

Práctica 1: Filtrado y Muestreo

David Gil Bautista

DNI: 45925424M

Universidad de Granada — October 19, 2018

1 Funciones OpenCV

USANDO LAS FUNCIONES DE OPENCV : escribir funciones que implementen los siguientes puntos:

- 1.1 El cálculo de la convolución de una imagen con una máscara Gaussiana 2D (Usar GaussianBlur). Mostrar ejemplos con distintos tamaños de máscara y valores de sigma. Valorar los resultados.**

Para resolver este apartado debemos saber cómo funciona una máscara Gaussiana, y para ello mostraré el siguiente gráfico que nos ayudará a comprender mejor el concepto.

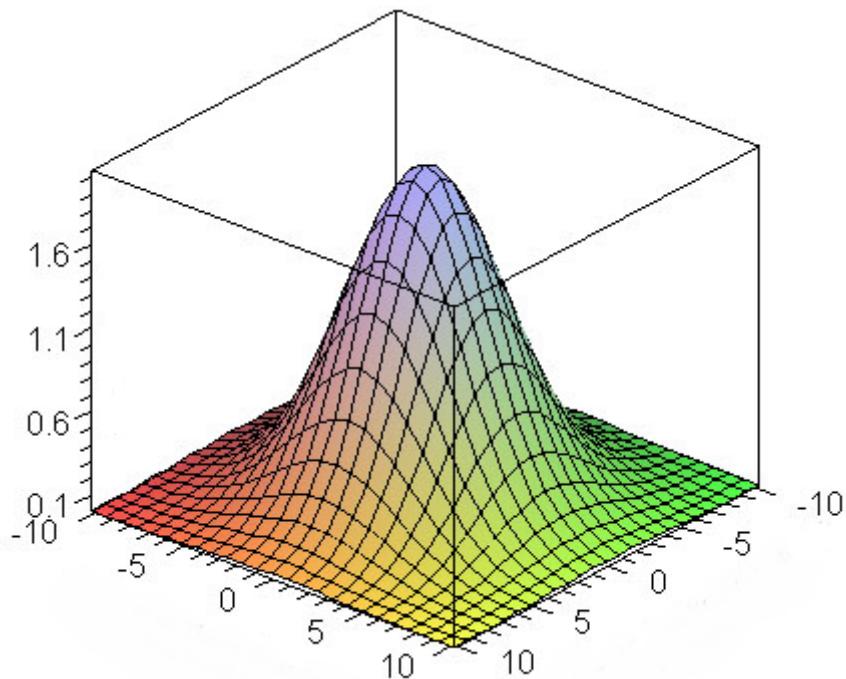


Figure 1: Campana de Gauss en 3D

Una máscara Gaussiana presenta un kernel que es una aproximación en dos dimensiones de una función Gaussiana.

$$h(u, v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{\sigma^2}} \quad (1)$$

En nuestro caso el kernel nos permitirá escoger el tamaño de la máscara con el que trabajaremos, y para ello probaremos con los siguientes tamaños 1, 3, 5 y 7 píxeles. Un tamaño de 5 píxeles quiere decir

que trabajaremos con el pixel central y dos a su izquierda y derecha, es decir, siendo x el pixel central y k el tamaño de la máscara, trabajaremos con las siguientes posiciones:

$$x \pm \frac{(k - 1)}{2} + 1 \quad (2)$$

La varianza, por definición es la desviación estandar al cuadrado. Esta desviación nos indica lo dispersos que están los elementos respecto de la media. Una función con una varianza alta hace que la función sea más plana y que los datos no estén tan cerca de la media. Sabiendo esto podemos decir que cuanto menor sea la varianza vamos a tener más ceros en el mismo kernel, por lo que estos datos serán menos relevantes.

