PROYECTO 4

En esta práctica veremos aplicaciones de varios tipos de filtros (gaussiano, bilateral)

1. Realce de bordes: filtro de nitidez o unsharp.

En esta apartado vamos a presentar un filtro clásico (ya conocido y usado en la fotografía analógica): el llamado filtrado de nitidez o realce de bordes, que trata de mejorar la nitidez de una imagen. Es un filtro que se suele aplicar a las imágenes raw en nuestra cámara ya que, como vimos en el LAB anterior, dichas imágenes son obtenidas a partir de una interpolación de sus canales, lo que resulta en imágenes más "suaves" de lo que sería preferible.

En Photoshop este filtro es tradicionalmente conocido filtrado "unsharp" y tiene tres parámetros: radio, cantidad y umbral. En este apartado explicaremos el significado de estos parámetros. La razón de denominar "unsharp" a un filtro que hace justo lo contrario (hacer más nítida o "sharp" una imagen) es que durante el proceso se utiliza una versión desenfocada (unsharp) de la imagen original.

Para construir la imagen desenfocada aplicaremos un filtro gaussiano con ancho σ, que corresponde al radio o primer parámetro del filtro. Por lo tanto lo primero es crear con fspecial() una máscara gaussiana Gs con ancho σ=2 Esta máscara cubrirá un rango de ±L respecto al punto de aplicación. Usando el criterio de que $L=2\cdot\sigma$, el tamaño del filtro a especificar en fspecial() será de $2\cdot L+1 = \text{round}(4\cdot\sigma+1)$.

Cargad la imagen del fichero "img1.jpg" y convertirla a double, pero manteniendo sus valores entre 0 y 255 (usad la función double() en vez de im2double()). Luego, filtrarla con el filtro Gs usando imfilter(). Como opción para los bordes usaremos "symmetric". Si visualizáis el resultado im_S veréis su pérdida de detalle respecto a la original. Calculando su resta entre ambas imágenes obtenemos el detalle eliminado por el filtrado: $im - im_S$ = (imagen) - (imagen - detalle) = detalle,

Vamos a escribir una función `function show_detail(det)` para visualizar este detalle así obtenido. Este detalle (la resta entre dos imágenes RGB) constará de tres planos de color con valores positivos y negativos (el valor del detalle es una medida de la diferencia entre cada píxel y sus vecinos). Para visualizarlo, primero calcular el valor absoluto de la imagen detalle y luego reducirla a un solo plano combinando sus tres planos como $0.3\cdot R+0.55\cdot G+0.15\cdot B$. Finalmente visualizar la imagen resultante usando la función imagesc() y dibujar a su lado una escala asociando los colores usados a los valores de la imagen: `colorbar('vert');`

Adjuntad código de vuestra función y la imagen obtenida para el detalle anterior. Los valores más bajos se dan en zonas homogéneas (cielo) con poco detalle. Luego hay un rango de valores intermedios en las zonas con textura (como la roca) y finalmente los valores más altos los tenemos en los bordes de la imagen original, donde había píxeles vecinos muy diferentes entre sí (a ambos lados de un borde).

Una vez que hemos extraído el detalle podemos realzarlo en la imagen original haciendo: im2 = im + α·detalle. Usando α>0 lo que estamos haciendo es añadir una "ración" extra de "detalle" a la imagen original. Este parámetro α corresponde al parámetro de "cantidad" del filtro de Photoshop, donde se suele expresar como un porcentaje (por ejemplo, añadir un 100% de detalle correspondería a usar α=1).

Juntad todas las etapas para hacer un realce de bordes de la imagen contenida en "img1.jpg":

- Leed la imagen y convertirla a double con valores entre 0 y 255.
- Aplicar un filtro gaussiano con S=σ=2 para obtener la imagen filtrada im_s.
- Aislar el detalle restando im_s de la imagen original.
- Tras aislar el detalle, reforzarlo en la imagen usando α=1.5 (150%) en la fórmula anterior.

Construir una imagen formada en su mitad superior por la imagen realzada y en su mitad inferior por la original. Adjuntad la imagen "compuesta". Observad como se ha realzado la textura de la roca.

