TRABAJO 1.

VC. FILTRADO Y MUESTREO.

Alberto Armijo Ruiz. 4º GII.

Ejercicio 1.

Apartado A.

Para la realización de esta función utilizaremos la librería matplotlib, utilizando la función subplot para dividir la imagen en subespacios donde dibujaremos las imágenes.
Los argumentos de la función son un vector de imágenes y un vector con los títulos de esas imágenes.

Lo primero que se hace es comprobar si la raíz del número de imágenes es entero , si esto es así, construiremos un subespacio de tamaño NxN, sino construiremos un subespacio de tamaño N+1xN+1.

Lo siguiente que se hace es recorrer el vector de imágenes, pintándolas en su correspondiente espacio dentro de la imagen. Además, se le añade su título.

Una vez hecho esto, se llama a la función show() para mostrar el conjunto de imágenes.

Apartado B.

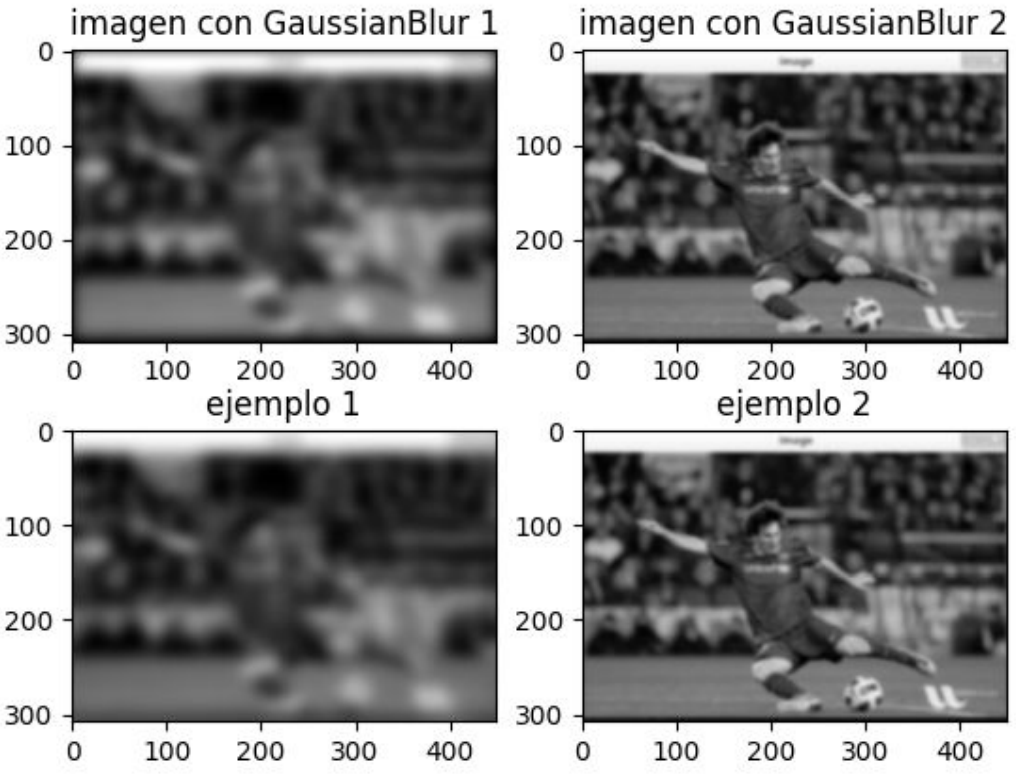
En este apartado, utilizaremos las funciones getGaussianKernel y filter2D. A la función le pasaremos como argumentos la imagen original, el valor de sigma y el tipo de borde que queremos utilizar para la función filter2D.

Con el valor de sigma, podemos calcular el tamaño de la máscara, este siempre será igual a 6*sigma+1. Teniendo estos valores, llamaremos a la función getGuassiankernel, que nos devolverá un kernel gaussiano separado. Este kernel es de tamaño (size,1).

Para aplicar el kernel completo a la imágen, debemos múltiplicar este mismo kernel por su traspuesta, ya que el kernel gaussiano es simétrico.

Una vez hemos obtenido el kernel guassiano completo, utilizaremos la función filter2D para realizar la convolución. Por último, devolveremos la imagen obtenida.

Los resultados obtenidos son los siguientes:



Como se puede apreciar, para valores mayores de sigma, se consigue una distorsión mayor en la imagen resultante.

Apartado C.