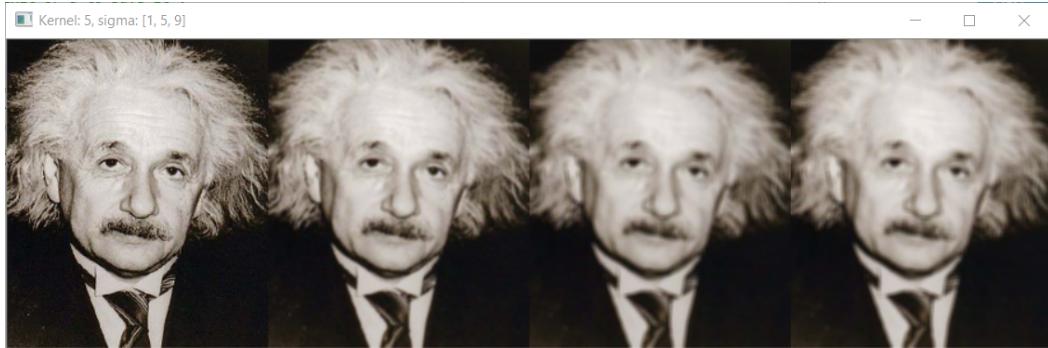


Figure 2: Kernel = 5, Sigma = 1,5,9

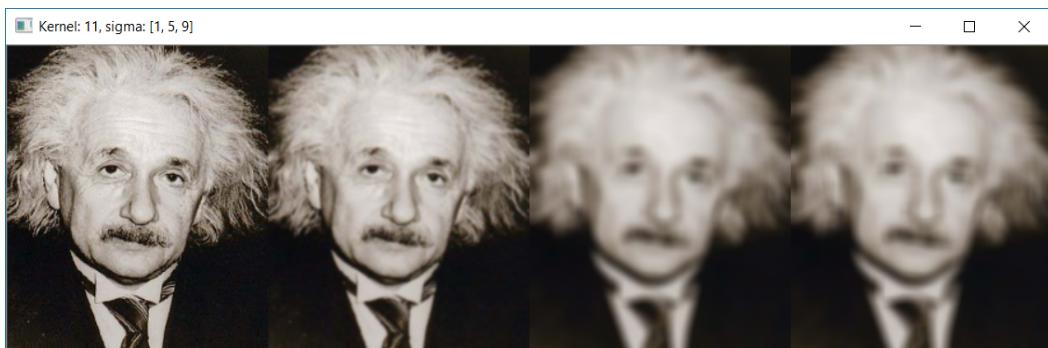


Figure 3: Kernel = 9, Sigma = 1,5,9

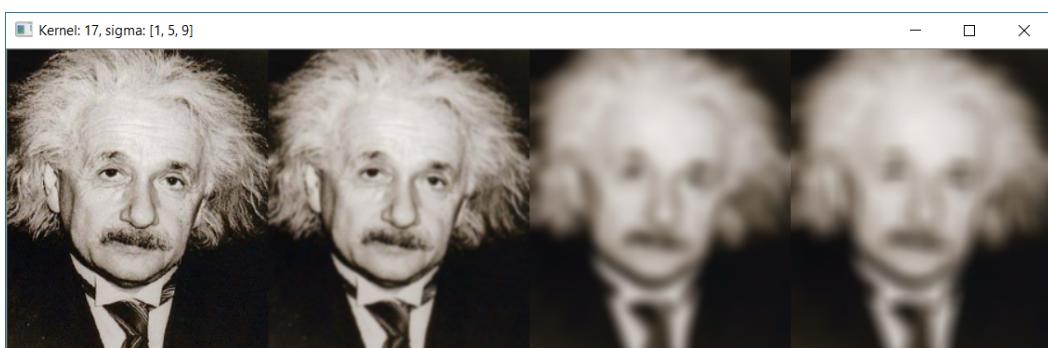


Figure 4: Kernel = 17, Sigma = 1,5,9

Como podemos ver en las imágenes mostradas, conforme aumentamos el tamaño del kernel y la varianza vamos obteniendo un desenfoque mayor.

1.2 Usar getDerivKernels para obtener las máscaras 1D que permiten calcular al convolución 2D con máscaras de derivadas. Representar e interpretar dichas máscaras 1D para distintos valores de sigma.