Este filtrado es sencillo pero viene con algunos efectos secundarios no deseados:

- El realce no diferencia entre el genuino detalle que deseamos realzar (como la textura de la roca) y el inevitable ruido de una imagen (visible p.e. en zonas homogéneas como el cielo): ambos serán realzados.

- Otro problema típico son los “halos” que aparecen (predominantemente en los bordes de la imagen original).

Haced un zoom en alguna zona de la imagen compuesta que incluya la imagen original y la realzada y donde se aprecien estos efectos (halos + elevación del nivel de ruido en la zona del cielo). Adjuntad zoom. ¿A qué son debidos estos halos que aparecen?

Para reducir el efecto de la amplificación del ruido se puede fijar un umbral U, de tal forma que si el valor absoluto del detalle es menor que U se considera que es ruido y no se amplifica en la imagen final. Si se supera el umbral U se considera que es verdadero detalle y se amplifica con la fórmula anterior. Este umbral U es el tercer parámetro del filtro unsharp de Photoshop, junto con el radio del filtro gaussiano aplicado (~ σ) y la cantidad de detalle extra añadido (~ α). Sobre la imagen anterior del detalle mover el cursor de datos sobre la zona del cielo y la de la roca. ¿Qué rango de valores típicos observáis en ambas zonas? Con estos resultados se podría decidir el umbral U que hay que usar para diferenciar entre el verdadero detalle que queremos ampliar (como la textura en la roca) y el ruido (zona del cielo) que no deseamos amplificar.

Respecto al problema de los halos, veremos en el siguiente apartado como podemos reducirlos usando un filtro bilateral en vez de gaussiano al promediar la imagen.

2. Implementación de un filtrado bilateral

Un filtro bilateral, además de la **distancia al punto de aplicación** (como el caso de un filtro gaussiano) también tiene en cuenta la **diferencia entre los valores** de los píxeles a promediar. La diferencias entre los colores de los píxeles vecinos y el color del píxel de aplicación se evalúan dentro de una segunda gaussiana (con un ancho distinto σr, r de rango) y modifican los valores de los coeficientes a usar.

La salida de un filtro bilateral (con ancho σs en el espacio y ancho σr en el rango) aplicado a una imagen I para el píxel con posición $\underline{p}$ y valor en dicho punto $I(\underline{p})$ es:

$$I_{BF}[\overline{p}] = \sum_{\mathbf{q}\in Q} G_{\sigma_s}(\|\mathbf{p}-\mathbf{q}\|)\cdot \frac{G_{\sigma_r}\left(\left\|I_{\mathbf{p}}-I_{\mathbf{q}}\right\|\right)}{W_{\mathbf{p}}}\cdot I_{\mathbf{q}}$$

El primer termino del sumatorio anterior es la parte "espacial" del filtro y es idéntica a la del filtro gaussiano: depende de la distancia entre los puntos $\underline{p}$ y $\underline{q}$ y el ancho espacial σs. Los coeficientes a usar corresponden a la máscara Gs que se calculaba en el apartado anterior. Como entonces, esta parte puede calcularse previamente.

La segunda parte es la que depende de la diferencia ||I(p) - I(q)|| entre el color del píxel I(p) en el que aplico el filtro y el valor I(q) del vecino que estoy considerando promediar. En general los valores serán en general colores por lo que al restarlos tendremos un vector 3D, cuya norma será la distancia a usar en la fórmula de la gaussiana correspondiente (ahora con un ancho distinto σ_R)

Esta nueva máscara G_R tiene que recalcularse para cada punto $\underline{p}$ porque depende de los píxeles vecinos. Tras calcularla, se multiplica por los coeficientes "espaciales" G_s (constantes para toda la imagen) y obtenemos la máscara a usar en ese punto $\underline{p}$. Antes de aplicarla hay que normalizarla (factor 1/Wp) para que su suma sea 1. Tras la normalización, la máscara resultante es la que finalmente se usa para promediar los puntos q del soporte Q alrededor del punto $\underline{p}$. El resultado es la salida del filtro bilateral en la posición $\underline{p}$, $I_{BF}(p)$. Obviamente, este filtrado bilateral es más costoso de calcular que el gaussiano al tener que recalcular sus coeficientes en cada punto. Hacerlo de forma eficaz no es trivial, pero una implementación directa de la expresión anterior no es difícil (aunque será bastante lenta).

