

## Ejercicio 1 (Filtros promedio)

**a)** Al hablar de la captura de las imágenes en el sensor vimos que la mayoría de las cámaras tienen un filtro anti-aliasing que suaviza un poco la imagen para reducir los efectos de Moire debidos a un muestreo inadecuado. Si partimos de una imagen digital y la submuestreamos tenemos un problema similar. Al usar menos muestras, detalles que son correctamente capturados en la foto original son demasiado "finos" para la nueva malla de muestreo, creando efectos de Moire o "aliasing".

Cargar la imagen "brick.jpg" y mostrarla con `imshow()`. Por defecto `imshow` muestra la imagen al tamaño original (o tan grande como permita el tamaño de la pantalla).

Submuestrear la imagen en un factor 4 en cada dimensión (quedarnos con 1 fila de cada 4 y una columna de cada 4) y visualizar el resultado. [¿Se aprecia el "aliasing"?](#) Podéis ver el mismo efecto si reducís el tamaño de la ventana conteniendo la imagen original. En este caso es la tarjeta gráfica la que realiza el submuestreo para que la imagen "quepa" en la ventana reducida.

Cread un filtro gaussiano `S` con un tamaño de  $7 \times 7$  y un ancho de  $\sigma = 2$  usando `S=fspecial('gaussian',7,2)`. Comprobad que, al ser un filtro promedio, la suma de los coeficientes de `S` es la unidad. Filtrar la imagen original con el filtro `S` usando `imfilter` con la opción 'replicate' y submuestrear la imagen resultante con el mismo factor 4 de antes. [Adjuntar ambas imágenes submuestreadas \(con/sin filtro previo\)](#). [¿Se han reducido los efectos de Moire?](#)

Otra alternativa para reducir una imagen un factor  $F$  es usar `im2=imresize(im,1/F)`. Aquí no es necesario filtrar previamente la imagen (`imresize()` se ocupa de ello).

**b)** Crear una imagen color de tamaño (400x600) rellena de colores aleatorios con `org=rand(400,600,3)`. Visualizarla con `imshow`.

Copiar `org` en `im1`. Extraer el canal rojo, filtrarlo (`imfilter`) con un filtro gaussiano `S` del apartado anterior y volver a colocar el resultado en el canal rojo de `im1`.

Copiar `org` en `im2` y repetir el proceso pero ahora modificando el canal verde de la misma forma. Finalmente, copiar `org` en `im3` y haced lo mismo con el canal azul. [Adjuntad el código usado para crear im1,im2,im3.](#)

Visualizar con `imshow` las tres imágenes modificadas (`im1`, `im2`, `im3`) y compararlas con la original. [Adjuntad las imágenes im1 e im2.](#)

Al haber usado un filtro promedio es normal que las imágenes filtradas se vean algo menos nítidas. [¿Se aprecian las tres imágenes igual de desenfocadas?](#)  
[¿Cuál veis más desenfocada? ¿Se os ocurre alguna explicación?](#)

**Ejercicio 2 (realce de bordes):** Muchas veces se desea el efecto OPUESTO al de un filtro promedio: realzar el detalle en lugar de eliminarlo. Para ello se usan los filtros de realce de bordes ("sharpening" o nitidez). Recordad que al aplicar un filtro promedio a una imagen  $I_s = \text{imfilter}(im, S)$ , el resultado es una imagen a la que le falta parte de su detalle. Si ahora hacemos:

$$im - I_s = (im) - (im - \text{detalle}) = \text{detalle},$$

y podemos separar la imagen en dos partes:  $im = I_s + \text{detalle}$

Esto es similar al LAB1 donde separamos la parte en B/W y la parte de color de una foto, que luego podíamos amplificar si queríamos aumentar la saturación del color. Lo mismo podemos hacer ahora con el detalle haciendo  $im2 = (im + \alpha \cdot \text{detalle})$  para añadir una ración extra de detalle (con  $\alpha > 0$ ).

Aplicar un "sharpening" a la imagen del fichero "img1.jpg", siguiendo estos pasos:

- Leer imagen y convertirla a double (manteniendo valores entre 0 y 255).
- Aplicar un filtro gaussiano S (soporte 11x11 y  $\sigma=3$ ) para obtener  $I_s$ .
- Aislar el detalle restando  $I_s$  de la imagen original (pasada a double). [Visualizar el detalle con imshow\(\), adjuntando la imagen resultante.](#)
- Tras aislar el detalle, reforzarlo usando  $\alpha=1.5$  en la fórmula anterior.

