Proyecto: Imágenes HDR (High Dynamic Range)

1) Lectura de imágenes y extracción de datos a usar

Las imágenes belg_1.jpg, belg_2.jpg, … son un conjunto muy usado para probar los algoritmos de HDR. Corresponden a P=9 tomas en color con los siguientes tiempos de exposición: 1/1000, 1/500, 1/250, 1/125, 1/60, 1/30, 1/15, 1/8 y 1/4 (seg).

Vimos en clase que la parte fundamental del algoritmo es determinar los valores de la función $\log_2(E)=g(Z)$. En principio habría una función g() distinta para cada plano de color, pero en la práctica las tres salen muy similares por lo que por simplicidad calcularemos una sola función g(Z) (a partir de los datos del plano de color verde).

Hacer un bucle para leer las nueve imágenes, convertirlas a double (manteniendo sus valores entre 0 y 255) y quedaros con el 2º plano (verde), juntándolas en una matriz 3D de dimensiones alto x ancho x 9 (nº de tomas) llamada hdr_data. Cread también un vector T con los tiempos (en segundos) de las P exposiciones.

Llamad a la función muestra_HDR(hdr_data,T) en la que podéis pinchar en distintos puntos de la imagen y ver las curvas obtenidas al graficar el valor de los píxeles en las 9 tomas. Estas curvas serán crecientes ya que a mayor exposición recibida los píxeles tomarán valores más altos. Adjuntad una captura de la figura tras pinchar en 10 o 15 puntos **variados** (claros, medios, oscuros,...). Podéis pinchar en cualquiera de las 9 imágenes, usando el botón derecho para terminar. Con la información de la gráfica de la derecha es con la que trabajaremos para obtener primero los valores de la función g(Z) y a partir de ellos, la imagen HDR.

Extracción de datos en N puntos de las P tomas

Ahora escogeremos N puntos de la imagen de los que guardaremos sus valores en las P tomas (extraídos del conjunto hdr_data creado antes). Para ello escribiremos la función: `function Zdata=extraer_datos(hdr_data,N)` que muestree las P tomas del canal verde (hdr_data) en N puntos y devuelva los valores en una matriz Zdata (tamaño P x N). Escogeremos los puntos aleatoriamente, aunque solo se aceptaran sus datos si cumplen una condición. El proceso completo será:

1) Escoger unas coordenadas (i,j) aleatorias dentro de la imagen en el rango (1,alto) x (1,ancho) : `i=floor(rand*(alto))+1; j=floor(rand*(ancho))+1;`

2) Extraer de hdr_data los datos de las P tomas en la posición (i,j). Aceptaremos únicamente aquellos puntos cuyos valores en las sucesivas exposiciones sean crecientes (como debería ser, ya que en cada toma aumentamos el tiempo de exposición). Una forma sencilla de comprobarlo es verificar que la diferencia entre los valores consecutivos (diff) sea siempre mayor o igual a 0.

3) Si se cumple la condición anterior guardamos los P valores en la 1ª columna de Zdata.

Repetir los pasos anteriores hasta tener rellenas las N columnas de Zdata. Antes de volver, ilustrar la posición de los puntos escogidos, superponiendo sobre una de las tomas (p.e. la 5ª) los puntos escogidos como puntos verdes (modificador 'g.').
Adjuntad la imagen obtenida para N=5000 puntos y el código de vuestra función.

2) Resolución de las ecuaciones para obtener la función g(Z)

El núcleo del algoritmo es la determinación de la función g(Z), la relación entre el valor del píxel Z registrado por la cámara y el logaritmo de la exposición E recibida en ese píxel durante esa toma:

$$\log_2(E) = g(Z)$$

Como la función g(Z) se evalúa en los valores de los píxeles (0,1,…, 255) nos basta conocerla en dichas posiciones, por lo que nuestras incógnitas serán (g_0, g_1, …, g_{255}) un vector con los 256 valores de g() en los 256 posibles valores de los píxeles.