Para implementar el filtro bilateral partiremos de la función filtra_gauss(im,S) que os doy y que implementa un filtrado gaussiano con σ = S usando la opción simétrica para los bordes. Podéis comprobar que la salida de **`r1=filtra_gauss(im,2)`** es igual a hacer **`S=2; L=2*S; H=fspecial('gaussian',2*L+1,S); r1=imfilter(im,H,'sym');`**
Eso sí, el primer comando será mucho más lento.

Vamos a guardar esta función como **`filtro_bilat.m`** y la modificaremos para incluir la parte que depende de la diferencia de colores en la imagen. Lo primero es añadir un parámetro adicional de entrada R, el ancho (σ_R) de la gaussiana aplicada a la diferencia de colores entre píxeles: **`function res=filtro_bilat(im,S,R)`**

Gran parte del código va a ser el mismo: la ampliación inicial de la imagen (para cumplir la condición de bordes simétricos), la creación de la máscara gaussiana, los bucles barriendo filas y columnas, etc.

Lo único que hay que modificar es el cálculo final de los coeficientes dentro de los bucles añadiendo la parte de las diferencias entre valores. Para ello:

1) Una vez extraída la vecindad vec calculamos la diferencia D entre vec y el píxel central im(k,j,:) y la dividimos por ancho R. El resultado es una matriz (2L+1)·(2L+1)·3 donde el píxel central vale (0,0,0) al haber sido restado. Luego, elevad al cuadrado los valores de D y sumarlos a lo largo de su 3ª dimensión (color), terminando con una matriz D2 de tamaño (2L+1)·(2L+1)

2) A la matriz resultante (cuyos valores se corresponden con $||Ip-Iq||^2/R^2$ para los píxeles de la vecindad) le aplicamos una exponencial para obtener los coeficientes de la 2ª gaussiana:

```
Gr= exp(-0.5*D2)
```

3) Multiplicad (punto a punto) la nueva máscara Gr (coeficientes de rango) por Gs (coeficientes espaciales) para obtener la máscara G. Finalmente hallar la suma de los valores de G y dividir G por esa suma para que sus coeficientes sumen 1 (antes esto no era necesario porque Gs ya estaba normalizada y no cambiaba durante todo el proceso).

4) Usad la máscara G final (en vez de Gs) para multiplicarla por la vecindad.

Como veis el proceso es similar, aunque será aún más lento al tener que recalcular los valores de la máscara G=Gr.*Gs para cada punto de la imagen, en vez de usar siempre la misma máscara precalculada Gs como en el caso del filtro gaussiano.

Cargad la imagen de "img1.jpg", convertirla a double (manteniendo rango 0-255) y aplicad el nuevo filtro bilateral con (S=2, R=20) con:

```
f1=filtro_bilat(im,S,R);
```

Para comprobar que está correctamente implementada comparar este resultado con la función de Matlab **`imbilatfilt()`** que implementa un filtro bilateral. Para que sean equivalentes usad los siguientes parámetros:

```
f2=imbilatfilt(im,R^2,S,'Padding','symmetric','NeighborhoodSize',2*L+1);
```

Notad que imbilatfilt() recibe R^2 en vez de R como parámetro. Calcular la diferencia (f1-f2) entre los resultados y volcar su máximo y mínimo (que debería ser del orden de la precisión de la máquina $\sim 10^{-16}$). Una vez verificada, adjuntad el código de vuestra implementación.

Como nuestra implementación será bastante lenta, para hacer pruebas y comparar con la función de MATLAB podéis usar una versión reducida de la imagen haciendo p.e. im=imresize(im,[400 600]). Cuando os funcione dar los resultados para la imagen original.