Construir una imagen poniendo en su mitad izquierda (columnas 1-512) la imagen original y en su mitad derecha (columnas 513-1024) la imagen realzada. Llenad con 0's la columna 512 para separar ambas zonas. [Adjuntad la imagen "compuesta".](#)

Un problema de este realce de bordes es que no diferencia entre el genuino detalle que queremos resaltar (p.e. la textura de la roca) y el ruido presente (que también será realzado). Haced zoom en alguna zona del cielo en ambas imágenes donde se aprecien estas diferencias del nivel de ruido. [Adjuntad los zoom en ambas zonas.](#)

Para reducir este efecto de la amplificación del ruido se puede fijar un umbral U, de forma que para aquellos píxeles con un valor absoluto del detalle menor que U se considera que es ruido y no realza. Si se supera el umbral U es que estamos tratando de verdadero detalle y se refuerza con la fórmula anterior. Este umbral U es el tercer parámetro del filtro unsharp de Photoshop, junto con el radio del filtro gaussiano aplicado (nuestro valor de  $\sigma$ ) y el % de detalle extra añadido (nuestro  $\alpha$ ). [Repetir el filtrado usando un umbral U=5. Hacer zoom en una zona del cielo y ver como se ha reducido la magnificación del ruido.](#)

Otro problema son los halos que aparecen alrededor de los bordes más destacados. Hacer zoom sobre una zona de la imagen realzada donde sean especialmente obvios estos halos y [adjuntarla](#). El origen del problema es que el filtro gaussiano modifica drásticamente los bordes, lo que causa (al restar ambas imágenes) la detección de un falso detalle en esas zonas, que también es amplificado por el algoritmo. La solución en este caso es usar para suavizar la imagen un filtro tipo "bilateral" que promedie la imagen pero respetando los bordes.



# Proyecto: Registro de imágenes, representaciones en pirámides y deconvolución.

**1) Registro de imágenes (15%):** Imaginad disparar una ráfaga de fotos apuntando al mismo sujeto (estático). Obtendréis una serie de imágenes similares, desplazadas algunos píxeles en ambas direcciones ( $dX$ ,  $dY$ ), debido a los inevitables movimientos de la mano. En muchas aplicaciones es necesario determinar este desplazamiento como un paso previo antes de combinar las imágenes (por ejemplo para crear una fotografía HDR). A esta aplicación se le llama **registro** de imágenes.

Una forma de obtener este desplazamiento ( $dX, dY$ ) es con una correlación entre las imágenes (o un trozo de ellas para ser más rápido) usando `imfilter()`. Cuando al mover una imagen sobre la otra acertamos con una zona que es similar en ambas, la correlación dará valores altos. El máximo de dicha correlación nos indica cuando las imágenes coinciden lo más posible y de su posición hallamos el desplazamiento entre ambas imágenes.

En estas aplicaciones los resultados son mucho mejores si al hacer la correlación usamos la información del detalle en vez de la imagen original. La idea es que es más sencillo determinar cuando dos imágenes están "alineadas" si las imágenes usadas son imágenes "esquemáticas" de líneas, bordes, etc.

Cargar las imágenes 'gato0.jpg' y 'gato1.jpg' en `im0` e `im1` y convertirlas a `double` con `im2double()`. Son dos tomas (B/W) del mismo sujeto con un desplazamiento entre ellas de unos pocos píxeles ( $dx=-13$ ,  $dy=9.00$ ).

- Extraer su "detalle" (imagen - imagen suavizada) usando un filtro gaussiano de soporte  $9 \times 9$  y de ancho  $\sigma=2$ . Guardar los resultados en `det0`, `det1`.
- Extraer un trozo del detalle de la 1ª imagen (`det0`) de  $\pm 25$  píxeles alrededor de  $OY=90$ ,  $OX=300$ : **`OY=90; OX=300; R=25; T0=det0(OY+(-R:R),OX+(-R:R));`** Repetir para la 2ª imagen (guardando el trozo en `T1`). Lo de usar un trozo es para no hacer la correlación con toda la imagen y así hacer más rápido el proceso. La posición ( $OX, OY$ ) de donde sacamos los trozos no es crítica, salvo que sea una zona que carezca de detalles.
- Calculamos la correlación entre `T0` y `T1`: **`C=imfilter(T0,T1);`**
- Usando **`surf(C)`** pintar la correlación `C` obtenida como una superficie 2D. Adjuntad la imagen. Puede que tengáis que girar la imagen con  para ver mejor el pico de correlación. Usar el cursor de datos  para determinar la posición  $X, Y$  del pico. ¿Qué valores obtenéis?

