0061	0.0460
0.0499	-0.0923	-0.3182	-0.0923	0.0499
0.0460	0.0061	-0.0923	0.0061	0.0460
0.0239	0.0460	0.0499	0.0460	0.0239