Para este apartado se usa la función getDerivKernels para obtener las máscaras de derivadas que podemos usar para calcular la convolución en 2D. Para ello he probado con distintos tamaños de máscara y he escogido la máscara de la derivada respecto a y para calcular la convolución en dos dimensiones y aplicar un filtro de alisamiento.

```
Using kernel: 1
Derived mask rows: [[ -0.5 ]
 [ 0. ]
 [ 0.5 ]]

Derived mask columns: [[1.]]
```

Figure 5: Tamaño de kernel = 1

Para un kernel de tamaño 1 obtenemos una convolución con un único dato de valor 1, y al aplicar el filtro a una imagen obtenemos otra imagen exactamente igual puesto que al evaluar los píxeles estos toman su mismo valor.

Aquí podemos ver la representación gráfica.

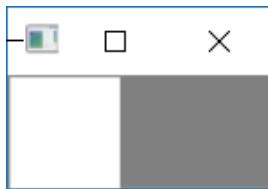


Figure 6: Convolución k=1

A medida que vamos cambiando el tamaño del kernel obtenemos una matriz en la que los valores se van atenuando más conforme se separan de la media.

En las siguientes imágenes podemos ver las máscaras 1D para un tamaño de kernel de 5 píxeles y las distintas representaciones de la convolución para otros tamaños adicionales.

```
Using kernel: 5
Derived mask rows: [[ -0.125 ]
 [-0.25 ]
 [ 0. ]
 [ 0.25 ]
 [ 0.125 ]]

Derived mask columns: [[ 0.0625 ]
 [ 0.25 ]
 [ 0.375 ]
 [ 0.25 ]
 [ 0.0625 ]]
```

Figure 7: Máscaras derivadas 1D con k=5

Para la representación de las siguientes convoluciones para tamaños de kernel de 3, 5 y 7 se ha multiplicado el resultado final por un número positivo para que el filtro se aprecie mejor.



Figure 8: Kernel = 3



Figure 9: Kernel = 5

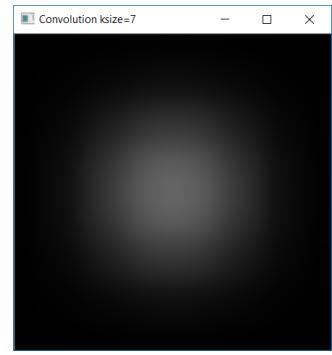


Figure 10:
Kernel = 7

Conforme aumentamos el tamaño del kernel podemos ver que el área de desenfoque se distribuye más y que los píxeles centrales a la media son menos relevantes que con un tamaño de máscara menor. De esta forma podemos ver gráficamente lo que hemos realizado en el apartado anterior donde para una misma varianza el tamaño de la máscara es proporcional al desenfoque que obtenemos.

1.3 Usar la función Laplacian para el cálculo de la convolución 2D con una máscara de Laplaciana-de-Gaussiana de tamaño variable. Mostrar ejemplos de funcionamiento usando dos tipos de bordes y dos valores de sigma: 1 y 3

Para este ejercicio hemos usado la función Laplaciana para detectar los bordes de una imagen y también la derivada parcial de la imagen para ver qué método es más efectivo.



Figure 11: Función laplaciana con kernel = 3



Figure 12: Función sobels con kernel = 3

Para un tamaño de máscara de 3 píxeles podemos ver que la función Laplaciana con dos tipos distintos de bordes nos ofrece un resultado similar. El escoger un determinado tipo de bordes nos ayuda a conseguir un resultado más fiable en una imagen en la cual nos encontramos con detalles en los bordes y que podemos perder en caso de rellenar el borde con un color sólido, pero en este caso los resultados han sido idénticos.

Respecto a los bordes de los objetos encontrados, podemos apreciar que la función laplaciana nos da unos bordes más finos y con más detalle, mientras que la suma de las derivadas parciales reconoce bien los bordes pero nos da un resultado más tosco.