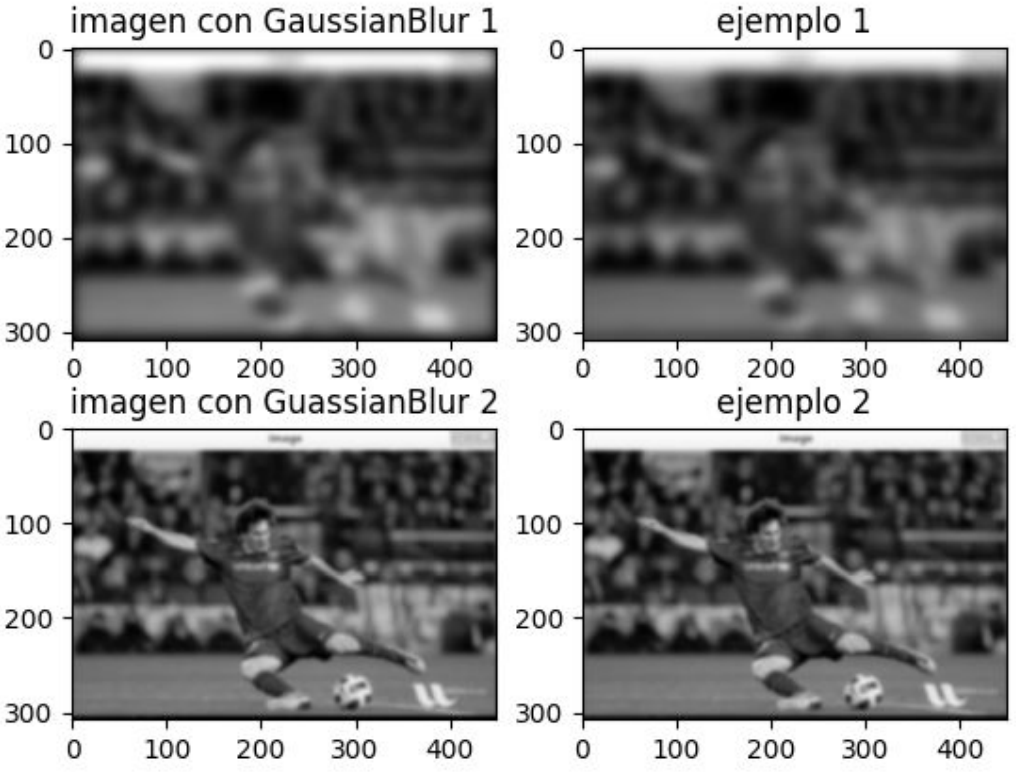
Para la realización de este apartado utilizaremos las mismas funciones que en el apartado anterior. Los argumentos de esta función son los mismos que en la anterior.

Al igual que antes, calcularemos el tamaño de la máscara y el kernel separado; pero ahora no necesitaremos utilizar el kernel completo.

Una vez calculado esto, obtendremos el número de filas y de columnas accediendo al atributo shape de la imagen.

Lo siguiente que haremos será aplicar el kernel separado a todas la filas de la imagen, después, aplicaremos el kernel separado traspuesto a todas las columnas de la imagen. Para ello utilizaremos la función filter2D. Por último devolveremos la imagen resultante.

Los resultados obtenidos son los siguientes:



Al igual que en el apartado anterior cuanto mayor sea el valor de sigma, más difuminada se mostrará la imagen como resultado.

Apartado D.

Para el desarrollo de esta función, utilizaremos las funciones getDerivKernels y filter2D. Los argumentos de esta función son la imagen sobre la que se realizarán los cambios, el tamaño del kernel, los índices de la derivada sobre y y sobre x y el tipo de borde que se quiere utilizar.

Lo primero que haremos será llamar a la función getDerivKernels especificando los índices de las derivada (ej: si el índice en x es 1, se calculará la derivada primera de la gaussiana). Esta función nos devolverá los kernels derivados en x e y. Dependiendo de los índices, lo aplicaremos a x o a y. Si derivX=1, se hará la derivada sobre el eje X, si derivY=1, se hará la derivada sobre el eje Y.

Lo siguiente que haremos es aplicar estos kernels a la imagen para trasformarla. Esto se realizará al igual que en el apartado anterior, pero esta vez utilizaremos los kernels obtenidos por la función getDerivKernels().