Usando la imagen original calcular el detalle como antes: det=(im-f1) y visualizarlo con la función show_detail(). Adjuntad la imagen resultante. ¿Qué rango de valores tenemos ahora en el detalle? ¿Cuál es la mayor diferencia con la imagen del detalle que se obtuvo en el apartado anterior usando el filtrado gaussiano?

Observaréis que los valores máximos del detalle son ahora bastante más pequeños que antes, lo que nos permite aumentar su % (valor de alfa) al realzar los bordes. Adjuntad la imagen realzada con α=250%. Hacer un zoom sobre zona donde antes aparecían halos. Adjuntad captura del zoom. A pesar de usar un "refuerzo" bastante mayor, los halos son menos visibles que con el filtro gaussiano.

Finalmente partid de la imagen original y aplicad 4 veces un filtro gaussiano con un ancho S=5 (aplicando cada filtrado al resultado del filtro anterior). Repetir para el filtro bilateral (con el mismo ancho espacial S=5 y usando R=20). Usad la función de Matlab para que sea más rápido. Adjuntad las 2 imágenes resultantes.

2b. Cross-bilateral filter

Vamos a hacer una pequeña modificación al filtro anterior para implementar el filtro bilateral cruzado (cross-bilateral filter):

```
function res=filtro_cross(im,im2,S,R)
```

La única diferencia en los parámetros es que la función recibe ahora una segunda imagen im2 (que debe ser del mismo tamaño que la primera).

El filtro bilateral cruzado o conjunto aplica los coeficientes G del filtro bilateral a la vecindad correspondiente de la imagen im como hacíamos antes (devolviendo el resultado en la salida res). La diferencia es que para calcular dichos coeficientes no se usa la propia vecindad de im como antes, sino los datos de la segunda imagen im2. Modificar vuestro código para que implemente este nuevo tipo de filtrado:

- Antes del bucle, convertir im2 a double y ampliarla de la misma forma que se hacía con im.

- En el bucle, extraer dos vecindades alrededor del píxel (k,j): vec (procedente de im) y vec2 (de im2). En el proceso de calcular la máscara G usad los datos de la vecindad vec2 y la imagen im2 (2ª imagen) pero una vez obtenidos los coeficientes, aplicadlos (como antes) a la vecindad vec (de la 1ª imagen).
  Adjuntad las línea/s de código cambiadas en vuestra función

Aplicación: Cargar las imágenes flash.jpg y no_flash.jpg y visualizarlas. Las fotos corresponden a dos tomas de la misma escena, una usando el flash y la otra sin él. La imagen sin flash es muy ruidosa pero muestra las sombras de la luz original y tiene un tono de color cálido que se quiere preservar. La imagen con flash es mucha más limpia, pero con una luz "fría" y sin sombras. Como nos gusta el "ambiente" de la foto sin flash, vamos a tratar de "limpiarla", reduciendo su ruido.

Para reducir el ruido de una imagen podemos aplicarle un filtro gaussiano. La idea es que al promediar los valores de una vecindad el ruido (si es aleatorio) tenderá a reducirse. Aplicad un filtro gaussiano (S=5) sobre la imagen sin flash. Observar que el filtro gaussiano reduce bastante el nivel de ruido en las zonas homogéneas dentro de los objetos (jarrones, sofá, mesa), pero deja muy borrosos los bordes entre los objetos y las zonas con detalles finos (dibujos de los jarrones).

Usar ahora un filtrado cross-bilateral (con el mismo S=5 y R=10) para filtrar la imagen sin flash, usando la imagen con flash como la segunda imagen (usada para determinar los coeficientes). Juntad la imagen original y el resultado de ambos filtrados (solo gaussiano + bilateral cruzado) en una única imagen haciendo im=[im1 im2 im3]. Adjuntad figura. ¿Por qué con el filtro bilateral se preservan mejor los detalles finos de la imagen como los dibujos de los jarrones?

3. Mejora del contraste: ecualización del histograma

Leed la imagen de "img2.bmp" y visualizarla con imshow(). Es una imagen (color) de una escena con un rango dinámico alto. La exposición se ajustó para evitar que las zonas más claras se "quemaran", por lo que las zonas a la sombra (muy extensas en la imagen) han quedado poco expuestas (muy oscuras).