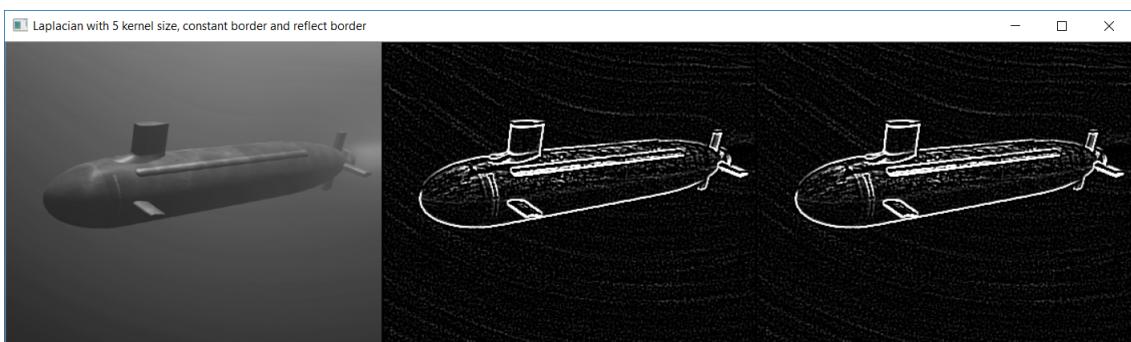


Figure 13: Función laplaciana con kernel = 5

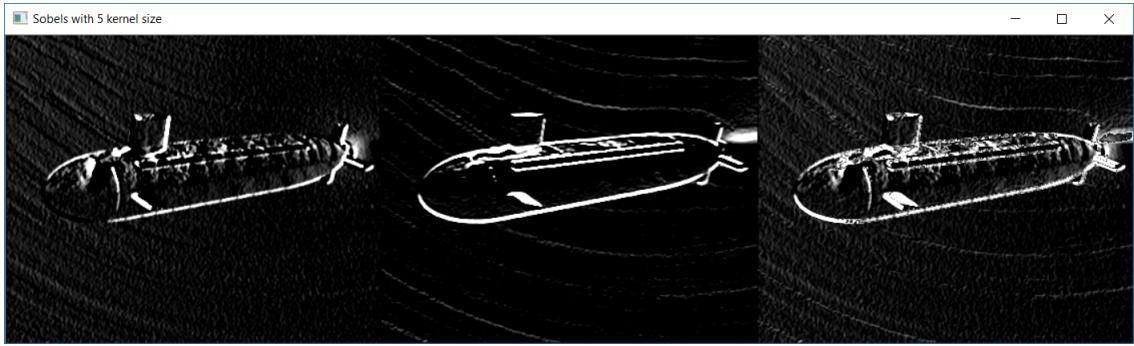


Figure 14: Función sobels con kernel = 5

Al aumentar el tamaño del kernel podemos ver que la función laplaciana ha hecho más gordos los bordes que ya había encontrado con un tamaño de kernel menor y que muestra ciertos detalles que antes no se apreciaban, a pesar de mostrar un poco más de ruido.

Usando las derivadas parciales generamos mucho ruido en el objeto, debido a las luces y sombras, y en la propia escena. Al sumar las derivadas obtenemos una imagen con mucho ruido y rugosidad. Por lo que podemos determinar que la función Laplaciana nos ofrece mejores resultados.

2 Convolución y pirámides

Implementar apoyándose en las funciones `getDerivKernels`, `getGaussianKernel`, `pyrUp()`, `pyrDown()`, escribir funciones los siguientes:

2.1 El cálculo de la convolución 2D con una máscara separable de tamaño variable. Usar bordes reflejados. Mostrar resultados.

Para el desarrollo de este apartado he optado por usar `getGaussianKernel` para obtener una máscara separable de tamaño `k` y `sigma` fijo en 3.

Una vez usamos la función y obtenemos nuestro kernel calculamos la convolución como la multiplicación de la máscara por su traspuesta y la aplicamos usando la función `filter2D` con bordes constantes (bordes negros)