Para hallar los valores de g hay que resolver un sistema lineal H·g ~ b de carácter sobredeterminado (un ajuste), con muchas más ecuaciones que incógnitas. En estos sistemas el ancho de H es el número de incógnitas, en este caso los 256 valores de g(Z). El número de filas de H (y de b) corresponde al número de ecuaciones Neq.

Cada punto suministra (P-1) ecuaciones, lo que da un total de N·(P-1) ecuaciones procedentes de los datos de la imagen. Además de éstas, impondremos otras 255 ecuaciones adicionales (siempre las mismas) tratando de asegurar que g(Z) sea creciente y fijando uno de estos valores (típicamente g_{128}) a 0.

Como típicamente usaremos miles de puntos (N) y tenemos P=9 tomas el número de ecuaciones será fácilmente del orden de decenas o centenares de miles. Afortunadamente en cada ecuación solo aparecen 2 (en la mayoría) o 3 incógnitas, por los que los valores de la matriz H van a ser casi todos igual a 0. En estos casos es un desperdicio reservar una matriz con millones de "casillas" cuando más del 99% de dichas entradas serán ceros. Para evitarlo se usan matrices de tipo "sparse" ("poco pobladas", también llamadas matrices "vacías"). Lo que se hace es guardar sólo aquellos elementos no nulos de la matriz, asumiendo que cualquier elemento no explícitamente especificado es cero.

Por ejemplo, **`H=[1 0 0 3; 0 0 0 -1; 0 2 0 0];`** sería la matriz $H=\begin{pmatrix} 1 & 0 & 0 & 3 \\ 0 & 0 & 0 & -1 \\ 0 & 2 & 0 & 0 \end{pmatrix}$.

Esta matriz también puede crearse como:

```
i = [1  1  2  3];
j = [1  4  4  2];
v = [1  3 -1  2];
H2 = sparse(i,j,v,3,4)
```

Si hacéis (H-H2) os sale una matriz de 0's, ya que H y H2 son la "misma" matriz.

Para crear estas matrices de tipo "sparse" creamos previamente dos vectores con la colección de coordenadas (filas i y columnas j) de los elementos no nulos y otro vector con los valores (v) a poner en dichas casillas. En el caso anterior, notad que en las {i} el 1 está repetido (hay dos elementos no nulos en la 1ª fila) y en las {j} el 3 no aparece (la 3ª columna de H es toda ceros). Luego llamamos a la función sparse() para crear la matriz, dándole también las dimensiones (3x4) de la matriz.

Nuestro problema es ideal para usar estas matrices, ya que en cada fila de H solo hay un par de coeficientes no nulos. Se trata de escribir una función que reciba la matriz Zdata (P x N) con los valores de las P tomas en N puntos y el vector T con los P tiempos de exposición. La función creará la matriz H y vector b, y su salida será un vector g (256x1) los valores de la función g(Z) en los 256 posibles valores de los píxeles, la solución del sistema H·g ~b: `function g = solve_G(Zdata,T)`

Veamos primero los pasos para crear la matriz H y el vector b del sistema:

1) Determinar el número de ecuaciones Neq a partir del nº de tomas P y píxeles usados N (las dimensiones de Zdata).

2) Cread un vector b (inicializado con 0's) de dimensiones (Neq x 1). Al contrario que la matriz H, b no será de tipo "sparse" ya que la mayoría de sus entradas no serán nulas.

3) Para crear una matriz sparse hay que rellenar previamente dos vectores i, j con las coordenadas de las entradas no nulas y un tercer vector v con sus valores.