Los resultados obtenidos son los siguientes:

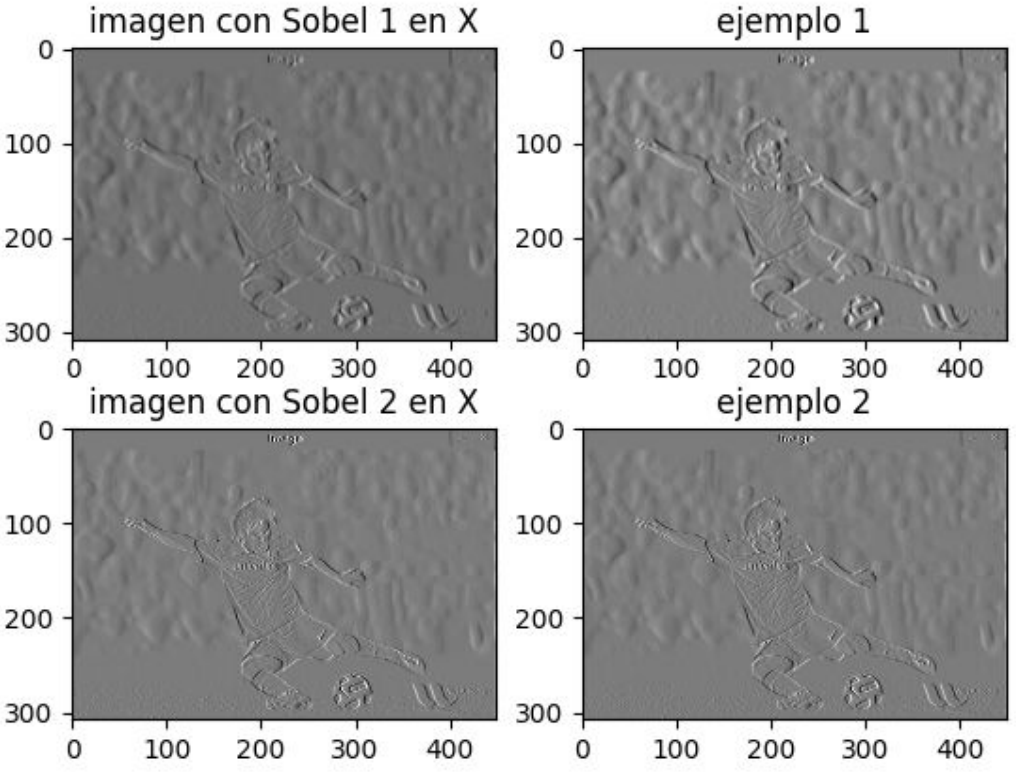


imagen con Sobel 1 en X
ejemplo 1
imagen con Sobel 2 en X
ejemplo 2
0
100
200
300
400

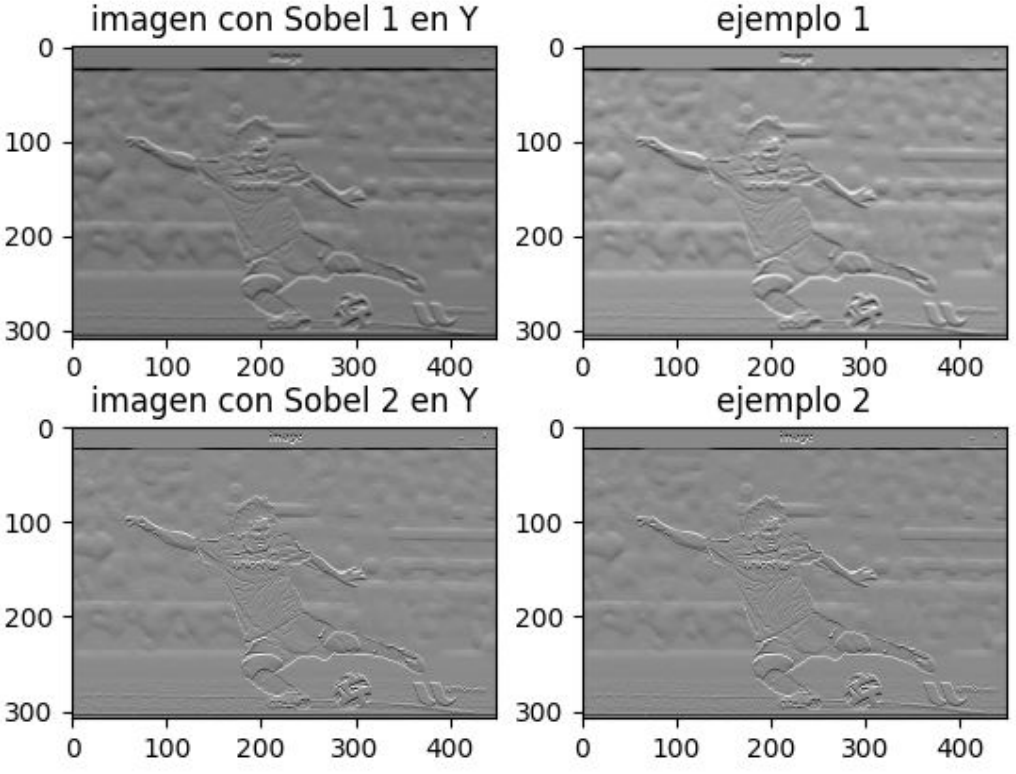


imagen con Sobel 1 en Y
ejemplo 1
imagen con Sobel 2 en Y
ejemplo 2
0
100
200
300
400