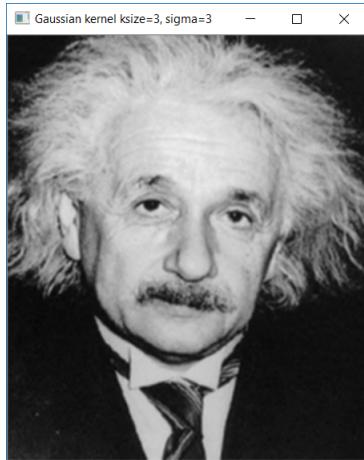


Figure 15: Kernel = 3

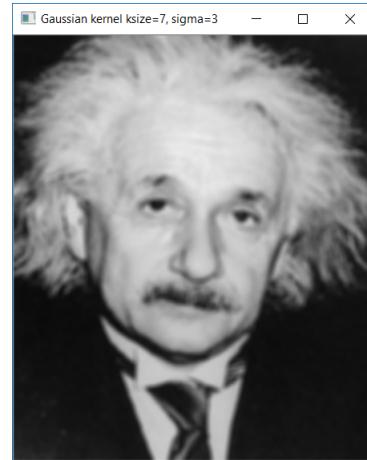


Figure 16: Kernel = 7

Ejercicio 2 - Apartado a

```
Reading images/einstein.bmp in grey mode
Convolution 2D: [[0.1069973 0.11310982 0.1069973 ]
 [0.11310982 0.11957153 0.11310982]
 [0.1069973 0.11310982 0.1069973 ]]
Press ESC to exit or S to save it

Convolution 2D: [[0.01129725 0.01491455 0.01761946 0.01862602
0.01761946 0.01491455
0.01129725]
[0.01491455 0.01969008 0.02326108 0.02458993 0.02326108 0.01969008
0.01491455]
[0.01761946 0.02326108 0.02747972 0.02904957 0.02747972 0.02326108
0.01761946]
[0.01862602 0.02458993 0.02904957 0.03070911 0.02904957 0.02458993
0.01862602]
[0.01761946 0.02326108 0.02747972 0.02904957 0.02747972 0.02326108
0.01761946]
[0.01491455 0.01969008 0.02326108 0.02458993 0.02326108 0.01969008
0.01491455]
[0.01129725 0.01491455 0.01761946 0.01862602 0.01761946 0.01491455
0.01129725]]
Press ESC to exit or S to save it
```

Figure 17: Convolución para k=3 y k=7

Como se puede comprobar en esta última imagen obtenemos dos matrices gaussianas en las que podemos apreciar que en la primera, con un tamaño del kernel de 3 pixeles, apenas hay diferencia entre los valores de la matriz, pero en la segunda, con un tamaño de 7 pixeles, se nota más la diferencia con los puntos que están más alejados de la media.

2.2 El cálculo de la convolución 2D con una máscara 2D de 1^a derivada de tamaño variable. Mostrar ejemplos de funcionamiento usando bordes a cero.

En este apartado usamos *getDerivKernels* para obtener una máscara de primera derivada con un tamaño variable de k. El uso de la primera derivada nos permite calcular los saltos que hay en las fronteras y realzarlos.

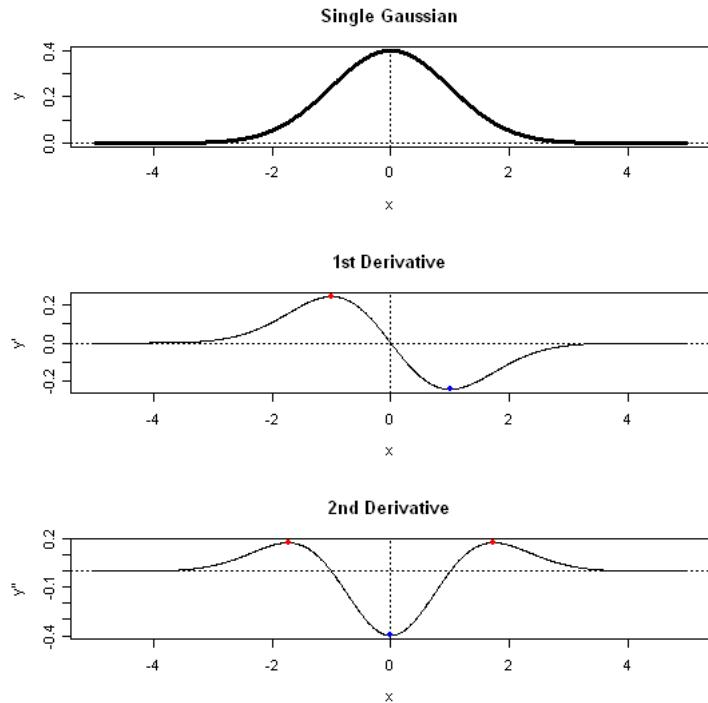


Figure 18: Derivadas

La primera derivada realza las partes de una imagen donde encontramos un contraste mayor, y esto suele darse en las fronteras donde la información los pixeles suele ser muy diferente.