   En este caso podemos determinar a-priori el número de elementos no nulos (la dimensión de los vectores i, j y v) sabiendo que cada ecuación procedente de los datos de las tomas solo tiene dos entradas no nulas. Por el contrario, las 254 ecuaciones impuestas para "regularizar" g(Z) tienen 3 elementos no nulos (en ellas aparecen g_{k-1}, g_k y g_{k+1}). Finalmente la ecuación imponiendo que $g_{128}=0$ solo involucra a una incógnita. Para N=10000 puntos, determinar el número de entradas no nulas de la matriz H y qué tanto por ciento suponen. Reservar espacio para los correspondientes vectores i, j y v.

4) Las primeras ecuaciones vienen de los datos de los píxeles, por lo que se trata de barrer todos los píxeles desde 1 hasta N. Para cada uno de ellos extraemos los P valores {Z} de las tomas y de entre ellos hallaremos el valor más cercano a 128 (Zref). Restando la ecuación con dicho valor a las demás, obtenemos las siguientes (P-1) ecuaciones:

$$g(Z_1) - g(Z_{ref}) = \log_2(T_1 / T_{ref})$$
$$g(Z_2) - g(Z_{ref}) = \log_2(T_2 / T_{ref})$$
$$\ldots$$
$$g(Z_M) - g(Z_{ref}) = \log_2(T_M / T_{ref})$$

Explicaremos ahora cómo ir añadiendo estas ecuaciones a nuestro sistema.

Cada una de estas ecuaciones involucra únicamente a 2 incógnitas, por lo que añade 2 entradas a los vectores i, j y v. En los dos casos el valor de la i es el mismo: la ecuación en la que nos encontremos (podéis llevar un "contador" de ecuaciones). Los valores de j serán las incógnitas involucradas en la ecuación, en este caso $1+Z_k$ y $1+Z_{ref}$. Lo de añadir +1 a los valores de los píxeles es porque los valores de Z (píxeles) irán de 0 a 255 y tendremos que sumarles uno para usarlos como índices de MATLAB (que empiezan en 1). En cuanto a los valores añadidos a v serán 1 y -1, que son los coeficientes que acompañan a g_k y g_{ref} en la ecuación.

En el caso de b basta guardar el lado derecho de la ecuación, $\log_2(T_k/T_{ref})$, en la fila correspondiente a la ecuación en la que nos encontremos.

Tras rellenar las dos entradas de i, j y v, y la correspondiente componente de b, avanzaremos el contador del nº de ecuación y procederemos a añadir los datos de la siguiente. Recordad que por cada punto (P tomas), añadimos solo (P-1) ecuaciones, ya que una de ellas se ha restado de todas las demás.

Tras añadir las (P-1) ecuaciones procedentes del primer píxel extraemos los datos del siguiente píxel y repetimos el proceso. Al terminar con los N píxeles nuestro contador de ecuaciones debe haber alcanzado el valor de N·(P-1).

5) Tras terminar con las ecuaciones de los N datos de la imagen, debemos añadir las ecuaciones que tratan de imponer que g sea creciente y fijar uno de sus valores. Dichas ecuaciones son:

$$-g_0+2g_1-g_2 \approx 0$$
$$-g_1+2g_2-g_3 \approx 0$$
$$\dots$$
$$g_{128}=0$$

Fijaros que para estas ecuaciones no hace falta modificar el vector b, ya que todas están igualadas a 0 (y el vector b se inicializó con 0's).

En cuanto a H, cada una de las primeras ecuaciones añade 3 valores a i, j y v, ya que involucran 3 incógnitas. Los valores de v añadidos serán siempre -1, 2, 1. Los valores de i serán de nuevo el nº de ecuación en la que nos encontremos. Por último, los valores de j serán los índices de las incógnitas involucradas: (1,2,3) para la 1ª ecuación, (2,3,4) para la 2ª, y así sucesivamente hasta llegar a (254,255,256) para la 254ª.

