Ejercicio 1: Al hablar del sensor en las cámaras digitales vimos que la mayoría de las cámaras disponen de un filtro anti-aliasing previo al sensor que "emborrona" un poco la imagen para evitar que se produzcan efectos de Moire.

Si submuestreamos una imagen digital tenemos un problema similar. Al usar menos muestras, detalles que eran captados correctamente en el original son demasiado "finos" para la nueva malla de muestreo, produciendo fenómenos de Moire.

Cargar la imagen "brick.jpg" y mostrarla con image(), usando fc_truesize para verla a su tamaño original (o tan grande como permita el tamaño de pantalla).

Submuestrear la imagen en un factor 8 en cada dimensión (quedaros con 1 fila de cada 8 y una columna de cada 8). Visualizar la imagen reducida con image + fc_truesize. ¿Se aprecia el efecto del "aliasing"? El mismo efecto podéis verlo si reducís el tamaño de la ventana que contiene la imagen original. En este caso es el sistema operativo el que realiza el submuestreo para que la imagen “entre” en la nueva ventana reducida.

Crear un filtro gaussiano G de tipo promedio con las siguientes indicaciones:

- Usar σ = 3 y un soporte desde -5 hasta 5 x = [-5:5]. El tamaño de x determina el tamaño del filtro: en este caso tendremos 11 coeficientes.
- Aplicar la fórmula h= $e^{-\left(\frac{x^2}{2\sigma^2}\right)}$ para crear el filtro 1D con la forma de una gaussiana. Normalizarlo para que su suma sea 1, usando la función sum().
- Aprovechando la separabilidad de un filtro gaussiano, construir el filtro 2D haciendo el producto cruzado G = h'*h (G debe ser una matriz 11x11)

Adjuntar los coeficientes (G) del filtro resultante.

Filtrar la imagen original usando la fc_imfilter(im,G,'symmetric) y submuestrear la imagen filtrada con el mismo factor 8 de antes. Adjuntar ambas imágenes submuestreadas (con y sin filtro previo). ¿Se han reducido los efectos de "Moire"?

Ejercicio 2: En muchas ocasiones se desea un filtrado que tenga un efecto OPUESTO al filtro promedio anterior. En vez de hacer que los detalles se pierdan queremos realzarlos. Es lo que se llama un filtro de realce de bordes o "sharpening".

Imaginad que aplicamos un filtro promedio G a una imagen prom = filtro(im,G)

En prom habrá desaparecido gran parte del detalle de im debido al filtrado. Por lo tanto podemos extraer el "detalle" de im haciendo:

im – prom = (imagen original) - (imagen sin detalle) $\cong$ detalle

Obviamente si hacemos im2 = prom + detalle recuperamos la imagen original.

Podemos incrementar el detalle de la imagen sumándole más detalle del que tenía:

im2 = prom+2*detalle, o en general, im2 = prom + K*detalle (con K>1)

Hacer un "sharpening" de la imagen contenida en "test.jpg".

- Para el filtro promedio usar el filtro gaussiano anterior (x=[-5:5] y σ = 3).
- Tras separar el detalle reforzarlo usando K=2 en la fórmula anterior.

Mostrar en la misma figura la imagen original y la resaltada usando subplot(211) y subplot(212). Las imágenes son en B/W así que deberéis usar colormap(gray(256)) para verlas correctamente.

Adjuntar la figura resultante. Notad como la textura del hielo se ve más marcada en la segunda imagen. ¿Notáis algún efecto indeseable en la imagen procesada?

Este efecto es inevitable si usamos filtros de promediado lineales, debido a como estos filtros degradan los bordes de una imagen. Considerar la "imagen" 1D generada por los siguientes comandos MATLAB:

```
t = (0:0.01:1); y(t<0.4)=60; y(t>=0.4)=80; y = y+sin(2*pi*t*20);
```

La "imagen" simula un borde entre un valor 60 a la derecha y 80 a la izquierda. Las pequeñas oscilaciones en ambas zonas representan el "detalle fino" de la imagen que tratamos de realzar.