Figure 19: Primera derivada k=3

Conforme aumentamos el tamaño de la máscara usamos más valores para operar con la derivada y en este caso hacemos que los saltos sean más grandes.



Figure 20: Primera derivada k=9

2.3 El cálculo de la convolución 2D con una máscara 2D de 2^a derivada de tamaño variable.

En este apartado volvemos a usar *getDerivKernels* pero esta vez usando la segunda derivada para crear una máscara con tamaño k.

En el apartado anterior hemos podido ver la forma que tiene la segunda derivada, dicha máscara realza un pixel central sobre los contiguos al mismo. De esta forma también conseguimos realzar las fronteras en una imagen.

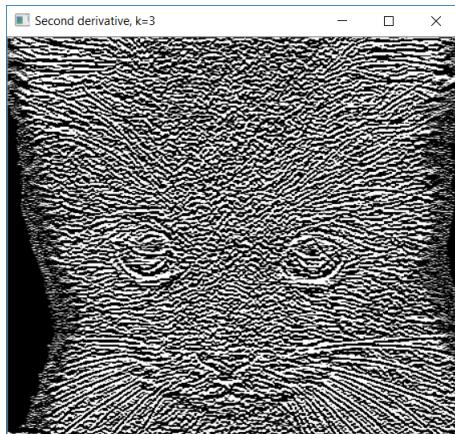


Figure 21: Kernel = 3



Figure 22: Kernel = 15

Al igual que en el caso anterior conforme aumentamos el tamaño del kernel los saltos entre las fronteras son más pronunciados por lo que a pesar de representar aquellas fronteras más relevantes perdemos detalle en las que son menos apreciables.

2.4 Una función que genere una representación en pirámide Gaussiana de 4 niveles de una imagen. Mostrar ejemplos de funcionamiento usando bordes.

Con una pirámide Gaussiana representamos cómo se vería una imagen al hacerla más pequeña y después volver a escalarla al tamaño anterior. Haciendo esto podemos ver que en cada nivel la imagen tiene menos calidad debido que al hacer más pequeña la imagen estamos perdiendo información ya que no vamos a poder ver todos los píxeles que teníamos antes, y al volver a escalarla interpolamos menos información a un tamaño mayor, por lo que podemos ver gráficamente cómo va empeorando con el paso de los niveles.

Se ha realizado con dos tipos de bordes, el borde por defecto que tiene OpenCV y el borde reflejado que replica la información de las fronteras de la imagen. El resultado es similar para ambos bordes.

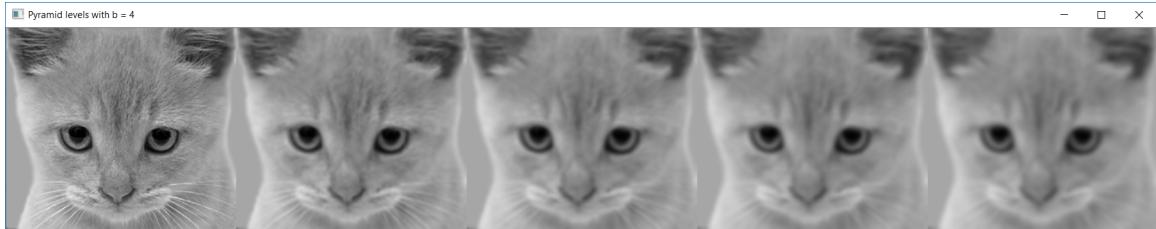


Figure 23: Pirámide Gaussiana de 4 niveles con bordes por defecto



Figure 24: Pirámide Gaussiana de 4 niveles con bordes reflejados

2.5 Una función que genere una representación en pirámide Laplaciana de 4 niveles de una imagen. Mostrar ejemplos de funcionamiento usando bordes.