Como se puede apreciar, cuanto mayor es el valor de sigma, mayor es el tamaño de la máscara obtenida, y se obtiene un resultado con un nivel de detalle mayor.

Apartado E.

Para el desarrollo de esta función utilizaremos la función anteriormente implementada, lo único que tendremos que hacer, es multiplicar el coeficiente en x o en y por dos, para obtener la segunda derivada en X o en Y.

Los resultados obtenidos son los siguientes:

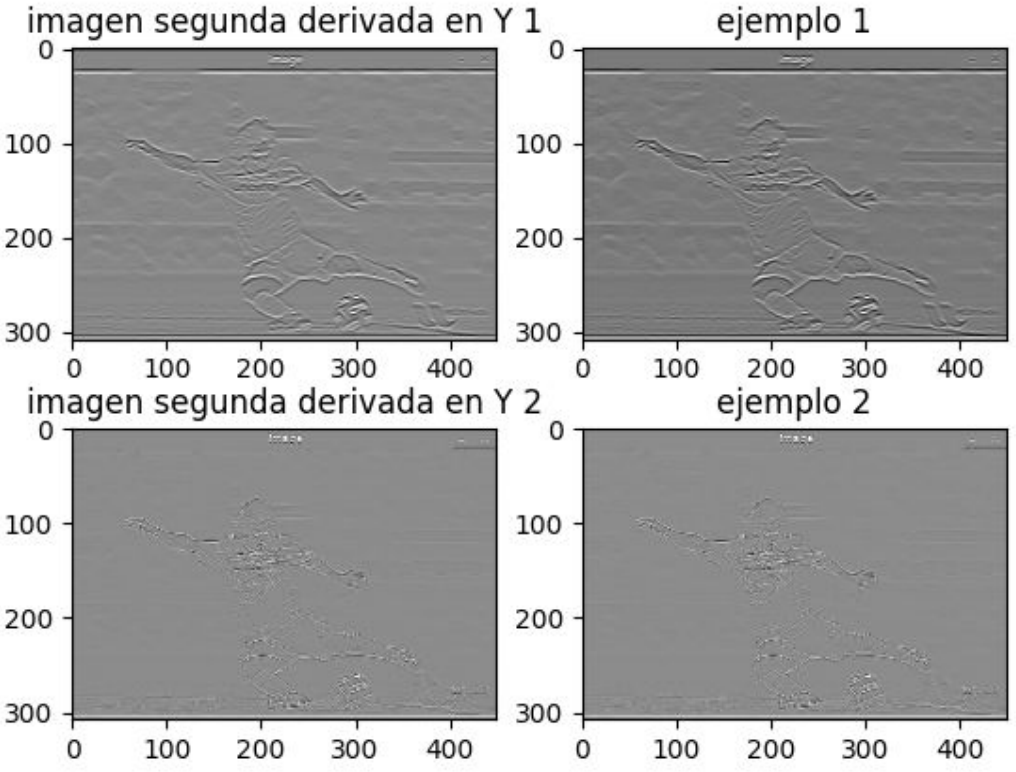


imagen segunda derivada en Y 1
ejemplo 1
imagen segunda derivada en Y 2
ejemplo 2
0
100
200
300
400