Vamos a comprobarlo calculando su histograma usando la función histcounts() de MATLAB:

```
v=(0:255); edges=v(1)-0.5:v(end)+0.5; h = histcounts(im(:),edges);
```

donde v es un vector con los valores posibles de los píxeles (de 0 a 255). La función devuelve un vector h con el número de píxeles encontrados para el correspondiente valor de v. Hacer un plot de h en función del valor del píxel (v) usando la función bar() cuyo uso es similar a plot pero usando un gráfico de barras.. Adjuntad gráfica. ¿Qué valor es el más común en la imagen? ¿Y el más raro? ¿Cuántas veces aparecen cada uno de esos valores en la imagen?

Como queremos trabajar con probabilidades, dividid el vector h por el número total de píxeles de la imagen (multiplicados por 3, ya que hemos hecho el histograma de los tres canales juntos). Tras dividir comprobad que la suma del nuevo vector h es ahora 1. De esta forma h(k) representa la probabilidad de que un cierto píxel tenga un valor v(k). ¿Qué probabilidad (en %) tenemos de encontrar un valor de 50 en alguna de las componentes RGB de un píxel?

Usaremos este histograma normalizado para construir de forma automática una transformación punto a punto T(x) que, aplicada a la imagen original, nos dé una imagen con un histograma más plano, "mejor" repartido entre todos los niveles.

La teoría que hemos visto nos dice que T(x) puede aproximarse por $T(x)=\sum_{m=1}^{x} h[m]$

de forma que T(1)=h(1), T(2)=h(1)+h(2), T(3)=h(1)+h(2)+h(3), ...

La función cumsum() de MATLAB calcula justamente este sumatorio acumulativo. Calculad T y adjuntad su gráfico (plot). Se ve que T(x) es una función siempre creciente que toma valores de 0 a 1. Esto será así para cualquier imagen. ¿Por qué?

Ahora solo falta aplicar la transformación encontrada a la imagen original. Se trata de cambiar cada valor de la imagen por la correspondiente entrada de la tabla T que hemos construido (se valorará no usar bucles). Como la función T tiene un rango de calores entre 0 y 1, la imagen resultante tendrá valores entre 0 y 1. Multiplicarla por 255 para que tenga el mismo rango de valores (0-255) que la imagen original. Adjuntad el código usado para obtener la nueva imagen. Juntad la imagen resultante con la original formando una nueva imagen poniendo una al lado de la otra y visualizarlas con imshow(). Adjuntad la imagen resultante.

Calculad el histograma de la nueva imagen, visualizarlo y adjuntadlo.

Observaréis que el resultado dista mucho de ser plano. Esto es debido al hecho de que al trabajar con valores discretos (0,1, ..., 255) no es posible "diferenciarlos" para distribuirlos uniformemente en el nuevo histograma. Se puede apreciar mejor cómo la nueva imagen tiene un histograma más plano calculamos los histogramas con un número menor de niveles (p.e. 32 en vez de 256, de forma que cada nuevo nivel junte ahora 8 de los antiguos valores). Haciendo h=histcounts(im(:),32) calculad los histogramas de la imagen original y la transformada usando sólo 32 niveles. Visualizar ambos en la misma figura usando subplot(121) y subplot(122). Adjuntad la figura resultante. Ahora se debería apreciar como el histograma de la imagen procesada es mucho más plano que el de la original.