En este caso hacemos una pirámide similar a la del ejercicio anterior en la que lo primero que debemos hacer es escalar la imagen original a una más pequeña y volverla a escalar al tamaño original. Una vez hecho esto tenemos la misma imagen que en el apartado anterior, una imagen con menos calidad y más difuminada. Con dicha imagen la restamos a la imagen original y de esta forma conseguimos eliminar las bajas frecuencias y hallar los bordes. Mientras iteramos sobre las imágenes modificadas vamos eliminando cada vez más información y al final encontramos una imagen con las fronteras más relevantes.

Para esta representación también se han usado dos bordes, el que nos ofrece por defecto OpenCV y el borde reflejado. Al igual que en el apartado anterior la diferencia apenas es apreciable.



Figure 25: Pirámide Laplaciana de 4 niveles con bordes por defecto



Figure 26: Pirámide Laplaciana de 4 niveles con bordes reflejados

3 Imágenes Híbridas

Mezclando adecuadamente una parte de las frecuencias altas de una imagen con una parte de las frecuencias bajas de otra imagen, obtenemos una imagen híbrida que admite distintas interpretaciones a distintas distancias (ver hybrid images project page).

Para seleccionar la parte de frecuencias altas y bajas que nos quedamos de cada una de las imágenes usaremos el parámetro sigma del núcleo/máscara de alisamiento gaussiano que usaremos. A mayor valor de sigma mayor eliminación de altas frecuencias en la imagen convolucionada. Para una buena implementación elegir dicho valor de forma separada para cada una de las dos imágenes (ver las recomendaciones dadas en el paper de Oliva et al.).

Recordar que las máscaras 1D siempre deben tener de longitud un número impar. Implementar una función que genere las imágenes de baja y alta frecuencia a partir de las parejas de imágenes (solo en la versión de imágenes de gris) . El valor de sigma más adecuado para cada pareja habrá que encontrarlo por experimentación.

3.1 Escribir una función que muestre las tres imágenes (alta, baja e híbrida) en una misma ventana. (Recordar que las imágenes después de una convolución contienen número flotantes que pueden ser positivos y negativos)

Para este apartado he optado por usar un filtro gaussiano con un tamaño de kernel variable que elegiremos en función del tamaño de las imágenes que queramos combinar para que el resultado se ajuste mejor. Con la misma máscara de alisamiento le aplicamos a una imagen el filtro para obtener las bajas frecuencias y con la otra imagen hacemos lo mismo para obtener las bajas frecuencias y restarselas a la original para obtener las altas. De esta forma ya tenemos nuestras dos imágenes con bajas y altas frecuencias, haciendo uso de la función *add* de OpenCV las combinamos para obtener una imagen final que esté contenida en el rango (0, 1).

3.2 Realizar la composición con al menos 3 de las parejas de imágenes

Aquí podemos ver las distintas imágenes que se han obtenido usando la función del apartado anterior.