Los valores anteriores no son todavía los desplazamientos buscados. Para hallar ( $dX, dY$ ) considerad que si las imágenes no hubieran estado desplazadas, ambos

trozos serían el mismo y el máximo caería exactamente en el centro de la correlación. Como los trozos tiene un lado de  $(2 \cdot R + 1)$  píxeles, su centro está en la posición  $(R+1, R+1)$ . Restad  $(R+1)$  a la posición del pico de C para obtener el desplazamiento entre imágenes. [¿Valores de los desplazamiento dX, dY?](#)

En realidad, no hace falta visualizar C y seleccionar el máximo manualmente. Se puede automatizar haciendo:

```
[M pos]=max(C(:)); [i,j]=ind2sub(size(C),pos);
```

La función max devuelve el valor máximo (M) de C y su posición (pos) dentro de C. La pega es que pos es un índice lineal (que recorre la matriz C por columnas), pero conocido el tamaño de C, el segundo comando nos da la fila i y columna j a las que corresponde el índice pos. En (i, j) debéis obtener los valores obtenidos antes al pinchar en el máximo. Restando  $(R+1)$  a (i, j) obtenemos el desplazamiento (dY,dX) entre ambas imágenes. [Adjuntar vuestro código.](#)

Repetir ahora con trozos más grandes ( $R=50$ ). [Adjuntad la imagen de la correlación obtenida.](#) [¿Obtenéis el resultado correcto?](#) [¿Cuáles creéis que son las ventajas y desventajas de usar un R mayor?](#)

Volver a correr el código pero ahora usando sub-imágenes más pequeñas ( $R=10$ ). [Adjuntad la imagen de la correlación obtenida.](#) [¿Se obtiene el resultado correcto?](#) [¿Cuál es el problema?](#)

## 2) Implementación de la pirámide Laplaciana (20%)

Se trata de escribir una función implementando la pirámide laplaciana vista en clase. Usaremos este template:

```
function p=lap(im,N)
    if nargin==1, N=5; end
    im=im2double(im);
    p=cell(1,N);
    ...
return
```

La función recibe la imagen im y el número N de "pisos" en la pirámide. Si no se especifica nada se usan  $N=5$  niveles. Recordad que  $N=5$  significa que salen 4 niveles de detalle + 1 versión (muy) reducida de la imagen original.

Los resultados de los diferentes niveles se guardarán en el argumento de salida p, (un array de celdas). Un array de celdas es como un vector o tabla normal pero que puede contener diferentes tipos de datos en sus casillas (en este caso matrices de diferentes tamaños). La diferencia es que usamos llaves en lugar de paréntesis para acceder a sus elementos:  $p\{1\}$ ,  $p\{2\}$ , en lugar de  $p(1)$ ,  $p(2)$ . Si, por ejemplo,  $p\{2\}$  es una matriz podemos direccionar sus componentes haciendo  $p\{2\}(3,4)$ .

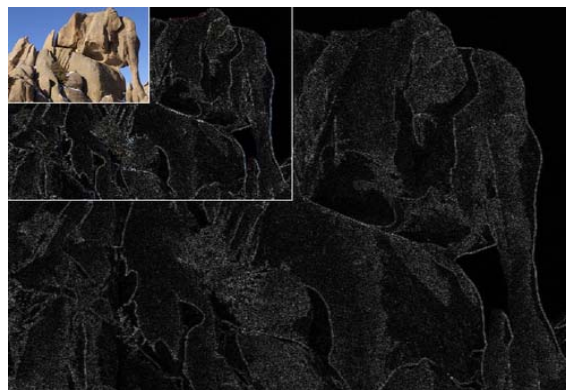
El proceso es sencillo. Haced un bucle desde  $k=1$  a  $N-1$  para calcular los  $N-1$  niveles de detalle y en cada paso dentro del bucle:

1. Crear una versión reducida de la imagen  $im$  en un factor 2, usando `imresize`.
2. Volver a aplicar `imresize()` a la imagen reducida pero ahora para ampliarla en un factor 2. La imagen resultante,  $im2$ , es del mismo tamaño que la imagen  $im$  de partida, pero habrá perdido el detalle, ya que `imresize()`, al reducir la imagen elimina parte de su detalle para evitar efectos de aliasing (ejercicio 1) y al ampliarla y recuperar el tamaño original ese detalle ya no está presente.
3. Restar a la imagen original ( $im$ ) la imagen sin detalle ( $im2$ ). De esta forma se obtiene el detalle correspondiente a ese nivel, que se guardará en  $p\{k\}$ .
4. Guardar en  $im$  la versión reducida para que en el siguiente nivel repitamos el proceso partiendo de una versión de la imagen de tamaño mitad.

Tras terminar el bucle, guardar la última versión de la imagen reducida en el último nivel de la pirámide,  $p\{N\}$ . [Adjuntar código de vuestra función lap.m](#)

Probad la función con "img1.jpg" usando  $N=3$ . Verificar que los 3 niveles de la pirámide tienen tamaños: 768x1024, 384x512, 192x25

Con la función `ver_lap(p)` podéis visualizar el contenido de los niveles de la pirámide laplaciana (ver figura adjunta).



[Repetir para  \$N=6\$  niveles.](#)  
[Adjuntad la imagen resultado.](#)

Escribiremos ahora la función inversa a `lap.m` que recibe una pirámide y la colapsa, recuperando la imagen original (o algo muy parecido). El template será:

```
function im=invlap(p)
N = length(p); % Número de niveles de la pirámide
...
end
```

Tras averiguar cuántos niveles tiene la pirámide, inicializaremos la imagen  $im$  a recuperar con el último nivel de la pirámide  $p\{N\}$  (la versión miniatura de la imagen original). Luego haced un bucle desde  $k=N-1$  a 1, barriendo el resto de los niveles. En cada paso:

1. Ampliamos  $im$  por un factor 2 usando `imresize()`.
2. Sumamos a  $im$  el detalle correspondiente a ese nivel,  $p\{k\}$ .

Al terminar tenemos en  $im$  una imagen del tamaño original. Multiplicarla por 255 y convertirla a `uint8` antes de salir. [Adjuntar código de vuestra función invlap.m](#)

Aplicar `inv_lap()` a la pirámide antes calculada y visualizar la imagen recuperada. Para ver si hay diferencias hallar su resta  $\text{dif} = \text{double}(\text{im}) - \text{double}(\text{rec})$  y calcular su máximo/mínimo haciendo  $\text{max}(\text{dif}(:))$ ,  $\text{min}(\text{dif}(:))$ . Debe salir 0 en ambos casos, indicando que son idénticas. Repetir el cálculo pero eliminando ahora la conversión final a `uint8` dentro de `inv_lap` para poder apreciar los posibles errores de redondeo. ¿De qué orden son ahora el máximo y mínimo obtenidos?

Los siguientes apartados 3) y 4) son independientes.

### 3) APLICACIÓN: alineación de imágenes (35%)

En este apartado vamos a aplicar la pirámide laplaciana (apartado 2) para la alineación de imágenes, usando la técnica de correlación que hemos visto en el apartado 1.

Las imágenes a alinear pertenecen a la colección de fotografías tomadas por Sergey Prokudin-Gorskii durante sus viajes por el imperio ruso a principios del siglo XX. De cada imagen el fotógrafo tomaba 3 exposiciones usando un filtro rojo, uno verde y otro azul. Posteriormente, usando 3 proyectores con luz de diferente color y alineando cuidadosamente las imágenes sobre la pantalla de proyección los asistentes podían ser testigos de algunas de las primeras fotografías en color (y las más antiguas conservadas).

Cargar la imagen contenida en el fichero `multiplex.jpg`. Visualizarla con `imshow`. La imagen (derecha) tiene un tamaño de 4800 (alto) x 1792 (ancho) y consta de las tres imágenes tomadas con los diferentes filtros montadas una encima de la otra (cada una con un tamaño de 1600 x 1792). La imagen superior corresponde al filtro azul, la central al filtro verde y la de abajo al filtro rojo.



1) Dividir la imagen en tres subimágenes R, G y B, cada una de tamaño 1600x1792.