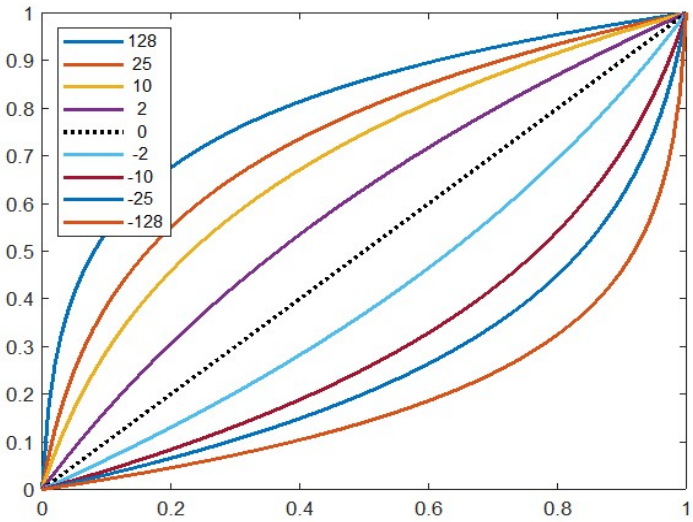
El problema de esta imagen en particular es que aunque una mayoría de píxeles son oscuros y queremos aclararlos también tiene bastantes valores altos (como se apreciaba en el histograma original). De hecho, considerada globalmente, la imagen original tiene un buen contraste ya que tiene valores oscuros (cerca de 0) y también claros (cerca de 1). Si miráis la gráfica de la T(x) obtenida con el histograma acumulativo veréis que la salida está por encima de la entrada para TODOS los valores. Al aclarar TODOS los píxeles se mejora el contraste en la zona oscura pero se pierde en la zona clara. Observad como los edificios blancos a la izquierda del río han perdido el detalle, o como el fondo de montañas se ve más desvaído. La razón es que la función aplicada T(x) junta los valores altos reduciendo el contraste en esas zonas.

En el siguiente apartado veremos cómo tratar de solucionar este problema usando los llamados algoritmos de mejora LOCAL del contraste.

3b. Mejora local del contraste

Hay muchos algoritmos propuestos para mejora local del contraste: el que vamos a usar podéis encontrarlo en esta referencia. Es importante indicar que tal como está descrito este algoritmo trabaja con imágenes normalizadas entre 0 y 1, por lo que usaremos im2double() para normalizar la imagen de partida tras leerla del fichero.

Lo que hace el algoritmo es aplicar unas transformaciones similares a las mostradas en la figura adjunta. Estas transformaciones son similares a las ya conocidas. Aquellas etiquetadas con valores positivos (por encima de y=x, la línea negra discontinua) expanden los tonos oscuros y contraen los claros. Por el contrario, las correspondientes a números negativos (debajo de la línea discontinua) expanden las luces y son más adecuadas para dar más contraste a una imagen excesivamente clara.



Nuestro problema es que en la misma imagen (como en el ejemplo del apartado anterior) podemos tener zonas oscuras a "expandir" pero también zonas claras que se verán perjudicadas por dicha transformación. Una solución sería usar distintas transformaciones según la zona de la imagen. De las transformaciones anteriores aplicaríamos aquellas con valores positivos en zonas predominantemente oscuras y viceversa. Para decidir si el entorno de un punto es claro o oscuro podemos usar un filtro promedio (tipo gaussiano o bilateral) aplicado previamente sobre la imagen. En este ejercicio, para el filtrado bilateral, podéis usar la función de MATLAB (mucho más rápida) en vez de vuestra implementación. Ya vimos que daban los mismos resultados si tenemos en cuenta la forma diferente de pasarle los parámetros.

Una vez cargada la imagen ("img2.bmp") y convertida al rango [0,1] con im2double el algoritmo consiste de las siguientes etapas:

1. Obtener con rgb2gray una versión BW de la imagen. Guardarla en I.

2. Aplicación del filtro promedio a la imagen I para obtener una imagen w. Luego re-escalar w de forma que su valor mínimo vaya al 0 y su valor máximo a 1. Los valores de w indican si la vecindad de un punto es predominantemente clara (w>>) u oscuro (w<<). Usaremos un filtro bilateral con σ_S=5 y σ_R=0.05. El valor de σ_R es ahora mucho más pequeño que antes porque las imágenes son ahora entre 0 y 1 en vez de entre 0 y 255. Para el soporte espacial usaremos como antes 2·L+1 siendo L=2· σ_S. Si usáis la función de MATLAB recordad que el parámetro que se le pasa es R^2, no R.