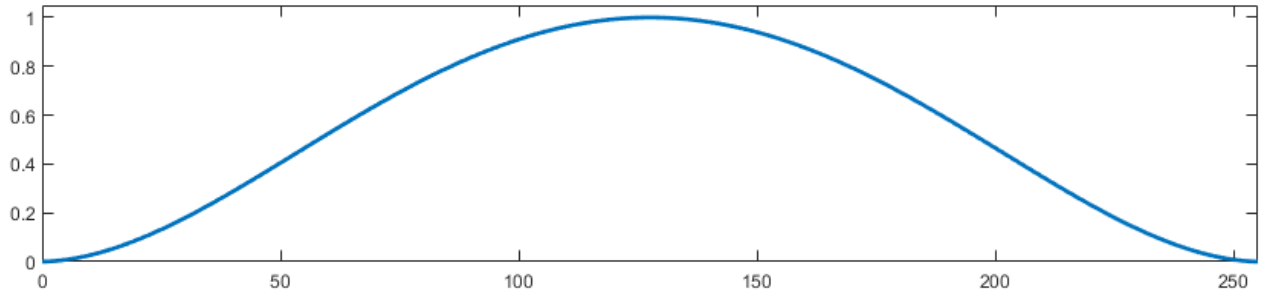
Finalmente, para incluir la última ecuación, añadiremos fila=nº de ecuación, y columna =129 (la posición en el vector de incógnitas de g_{128}) a los vectores (i, j) y un 1 al vector de valores v (el coeficiente de g_{128} en la ecuación).

6) Una vez que hemos terminado de crear los vectores i, j y v, creamos la matriz H "sparse" del sistema como: H = sparse(i,j,v,Neq,256) correspondiente a un sistema con Neq ecuaciones y 256 (g's) incógnitas.

At terminar haced whos H para comprobar que sus dimensiones son las correctas. Una vez creada, podemos hacer con H casi todas las operaciones que podríamos hacer con cualquier otra matriz en Matlab. Por ejemplo, haciendo sum(H(:,10)~=0) podemos ver cuántas veces aparece la incógnita g_9 en las ecuaciones. Cread la matriz H usando N=5000 puntos y adjuntad un plot de sum(H~=0). ¿Qué representan los valores de esta gráfica? A partir de la gráfica, indicad el número de veces que aparecen g_{50} y g_{200} en vuestras ecuaciones.

Uso de pesos: en la práctica es importante ponderar estas ecuaciones en función de su importancia o fiabilidad. Esto se hará al procesar cada ecuación, multiplicando los coeficientes indicados antes (tanto en los valores v de la matriz H como en b) por el peso W determinado para cada ecuación.

Como los píxeles con valores cerca de 0 o de 255 es posible que estén saturados, usaremos un vector de pesos w con un valor máximo 1 cerca del 128 y bajando hasta casi 0 para los valores de los píxeles cerca de 0 o 255 (que son poco fiables).



Una posible elección para estos pesos se ilustra en la figura adjunta, y se obtendría como:

```
M = 256; t=(1:M)'/(M+1); w=(t.*(1-t)).^2; w=w/max(w);
```

Tras definir el vector w, a cada ecuación procedente de los datos le asignaremos el peso $W=\sqrt{w(Z_k)\cdot w(Z_{ref})}$, la media geométrica de la fiabilidad de los dos valores que aparecen en la ecuación. De nuevo tened en cuenta la diferencia entre valores de los píxeles (0...255) y los índices (1...256) al extraer los valores de la tabla w. Recordad que este peso debe aplicarse tanto a los valores v como al vector b.

Indicad el peso que usaríais para la ecuación $g_{50} - g_{120} = ...$

Las 255 ecuaciones finales sobre los g's son "teóricas" (esto es, no se basan en datos de los imágenes) por lo que no hace falta aplicarles estos pesos w. Lo que si haremos es multiplicar estas ecuaciones por un peso λ constante que nos permitirá regular la importancia de las 255 ecuaciones "teóricas" frente a las $N\cdot(P-1)$ procedentes de los datos. En principio usaremos $\lambda=1$, dando similar importancia a ambos tipos de ecuaciones. Adjuntad el código para crear H y b incorporando pesos.