2) Para eliminar la zona de los bordes que tienen un borde negro y están más deteriorados, extraeremos la parte central de la imagen, quedándonos con una imagen de un tamaño de 1401 x 1601 píxeles (alto x ancho):


- A partir del centro de las imágenes  $CX=896$ ,  $CY=800$  definimos el rango de píxeles a extraer:  $rx = CX + (-800:800)$ ;  $ry = CY + (-700:700)$ ;
- Reservamos espacio para una imagen en color de tamaño 1401x1601x3, donde guardaremos la imagen en color resultante.
- Extraer el rango de píxeles  $(ry, rx)$  de la imagen G y colocarlos en el 2ª canal de la imagen en color. Repetir para los canales R (1er canal) y B (3er canal).

Visualizad el resultado. [Haced zoom sobre alguna de las caras y adjuntadlo.](#)



### 3a) Alineación manual:

Nuestro objetivo es hacer una alineación automática pero, para saber lo que tiene que salir, lo haremos primero de forma manual. Escogeremos uno de los canales (el verde G) como referencia y determinaremos de forma manual el desplazamiento ( $dX, dY$ ) entre los otros dos canales rojo (R) y azul (B) respecto al verde (G).

Para ello visualizar el canal verde G en una figura y el rojo R en otra. Haciendo zoom en alguna zona distintiva marcar algún punto que podáis identificar en ambas imágenes usando el cursor de datos de MATLAB (). Anotad la posición del punto en ambas imágenes: la diferencia entre su posición ( $X_r, Y_r$ ) en el canal R y su posición ( $X_g, Y_g$ ) en el canal G será el desplazamiento  $dX, dY$  entre ambos canales.

Una vez conocido el desplazamiento ( $dX, dY$ ) del canal rojo respecto al verde vamos a usar esa información al extraer los datos de R para rellenar el 1er canal de la imagen en color. Basta restar  $dY, dX$  al rango  $ry, rx$  de píxeles a extraer, cogiendo los datos de  $dY$  filas más arriba y  $dX$  columnas más a la izquierda.

Repetir para el canal B (de nuevo usando el canal verde G como referencia). Usando ambos desplazamientos tendremos una imagen color con sus canales correctamente alineados. El canal verde no hay que modificarlo al ser el usado como referencia.

Mostrar un zoom de los tres canales marcando los puntos usados como referencia.  
 Dar los desplazamientos ( $dX, dY$ ) hallados para el canal rojo y el azul.  
 Adjuntar la imagen alineada, junto con un zoom en la zona de una de las caras.

### 3b) Alineación automática usando pirámides:

Se trata ahora de automatizar el proceso anterior de alineación de los canales. Para ello escribiremos una función:

```
function [dX,dY]=registra(im0,im1,X0,Y0)
% Receives two images im0 im1.
% Computes the correlation around position X0,Y0
% Return displacement (dX,dY) found between im1 and im0
```

La función recibe dos imágenes ( $im0, im1$ ) y una posición ( $X0, Y0$ ) alrededor de la cual haremos la correlación para detectar el desplazamiento ( $dX, dY$ ) entre ambas.

Para estimar el desplazamiento de forma eficaz usaremos un **enfoque piramidal**, empezando la búsqueda en el nivel más bajo de detalle  $p\{N-1\}$  de la pirámide. En ese nivel las imágenes son pequeñas, pero corresponden a zonas muy amplias de la imagen original. Por ejemplo, en una pirámide con 5 niveles, una búsqueda usando un trozo de  $\pm 15$  píxeles en el nivel  $p\{5\}$  equivale a  $\pm 30$  píxeles en  $p\{4\}$ , etc. hasta llegar a  $\pm 240$  píxeles en  $p\{1\}$ . De esta forma, una pequeña búsqueda en el nivel más bajo puede detectar un desfase muy grande en el original. Así podemos

usar correlaciones con trozos pequeños, que tienen un coste computacional bajo, sin el problema de no encontrar el desplazamiento si éste era mayor que la zona a explorar.

Además, la búsqueda en una pirámide se puede refinar iterativamente. Suponed que el mejor ajuste en el nivel más bajo se ha encontrado para un desplazamiento de  $(+2, -3)$  píxeles. En el siguiente nivel refinaremos esa estimación partiendo de un desplazamiento inicial de  $(+4, -6) = 2 \times (2, -3)$ . De esta forma siempre tenemos una buena hipótesis inicial del desplazamiento en cada nivel y basta buscar en su vecindad para mejorar la estimación. En cada paso duplicamos los valores obtenidos del nivel anterior para obtener la hipótesis de partida, hasta llegar al nivel superior de detalle (que corresponde a trabajar sobre la imagen completa).