Crear una máscara gaussiana g 1D normalizada con x=[-2:2] y s=2. Usarla para filtrar la señal y usando el comando MATLAB:

```
yy = conv(y,g,'same');
```

Calcular el detalle como (org – promedio) = (y-yy). Eliminar las primeras y últimas muestras del "detalle" (erróneas debido al problema de aplicar un filtro en los extremos) haciendo:

```
detalle(1:3)=NaN; detalle(end-2:end)=NaN;
```

Hacer un plot del "detalle" extraído y adjuntar imagen resultante.
¿Hemos capturado las pequeñas oscilaciones que simulaban el detalle de la imagen? ¿Qué más aparece en la gráfica?

Superponer una "imagen" de la señal original (y) y de una señal con el detalle resaltado (yy + 3*detalle). Adjuntar la gráfica resultante. ¿Se ha magnificado el detalle? ¿Qué ocurre en el borde?

Proyecto: representaciones en pirámide. (a entregar, por parejas, dentro de dos semanas)

1) Implementación de la pirámide Laplaciana

Se trata de escribir una función implementando la pirámide laplaciana para imágenes BW o color. Usaremos el template suministrado en lap.m

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

p=cell(1,N);

% Crear filtro gaussiano 2D a usar
...

% Calcular los niveles p{1}, p{2}, ..., p{N} de la piramide laplaciana
% Recordad que p{N} es la versión reducida de la imagen
im=double(im);
for k=1:N-1,
  ...
end

return
```

La función recibe una imagen im (en blanco/negro o en color) y el número de "pisos" de la pirámide. Si no se indica se usan N=5 niveles. Recordad que N=5 significan 4 niveles de detalle + 1 versión reducida de la imagen original.

Los resultados de los diferentes niveles se guardarán en p, una array de celdas en MATLAB. Un array de celtas es como un vector o tabla normal pero que puede tener diferentes tipos de datos en sus casillas (en este caso matrices con diferentes tamaños). La única diferencia práctica es que se hay que usar llaves en lugar de paréntesis para acceder a sus elementos: p{1},p{2}, … en lugar de p(1), p(2), etc.

Para completar la función debéis en primer lugar definir el filtro gaussiano G a usar para suavizar la imagen previo a su submuestreado. Crear un filtro gaussiano como el usado en el ejercicio 1, con un soporte x=[-5:5] y un ancho $\sigma = 3$.

El proceso es sencillo. En cada paso dentro del bucle desde k=1 a N:

1) Filtrar la imagen (fc_imfilter) usando el filtro gaussiano G para obtener la imagen suavizada.

2) Restar la imagen original – suavizada = detalle y guardarlo en el nivel k de p{}

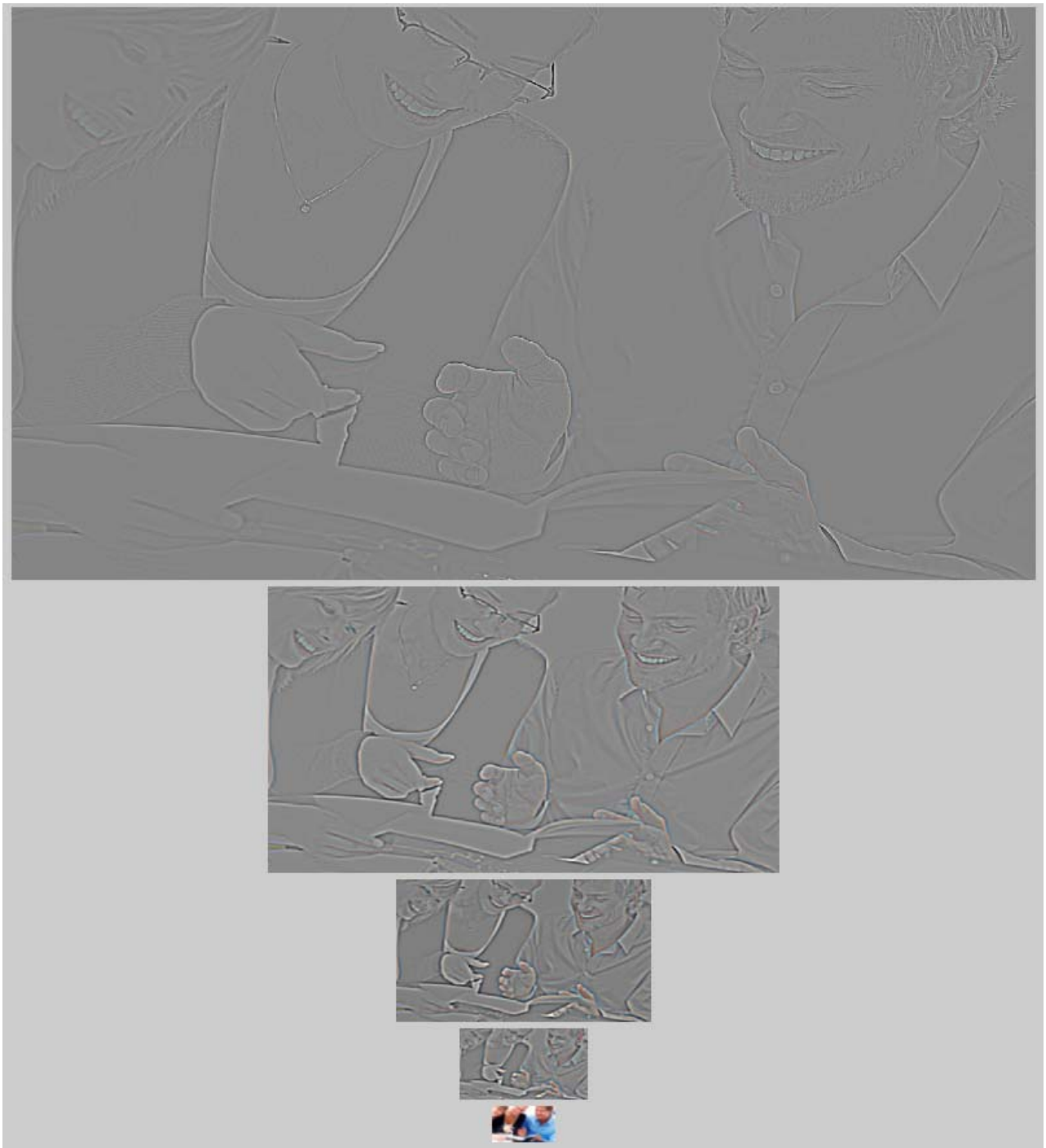
3) Submuestrear la imagen suavizada y volver a repetir el bucle.

Al salir del bucle guardar la última versión de la imagen submuestreada en el último nivel de la pirámide, p{N}. Adjuntar código de vuestra función lap.m

Usar la imagen "faunia2.jpg" de tamaño 896 x 608. Verificar que los 5 niveles de la pirámide tienen tamaños:

896 x 608, 448 x 304 , 224 x 152, 112 x 76, 56 x 38

Usar la función suministrada visualiza_lap(p); para visualizar los diferentes niveles de la pirámide. Debéis obtener algo similar a la figura adjunta (pero para vuestra foto de loros). Adjuntar la imagen obtenida en vuestro caso.



2) Inversa de la pirámide Laplaciana

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