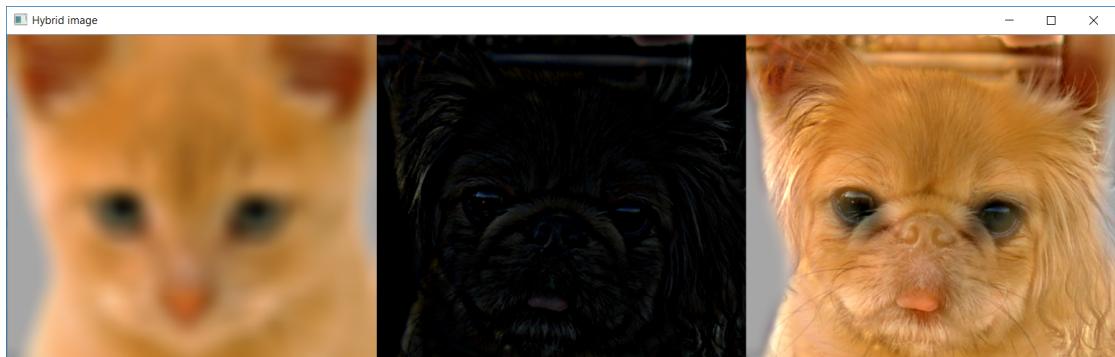


Figure 27: Gato y Perro

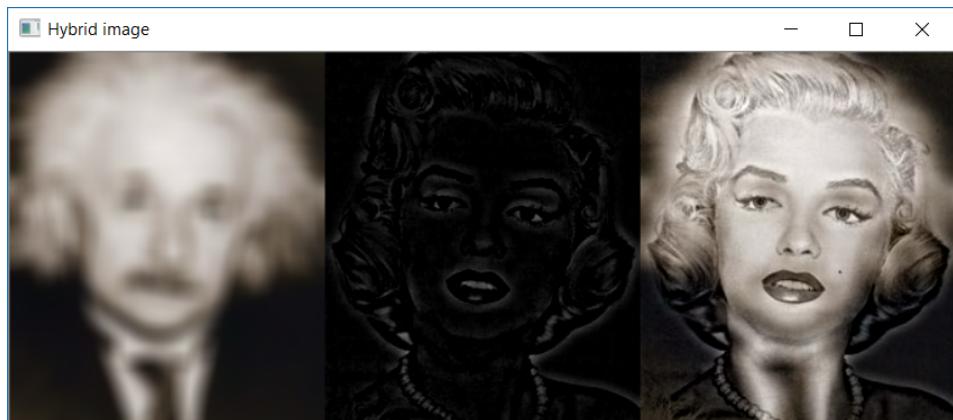


Figure 28: Einstein y Marilyn

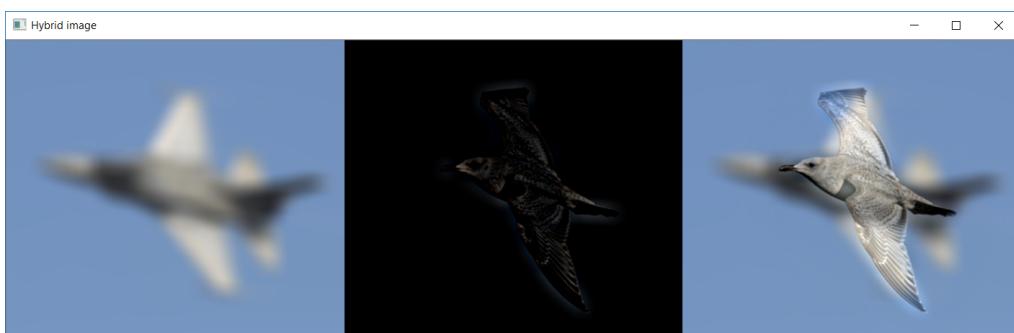


Figure 29: Avión y pájaro



Figure 30: Pescado y submarino

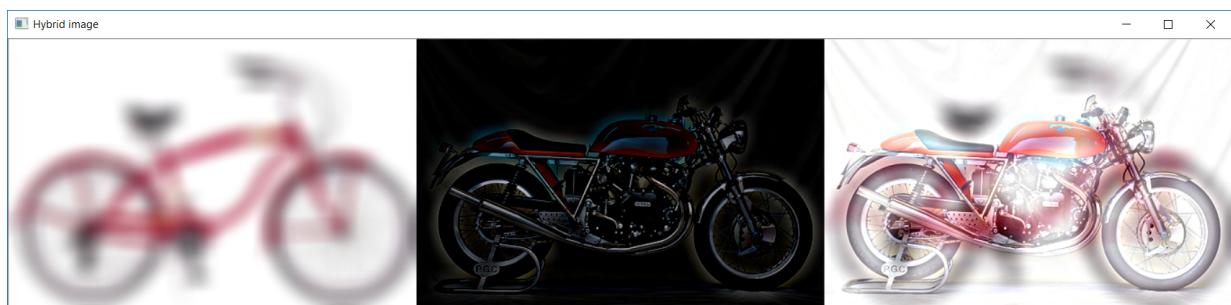


Figure 31: Bicicleta y Motocicleta