3. A partir del promedio w (entre 0 y 1) se obtiene una imagen alfa del mismo tamaño (con valores entre -128 y 128) de acuerdo a la siguiente fórmula:

$$\alpha(w) = \begin{cases} 128 \cdot \left(1 - \left(\frac{w}{0.5}\right)^{\gamma}\right) & si \quad w \le 0.5 \\ -128 \cdot \left(1 - \left(\frac{1-w}{0.5}\right)^{\gamma}\right) & si \quad w > 0.5 \end{cases}$$

Para el parámetro γ usaremos γ=0.05. Estos valores alfa corresponden al parámetro de las curvas mostradas en la figura anterior (curvas con alfa<0 realzan los tonos oscuros y las de alfa>0 los claros). Escribid una función **`function alfa=f_alfa(w,G)`** que reciba la imagen w y parámetro γ y devuelva la imagen alfa. Como w tiene valores distintos en cada punto de la imagen, alfa también tendrá valores diferentes, en vez de usar un valor común para toda la imagen. Adjuntad vuestro código de la función (se valorará no usar bucles).

4. Finalmente para cada píxel (x,y) de la imagen I usamos su valor I(x,y) y el parámetro alfa(x,y) obtenido antes para la misma posición y aplicamos la siguiente transformación:

$$L_\alpha(x) = \begin{cases} \dfrac{\log(1+\alpha\cdot x)}{\log(1+\alpha)} & si \quad \alpha > 0 \\ x & si \quad \alpha = 0 \\ 1 - \dfrac{\log(1+|\alpha|(1-x))}{\log(1+|\alpha|)} & si \quad \alpha < 0 \end{cases}$$

De esta forma en cada punto aplicamos una curva de transformación distinta (un valor de α distinto, que a su vez depende del nivel medio de la imagen en el entorno del punto). Llamaremos I_out a la imagen resultante de aplicar esta transformación a I. Adjuntad código de **`function I_out=L_alfa(I,alfa)`**

5. Hallar la relación entre I_out e I como el cociente R = I_out/(I+0.001). El factor 0.001 es para evitar un resultado de ∞ en los valores nulos de la imagen de partida. Haced la división punto a punto para obtener una matriz R del mismo tamaño que la imagen.

6. Multiplicar la matriz R así obtenida por cada plano de color de la imagen de partida im. La imagen resultante será la imagen final del proceso.

Adjuntad código del proceso completo. Tras aplicar el procedimiento a la imagen indicada, mostrad los pasos intermedios (alfa y ratio R) como sendas imágenes (usad imagesc). Acompañarlas de una barra vertical al lado mostrando la relación entre colores y valores. Adjuntad imágenes resultado.

Construid una imagen poniendo juntas (con el operador []) la imagen original, la imagen final de este algoritmo y (como comparación) la imagen obtenida aplicando la transformación $\sqrt{x}$. Adjuntad el resultado. Indicad en qué zonas este algoritmo muestra mejoras frente a usar directamente la raíz cuadrada.

Finalmente, en vez del filtro bilateral usad un filtro gaussiano para calcular w. Para la máscara gaussiana usad el mismo ancho espacial σ=5 y tamaño 2·L+1 que antes. Repetir los cálculos para este nuevo w y adjuntad la imagen final. ¿Qué problemas se observan ahora que no aparecían con el filtro bilateral? ¿Cuál es su causa?

4. Deconvolución de una imagen

En el filtro del apartado 1) el objetivo era tratar de hacer una imagen más nítida sin saber nada sobre el origen del problema. Esto puede funcionar para imágenes que no son muy malas de partida, pero para imágenes con alteraciones o desenfoques más serios es necesario conocer los detalles del proceso de adquisición si queremos intentar una recuperación.

En una primera aproximación, muchos 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 sería importante saber invertir las operaciones de filtrado que hemos estado usando. Se conoce como deconvolución el problema en el que se parte de una imagen degradada G, obtenida como un filtrado de la imagen original F (que es la que queremos hallar) usando una máscara o filtro K (que conocemos):

G = imfilter(F,K)

El nombre deconvolución indica que se trata de "deshacer" o invertir la operación de convolución (o filtrado) para recuperar la imagen original F. La máscara o "kernel" K recoge los detalles de la degradación sufrida por la imagen original.