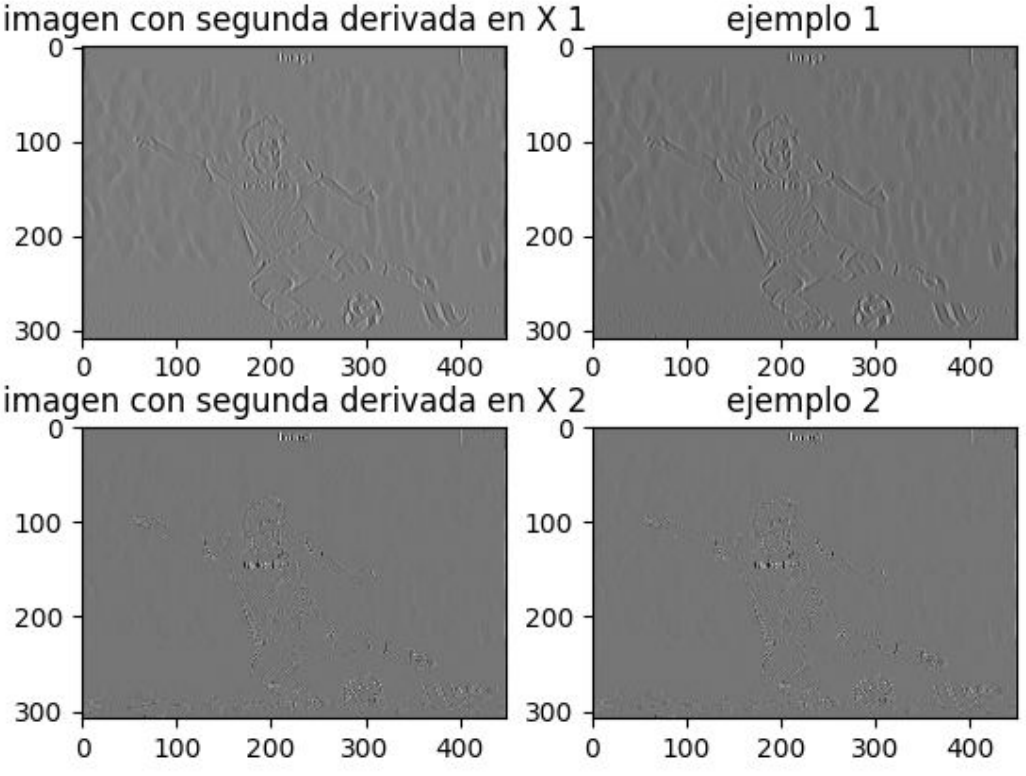


imagen con segunda derivada en X 1
ejemplo 1
imagen con segunda derivada en X 2
ejemplo 2
0
100
200
300
400

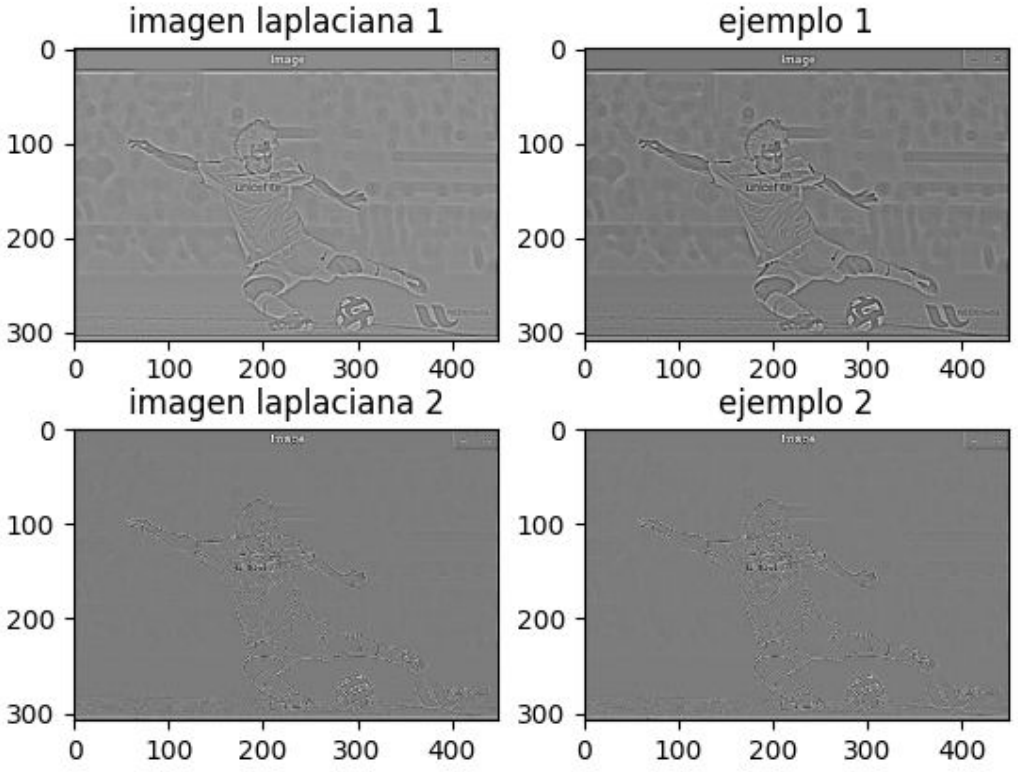
Como se puede apreciar, cuanto mayor es el tamaño de la máscara, y por tanto el de sigma, mayor es el nivel de detalle que se obtiene con el filtro.