```
function im=invlap(p)
N = length(p); % Número de níveles de la piramide
im=p{N};  % Inicializamos con version tamaño sello.

...
end
```

Tras saber cuantos niveles tiene la pirámide, inicializamos la imagen im a recuperar con el último nivel de la pirámide (versión sello de la imagen original). Luego solo hay que hacer un bucle barriendo el resto de los niveles. En cada paso:

1) Ampliar la im por un factor 2 (usando la función amplia_2) suministrada.
2) Sumar a im el detalle correspondiente a ese nivel, p{k}.

Al terminar tendremos una imagen de tamaño original. Convertir a bytes (uint8) antes de devolverla, para poder luego visualizarla correctamente.

Adjuntar código de vuestra función invlap.m

Visualizar la imagen original y la recuperada y comprobad que son muy similares. Para medir sus diferencias calcular la media (mean2 o fc_mean2) del valor absoluto de la resta de ambas imágenes (pasar a double las imágenes antes de hacer las cuentas).

Comando usado para calcular la discrepancia entre las imágenes original y recuperada. ¿De que orden es dicha diferencia?

Aplicación: Fusión de imágenes con pirámide Laplaciana.

Una vez implementada la pirámide Laplaciana vamos a usarla en una aplicación de fusión de imágenes (consultar la descripción en el material de clase).

Datos de partida:

- Dos imágenes, im0+im1, ambas del mismo tamaño NxM (x3 si son en color).
- Máscara (mask) de fusión (también NxM o NxMx3): una máscara binaria donde un 0 indica que aparezca im0 en la imagen final, y un 1 que sea im1.

A partir de estos datos podríamos aplicar una fusión directa:

```
im = (1-mask).*im0  + mask.*im1
```

O también (al ser mask una máscara de 1/0) usando indexado lógico de MATLAB:

```
im = im0;  im(mask)=im1(mask);
```

Una fusión por bandas consiste en:

1) Hallar las **pirámides laplacianas** (p0, p1) de las dos imágenes (im0, im1) usando la función lap que habéis escrito antes.

2) Calcular una **piramide gaussiana** (m) de la mascara mask. En una pirámide gaussiana cada nivel guarda una versión progresivamente más suavizada (ver detalles en transparencias). Suavizar con el mismo filtro gaussiano (soporte x=[-5:5] y ancho σ = 3) usado en la pirámide laplaciana.

3) Se crea una pirámide laplaciana compuesta, donde cada nivel se halla combinando el correspondiente nivel de ambas pirámides (p0, p1), usando el nivel adecuado de la pirámide gaussiana de la máscara:

```
new{k}  = (1-m{k}).*p0{k}  + m{k}.*p1{k}
```

4) Se "colapsa" la pirámide mixta así obtenida usando vuestra función invlap para invertir la pirámide laplaciana, obteniendo así la imagen fusionada.

La imagen de partida se halla en el fichero “face.jpg”. Esta será la 1ª imagen im0. Cargar y visualizar.

Vamos a añadirle un tercer ojo. Para ello lo primero es crear la 2ª imagen (im1). Con el cursor de datos determinar el centro del ojo (p.e. el derecho en la foto) y extraer una subimagen alrededor de dicho centro (X0,Y0). El tamaño de la subimagen debe ser lo suficientemente grande para que "capture" todo el ojo.

Elegir un punto adecuado de destino X1,Y1 en la misma imagen. Crear una copia de la imagen im1=im0 e insertar en (X1,Y1) el trozo extraído antes. Las imágenes im0 e im1 a fusionar deben ser algo parecido a las mostradas aquí:

A continuación usando la función suministrada crea_mask creamos una máscara binaria de fusión:

```
mask = crea_mask(im0,[X1 Y1]);
```

La función crea una máscara binaria del mismo tamaño que la imagen im0 con forma elipsoidal y centrada en la posición elegida de destino [X1,Y1]. Podéis visualizar la máscara haciendo simplemente image(mask). Verificar que está en la posición donde habéis colocado el 3er ojo.

Las dimensiones de la máscara pueden cambiarse dando un tercer argumento w a la función crea_mask. Este tercer argumento es un vector w=[wx wy] con el ancho (en píxeles) para ambas dimensiones de la máscara. Los valores por defecto se ajustan al tamaño del ojo en esta foto.

Adjuntar vuestra imagen im1. Hacer una fusión directa con la máscara binaria. Adjuntar código e imagen final.

Hacer fusión usando pirámides tal como se ha descrito anteriormente, usando 5 o 6 niveles en las pirámides (tanto en las pirámides laplacianas de las imágenes como en la pirámide gaussiana de la máscara). Adjuntar vuestro código y la imagen final.

En la imagen adjunta se aprecia un detalle de la diferencia entre la fusión directa (izquierda) y la fusión con pirámides.

EXTRA: Cuando hayáis replicado estos resultados, usar vuestro programa con una foto o fotos de vuestra elección. Tened en cuenta que:

- Dependiendo del objeto a "fusionar" tendréis que cambiar el tamaño de la máscara usando el tercer argumento opcional de la función crea_mask.

- Vuestras funciones lap e invlap solo van a funcionar correctamente si las dimensiones (alto/ancho) de la imagen son múltiplos de una potencia de 2 suficientemente alta para seguir siendo enteros después de todas las divisiones por 2 que hacemos en cada nivel de la pirámide. Tendréis que recortar las imágenes elegidas para que tengan las dimensiones adecuadas.

¿Cómo podríais modificar lap (e invlap) para que funcionaran con imágenes de un tamaño arbitrario?