Este problema es complicado porque al aplicar una máscara K a una imagen F se 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. No hay un algoritmo directo que nos permita invertir la operación de filtrado, pero lo que siempre podemos hacer si tenemos una posible solución F* es filtrarla con la máscara K y comparar el resultado G* con la G que nos han dado. En función de sus diferencias podemos modificar la hipótesis inicial y se vuelve a comprobar.

Cargar la imagen del fichero 'degradado.png', guardarla en G y convertir a double **con valores entre 0 y 1**. Visualizarla con **`imshow()`**: es una fotografía de un texto simulando una toma movida donde la cámara se movió durante la toma.

Haced load K para obtener la máscara K (17x17) usada para degradar la imagen original F (desconocida). Para verla podéis hacer: **`imagesc(K);colormap(gray)`**. El trazo blanco indicaría el movimiento de la cámara mientras se tomó la foto.

Vamos a escribir un algoritmo que parta de la imagen degradada y de la máscara K y trate de recuperar la imagen para poder leer el texto. Esencialmente se trata de recuperar una imagen F tal que al filtrarla con la máscara K, **`imfilter(F,K,'sym')`**, obtengamos algo lo más parecido posible a la imagen degradada de partida **`G`**.

Descripción del algoritmo:

- Lo primero que hay que hacer es crear una variante K_2 de la máscara original, que se usara durante el algoritmo. Es simplemente una versión reflejada en la horizontal y en la vertical de la máscara original: `K2=fliplr(flipud(K);`

- También necesitamos una hipótesis inicial para la imagen F a recuperar. Una hipótesis razonable es empezar con F=G, ya que todo lo que sabemos de F es algo parecido a la imagen degradada G.

- Luego entramos en un bucle donde, en cada iteración:

  - Usando **`imfilter()`** con la opción `'sym'` filtramos nuestra hipótesis actual F con la mascara K. El resultado lo guardamos en G2.

  - Calculamos el cociente (punto a punto) de G (resultado deseado) dividido por (G2+0.001) (el resultado obtenido), obteniendo una matriz Q de las mismas dimensiones que G y G2. Lo de sumar 0.001 a G2 es para evitar hacer divisiones por 0, en caso de que G2 se anule en algún punto.

  - Usando imfilter() con la misma opción `'sym'` de antes filtramos el cociente Q pero ahora usando la máscara K_2 modificada. Guardar el resultado en Q.

  - Finalmente actualizamos la hipótesis F multiplicándola punto a punto por Q. Tras actualizar F forzar a que sus valores siguen estando en el intervalo [0,1], poniendo a 0 los valores <0 y a 1 los que sean >1.

  - Calculad dF = (F_{new} - F_{old}), la diferencia entre la imagen F antes y después de ser actualizada. Guardad en cada paso la desviación standard σ de dF calculada con std2 para monitorizar lo grandes que son las modificaciones que va haciendo el algoritmo en cada paso (recordad que los valores de la imagen de partida están entre 0 y 1).

El bucle puede continuarse por un número fijo de iteraciones o poner una condición para parar cuando las modificaciones introducidas por el algoritmo (dF) se hagan muy pequeñas. En nuestro caso usaremos 400 iteraciones. Adjuntad el código del algoritmo. Tras finalizar el bucle adjuntad la gráfica de la evolución de la desviación de dF (usando escala logarítmica en el eje Y) en función del número de iteración. Adjuntad la imagen final resultante. ¿Cuál es la última frase del texto?

Un problema de estos algoritmos es que es necesario conocer de forma bastante exacta la máscara K que modela la degradación de la imagen. Probad a aplicar el algoritmo de recuperación con la máscara `K=fspecial("gauss",17,4);` también de tamaño 17x17 pero que corresponde a un filtrado gaussiano. El algoritmo converge y la imagen final parece hacerse más nítida, pero el texto final parece un alfabeto inventado. Adjuntar captura de la imagen final y el gráfico de la desviación σ de dF en función del nº de iteración.