Apartado F.

Para obtener la laplaciana, deberemos obtener antes las derivada segundas sobre X y sobre Y, justamente eso es lo que hemos implementado en la función anterior. Como argumentos de la función tenemos la imagen a la cual se van a hacer las transformaciones, el tamaño de la máscara, y el tipo de borde que se quiere utilizar.

Una vez hemos calculado ambas derivadas, la laplaciana se obtiene sumando ambas imágenes.

Los resultados obtenidos son los siguientes:



A mayor sigma, mayor es el tamaño de la máscara, y mayor es el nivel de detalle que obtiene la laplaciana.

Apartado G.

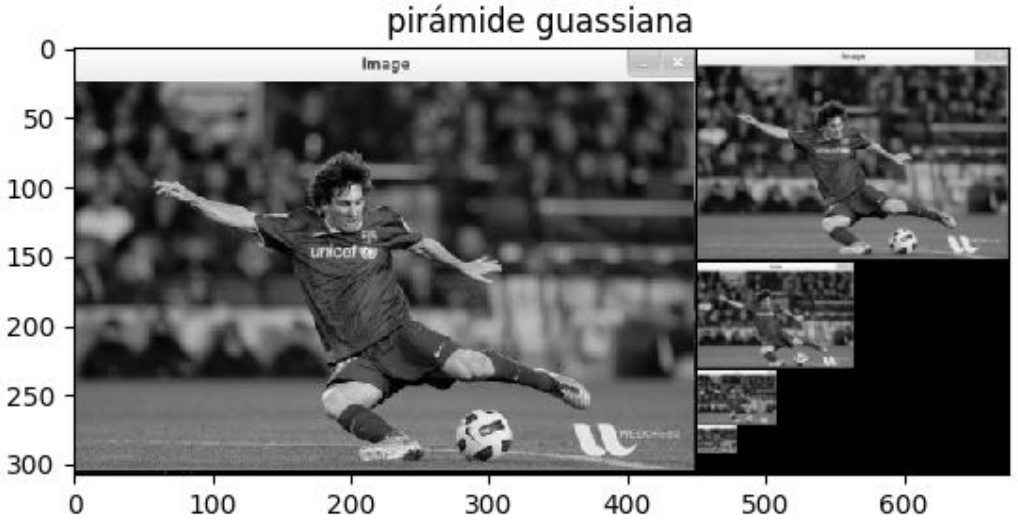
Para crear la pirámide guassiana, utilizaremos la función pyrDown, que nos permite aplicar un filtro gaussiano y reducir el tamaño de la imagen.
Esta función tiene como argumentos el tipo de borde de las imágenes y la imagen con la cual se va a crear la pirámide.

Lo primero que haremos en esta función es crear una imagen de tamaño mayor que la original donde introduciremos todas la imágenes que formarán la pirámide. Para ello calculamos el número de filas y de columnas, en mí caso, la imágen es de dimensiones (num_filas, 1.5*num_columnas+1); donde num_filas es el número de filas de la imagen original y num_columnas es el número de columnas de la imagen original.

Lo siguiente que haremos es pegar la imagen original en la imagen que de la pirámide. Después de esto reduciremos su tamaño con la función pyrDown, por defecto esta función reduce el tamaño de la función a la mitad, pero también se puede especificar el tamaño si se quiere.

Tras esto, pegaremos la imagen a la derecha de la imagen original, y debajo de otra imagen dependiendo del nivel en el que estemos.

Finalmente, devolveremos la imagen donde se encuentra la pirámide representada.
Los resultados obtenidos son los siguientes:



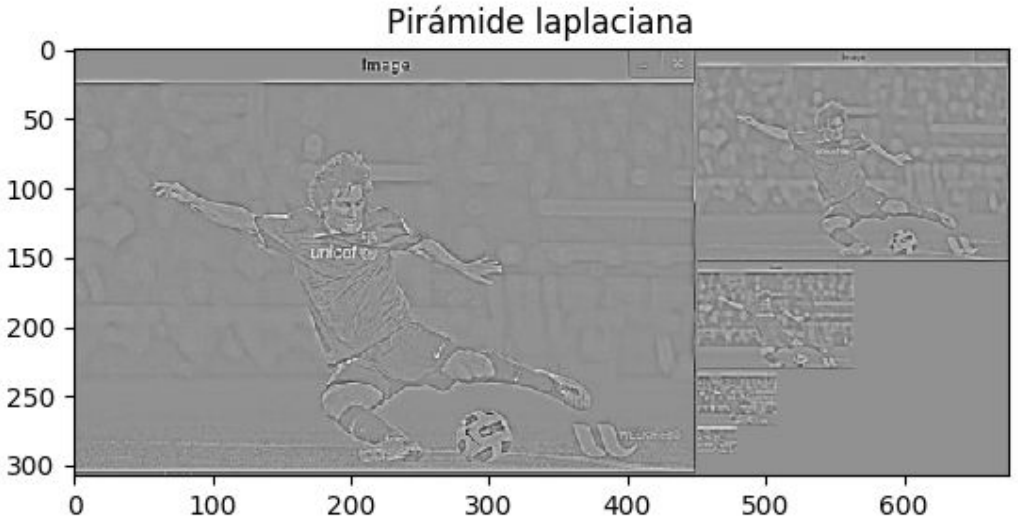
Apartado H.