Los pasos a realizar dentro de la función son:

- 1) Calcular las correspondientes pirámides  $p_0$  y  $p_1$  de las dos imágenes  $im_0$  e  $im_1$ . Usar  $L=6$  niveles, lo que corresponde a tener 5 niveles de detalle:  $p\{1\}, \dots, p\{5\}$ .
- 2) Definir  $R=15$ , el "radio" de los trozos usados en la correlación. A pesar de ser un  $R$  pequeño el algoritmo funcionará debido a su carácter piramidal.
- 3) Dividir las posiciones  $X_0, Y_0$  por  $2^{(L-1)}$  para determinar la posición de partida en el nivel más bajo de la pirámide. Redondear usando  $\text{round}()$ .
- 4) Inicializar los valores de  $X_1$  e  $Y_1$ . La posición  $(X_1, Y_1)$  es la posición en  $im_1$  que más se parece a  $(X_0, Y_0)$  en  $im_0$ . Inicialmente no conocemos el desplazamiento, por lo que suponemos que es nulo y hacemos  $X_1=X_0, Y_1=Y_0$ .

A continuación, entramos en un bucle barriendo los niveles de detalle desde  $k=L-1$  a 1. En cada paso del bucle:

- Duplicar las coordenadas  $(X_0, Y_0)$  y  $(X_1, Y_1)$  para determinar las posiciones en el nivel actual.
- Extraer dos "trozos"  $T_0$  y  $T_1$  del detalle a nivel  $\{k\}$  de ambas pirámides. Para la primera imagen  $T_0$  será el trozo de  $p_0\{k\}$  centrado en  $(X_0, Y_0)$ . Para la segunda,  $T_1$  será un trozo de  $p_1\{k\}$  centrado en las coordenadas  $(X_1, Y_1)$ . En ambos casos el rango de píxeles a extraer es de  $(-R:R)$  en ambos ejes.
- Calcular la correlación  $C$  entre  $T_0$  y  $T_1$  usando  $\text{imfilter}$  y hallar la posición  $(i, j)$  del máximo de la correlación  $C$  y su desplazamiento  $(dy, dx)$  respecto al centro ideal  $(R+1)$ , como hicimos en el ejercicio anterior. Recordad que las columnas  $(j)$  corresponden a coordenadas  $X$  y las filas  $(i)$  a la  $Y$ .
- Calculados  $(dy, dx)$ , restarlos a  $(Y_1, X_1)$  para actualizar sus valores.

El algoritmo arranca en el nivel más bajo asumiendo que no hay desplazamiento ( $X_1=X_0, Y_1=Y_0$ ). Si detectamos un desplazamiento para el máximo de la correlación



de  $dy=-1$ ,  $dx=2$  actualizamos  $X1=(X1-dx)=X-2$ ;  $Y1=(Y1-dy)=Y1+1$ . En el siguiente paso duplicamos  $(X0,Y0)$  y  $(X1,Y1)$  para tener en cuenta el tamaño de la imagen en el nuevo nivel y volvemos a comparar la zona centrada en  $(X0,Y0)$  en la 1ª imagen y la centrada en  $(X1,Y1)$  en la 2ª imagen. De nuevo, los desplazamientos detectados se van restando a  $(X1,Y1)$  para mejorar su estimación.

Para depurar, tras calcular la correlación  $C$  visualizarla con `surf(C)` (seguido de un pause que espera a que pulséis una tecla para continuar). Deberíais ver un pico de correlación que, si actualizáis correctamente  $X1,Y1$  en cada paso, debe mantenerse más o menos centrado al subir de nivel. Al acabar el bucle la diferencia entre  $(X1,Y1)$  y  $(X0,Y0)$  nos da el desplazamiento  $(dX,dY)$  de la imagen  $im1$  respecto de  $im0$  (que será la referencia). [Adjuntad código de registra.m](#)

Una vez completada la función llamarla usando las componentes G y R como  $im0$  e  $im1$  respectivamente. Para la posición a comparar  $(X0,Y0)$  podéis usar el centro de la imagen  $(CX, CY)$ . Repetir usando B para  $im1$  (de nuevo, usando G como  $im0$ ).

[Volcad las posiciones  \$\(X0,Y0\)\$  y  \$\(X1,Y1\)\$  que vais obteniendo para cada nivel. ¿Desplazamientos finales  \$\(dX,dY\)\$  obtenidos en ambos casos?](#) Deben ser similares a los obtenidos antes manualmente. Tras obtener  $(dX,dY)$  usarlos como se hizo en la parte manual para alinear los canales. Combinar todo el código en un script que:

- Lea la imagen original y la divida en 3 imágenes R,G,B (1600 x 1792)
- Alinee las tres imágenes R,G,B usando `registra()`, y use la info de  $dX, dY$  para montar la imagen color (tamaño 1401x1601) desplazando adecuadamente los canales R y B.
- Visualice la imagen color con `imshow()` verificando que está alineada.

[Adjuntar código del script y la imagen color obtenida. Extraer un zoom de una de las caras y compararlo con el resultado manual. Volver a ejecutar el script con `multiplex2.jpg`. Adjuntad los desplazamientos encontrados y la imagen resultante.](#)

#### **4) Deconvolución (30%)**

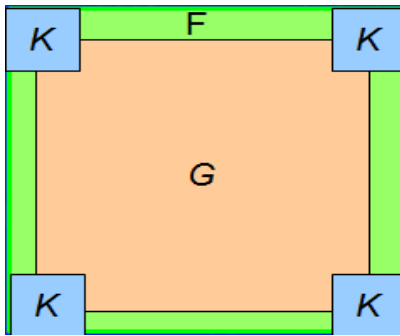
Si la degradación de una imagen es grande, para poder corregirla debemos conocer el proceso que causó el problema. En una primera aproximación, muchos de los potenciales problemas en la captura de una imagen (desenfoque, movimiento de la cámara, etc.) pueden modelarse como la aplicación de un filtro o máscara. Por ello es muy importante saber invertir una operación de filtrado. Se conoce como deconvolución el problema en el que conocemos una imagen degradada  $G$ , obtenida a partir de una imagen original  $F$  (que queremos hallar) a través de una operación de filtrado con una máscara o filtro  $K$  (también conocido):

$$G = F * K ,$$

donde  $*$  denota la operación de convolución o filtrado.

El nombre deconvolución indica que se trata de "deshacer" o invertir la operación de convolución para recuperar la imagen original  $F$ .

Este problema es complicado porque una máscara  $K$ , al aplicarse a una imagen  $F$  puede eliminar parte de su información, que ya no estará presente en  $G$ . Al ser  $G$  nuestro punto de partida, esa información será muy difícil de recuperar.



Además, el tamaño de  $G$  (datos/información de la que disponemos) es menor que el de  $F$ , (la información que queremos recuperar) porque los valores de  $G$  en sus puntos exteriores necesitan valores de  $F$  fuera del tamaño de  $G$  (debido al tamaño de  $K$ ). Podemos ver en el gráfico que si  $G$  es una imagen  $N \times M$  y la máscara  $K$  (cuadrada) es de  $(2n+1) \times (2n+1)$  la imagen  $F$  de partida a recuperar debe tener un tamaño ampliado de  $(N+2 \cdot n) \times (M+2 \cdot n)$ .

No hay un algoritmo directo que invierta la operación de convolución, pero lo que siempre podemos hacer, conocida una posible solución  $F'$ , es filtrarla con la máscara  $K$  y ver si el resultado  $G'$  se parece a la  $G$  que tenemos. Esto da lugar a algoritmos de deconvolución iterativos: prueban con sucesivas  $F'$  y usan  $dG = (G' - G)$  para obtener una  $dF$  con la que modificar la hipótesis, esperando que en sucesivas iteraciones  $dG$  sea cada vez más pequeña (nuestra única referencia de que nos vamos acercando a la solución). Podemos correr un número fijo de iteraciones o terminar las iteraciones si los valores de  $dG$  o  $dF$  son lo suficientemente pequeñas.

El más clásico de estos algoritmos iterativos para el problema de la deconvolución es el de Richardson-Lucy, propuesto en los años 70 para reducir el efecto de las perturbaciones atmosféricas en imágenes astronómicas.

**Datos del problema:** imagen degradada  $G$  tamaño  $N \times M$   
máscara  $K$  de tamaño  $(2n+1) \times (2n+1)$

**Objetivo:** encontrar una imagen  $F$ , de tamaño  $(N+2n) \times (M+2n)$ , ligeramente mayor que el de  $G$ , tal que  $\text{imfilter}(F, K) = G$  (en la parte del filtrado que da valores válidos, esto es, cuando la máscara  $K$  está completamente dentro de  $F$ ).