En este apartado se realizará el mismo proceso que para la pirámide guassiana. Los argumentos de la función también son los mismos que para la pirámide gaussiana.

A diferencia de la pirámide gaussiana, debemos representar la laplaciana de la imagen en cada nivel, para ello, debermos obtener la diferencia entre la imagen de un nivel y la imagen aumentada de un nivel posterior.

Para realizar este proceso crearemos una nueva función que utilizará las funciones pyrUp, pyDown y substract. Con pyrDown reduciremos el tamaño de la imagen a la mitad y se aplicará un filtro gaussiano, con pyrUp duplicaremos el tamaño de la imagen y se aplicará otro filtro. Finalmente, utilizaremos la función substract para calcular la diferencia entre imágenes y devolveremos el resultado.

Los resultados obtenidos son los siguientes:



Ejercicio 2.

Apartado 1.

Esta función utiliza la función de convolución separada implementada en el ejercicio anterior para obtener las imágenes con frecuencias altas y bajas; también utiliza la función addWeighted(), que nos permite mezclar ambas imágenes especificando el porcentaje de cada una.

Como argumentos, tendremos cada una de las imágenes originales y la variable alpha, que indicará cuanto debe mostrarse la primera imagen en la imagen híbrida.

Lo primero que deberemos de hacer es crear un vector donde guardaremos las tres imágenes (híbrida, frec.altas, frec.bajas). Tras esto, calcularemos la frecuencias altas utilizando un valor de sigma bajo. Para calcular las frecuencias bajas, utilizaremos un valor de sigma alto.

Una vez calculadas ambas imágenes las añadimos al vector, y utilizamos la función addWeigthed para mezclar ambas imágenes.

Finalmente, introducimos la imagen híbrida en el vector y devolveremos el vector.

Apartado 2.

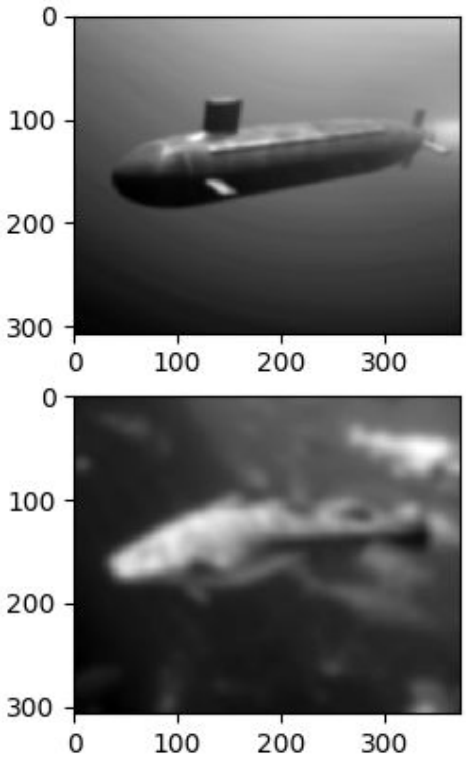
En este apartado, para mostrar las tres imágenes, se ha utilizado la librería gridSpec de matplotlib y la función subplot de pyplot.

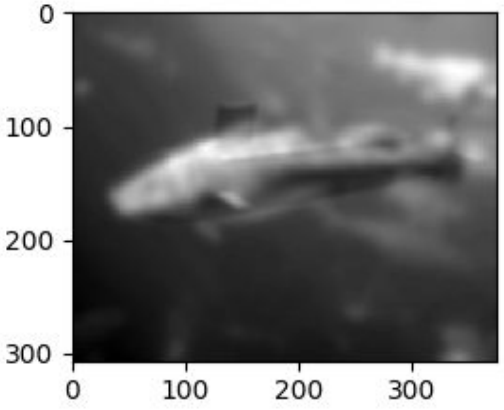
Con la librería gridSpec se ha creado un grid de 2x2. Después, con la función subplot se han asignado espacio a cada una de las imágenes. La imagen con frecuencias altas se coloca en la esquina superior izquierda del grid, la imagen con frecuencias bajas se coloca en la esquina inferior del grid; por último, la imagen híbrida se coloca en la parte derecha del grid.