### Algoritmo:

- Necesitamos una hipótesis inicial para  $F$ . Lo más sencillo es usar la imagen degradada  $G$  y aumentarla de tamaño ( $F$  es mayor que  $G$ ). Una posibilidad es partir de  $G$  y replicar su 1ª fila  $n$  veces por encima y la última fila  $n$  veces por debajo. Luego se replica su 1ª columna  $n$  veces a la izquierda y su última columna  $n$  veces a la derecha, terminando con una imagen  $(N+2n) \times (M+2n)$  que será la  $F$  inicial. [Adjuntad hipótesis inicial de partida \( \$F=G\$  ampliada\).](#)
- Entramos en un bucle donde, en cada paso:
  - Filtramos  $F$  con  $K$  usando  $\text{imfilter}$  y luego extraemos su zona central ( $N \times M$ ), (con índices  $n+1:\text{end}-n$ ) para eliminar el borde exterior de  $n$  píxeles. Estos son los valores no válidos al salirse la máscara  $K$  de la zona conocida. Llamar  $GG$  a esta zona central, que será del mismo tamaño que la  $G$  dada.

- Calcular la diferencia  $dG = (GG - G)$ . El algoritmo usará  $dG$  para construir una corrección  $dF$  con la que actualizar la hipótesis  $F$  actual.
- Para obtener  $dF$  filtraremos la diferencia  $dG$  usando `imfilter` (con la opción 'replicate'), pero con una máscara  $K$  modificada, una versión reflejada en la horizontal y en la vertical de la original: `K=flipud(flip1r(K))`.

Como además partimos de  $dG$  (del tamaño de  $G$ ) y queremos un resultado  $dF$  (del tamaño de  $F$ ) usaremos también la opción 'full' en `imfilter` (lo que hace que la salida sea del tamaño ampliado adecuado).

- Actualizar  $F = F + dF$  y luego, asegurar que los valores de  $F$  siguen estando entre 0 y 1, poniendo a 0 aquellos valores  $< 0$  y a 1 los valores  $> 1$ .

Cargar la imagen del fichero 'movida.jpg', guardarla en  $G$  y convertir a double con `im2double()`. Visualizarla con `imshow()`: veréis una imagen degradada que simula una fotografía tomada con un movimiento de la cámara. Haciendo load  $K$  tendréis la máscara (21x21) usada para obtener  $G$  a partir de la  $F$  original (desconocida). Visualizar la máscara  $K$  usando `imagesc(K); colormap(gray(255));` y adjuntadla.

Aplicad el algoritmo con 300 iteraciones. En cada paso guardar `mean(abs(dF(:)))`, el valor medio de los valores del incremento  $dF$  del paso y `mean(abs(dG(:)))` (lo mismo para la discrepancia entre la  $G$  calculada y la  $G$  real). Al finalizar el bucle adjuntad la imagen restaurada ( $F$  final) junto con la gráfica de la evolución del valor medio de  $dF$  y  $dG$  (superpuestas en ejes  $Y$  logarítmicos). Adjuntad zoom de la zona de los ojos para apreciar mejor la diferencia entre punto de partida e imagen restaurada.

Adjuntad el código de vuestro algoritmo.

Este tipo de algoritmos, aunque en este caso parecen funcionar muy bien, tienen algunos problemas en casos reales:

- Son muy sensibles al ruido, lo que es un problema en datos reales, donde siempre hay ruido presente en los datos. Añadir un poco de ruido (apenas visible) a  $G$  (una vez pasada a double):

```
G=G+0.02*randn(size(G));
```

y volver a correr el algoritmo. Adjuntad imagen final y las gráficas.

- Para que funcionen bien hay que estar seguros de la máscara  $K$  que se usó al degradar la imagen original, lo que suele ser difícil en casos reales. Volver a correr el algoritmo para la  $G$  original, pero ahora cambiando ligeramente el tamaño de la máscara  $K$ :

```
K=imresize(K,[17 17]); K=K/sum(K(:));
```

Adjuntad la imagen final y las gráficas. Notad que el  $K$  usado es muy similar al correcto, simplemente nos hemos equivocado un poco al estimar su tamaño.