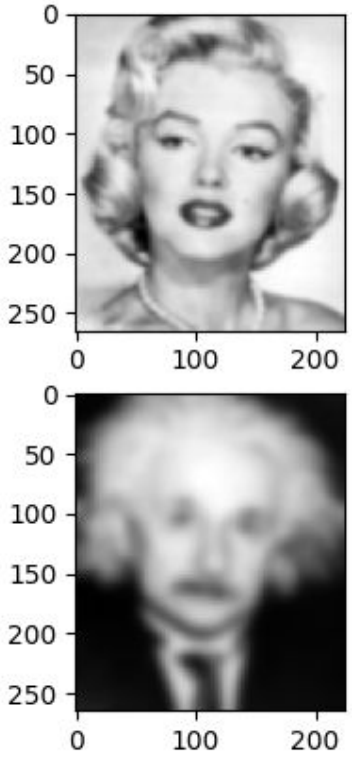
Finalmente, se muestra el resultado con la funcíon show().

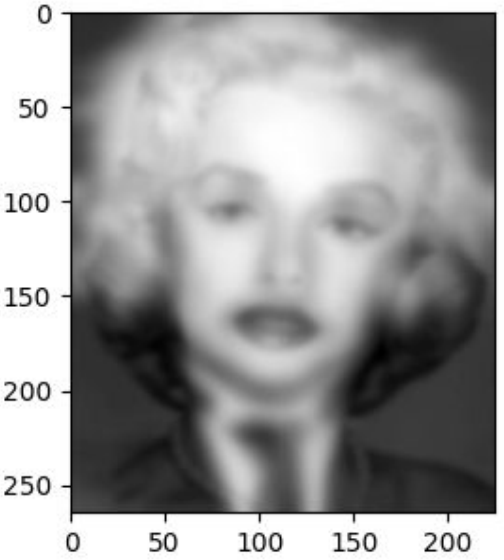
Apartado 3.

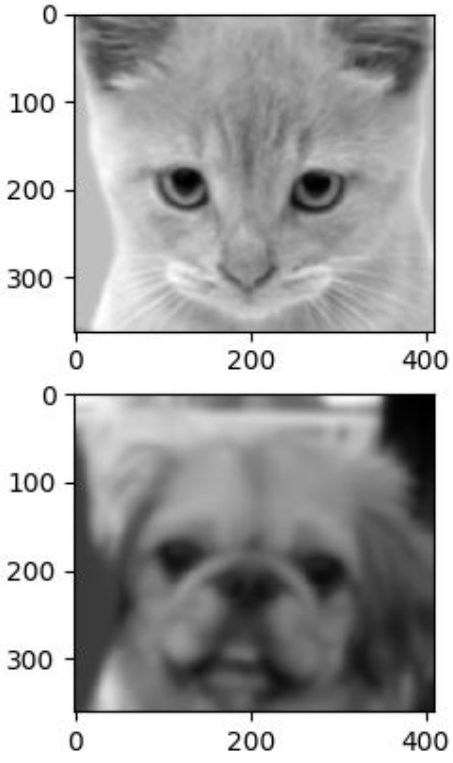
Algunos ejemplos son:

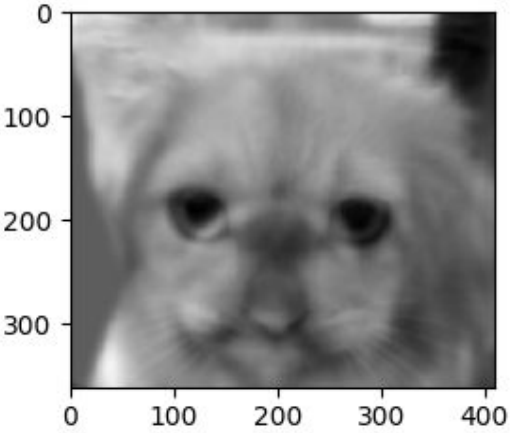





0
50
100
150
200
250
0
100
200
0
50
100
150
200
250
0
100
200


0
50
100
150
200
250
0
50
100
150
200


0
100
200
300
0
200
400
0
100
200
300
0
200
400


0
100
200
300
0
100
200
300
400