Resolución del sistema para obtener g(Z):

Tras construir H y b el ajuste H·g ~ b se resuelve como g=H\b, igual que se hace en Matlab al trabajar con matrices no "sparse". Tras completar de esta forma la función solve_G(), la usaremos en un script donde juntaremos todos los pasos descritos hasta ahora:

a) Leer las imagenes y crear hdr_data (N x M x P) + vector T de tiempos.

b) Definir el número de puntos N a usar y usar **`extraer_datos()`** para guardar la información de N píxeles en las P tomas en una matriz Zdata (P x N).

c) Usar solve_G() para hallar los valores de g (vector columna de tamaño 256).

Usando N=10000 y λ=1, resolver y adjuntad una gráfica de la función g(z) en función del valor de píxel (de 0 a 255). ¿Es la función g(Z) estrictamente creciente?

Resolver para N=100 puntos y adjuntad la nueva curva g(Z). ¿Es ahora más visible el problema? ¿Qué podríamos hacer en este último caso para intentar arreglarlo?

Podemos ver el efecto de eliminar completamente las restricciones impuestas a g(Z) haciendo λ=0. Esto puede funcionar si se usan muchos puntos, ya que los errores de los datos tenderán a reducirse y no será tan crítico el uso de las ecuaciones de "regularización". Por el contrario, si el número de puntos es bajo los resultados pueden ser muy incorrectos. Superponer en una gráfica las curvas g(Z) obtenidas para N=100 y 100000 puntos con λ=0. Adjuntad la gráfica y explicar por qué para N=100, ciertos valores de g(Z) son igual a 0 y completamente erróneos. Puede ser ilustrativo ver la gráfica que hicimos antes a partir de la matriz H con las veces que aparece cada valor g(Z) en las ecuaciones para este caso.

3) Estimar la radiancia R de la escena

En este apartado usaremos los datos de g(Z) obtenidos con N=10000 puntos y un peso λ=1. Una vez conocidos los valores de g(Z), podemos obtener la exposición E recibida en un píxel a partir de su valor Z ya que log2(E)=g(Z). A partir de E, sabiendo que E=R·T (tiempo de exposición en la toma) es sencillo estimar la radiancia de la escena completa para generar así la imagen HDR. Para cada píxel, dados sus valores en las P tomas $\{Z_1, Z_2,..., Z_P\}$ y los tiempos de exposición de esas tomas $\{T_1, T_2,..., T_P\}$ podemos despejar el logaritmo en base 2 de la radiancia de la escena R, obteniendo las siguientes ecuaciones:

$$\log_2 R \approx g_{Z1} - \log_2(T_1)$$
$$\log_2 R \approx g_{Z2} - \log_2(T_2)$$
$$...$$
$$\log_2 R \approx g_{Zm} - \log(T_M)$$

En principio, al ser la radiancia R de la escena común en todas las tomas, podríamos estimarla con cualquiera de esas ecuaciones, ya que conocemos los valores de g's y los tiempos de exposición. Usando los datos de hdr_data de la primera exposición, usar la 1ª ecuación para obtener el log2 de la radiancia de la escena (para el canal verde). Adjuntad el código usado (puede hacerse con una sola línea de MATLAB).

Para visualizar esta radiancia podemos un pseudo-color, donde se asocia un color distinto a los píxeles de la imagen en función del logaritmo de la luminosidad, como si se fuese un mapa de "temperaturas". Visualizar la imagen obtenida con imagesc() usando "hot" como paleta de color. Superponer en la figura una barra de color ilustrando el rango de valores. Adjuntad la imagen resultante. Al ser un logaritmo en base 2, el rango de valores corresponde al rango de "stops" fotográficos que cubre la escena. Indicad el rango dinámico de la escena en stops.

El problema de usar una sola toma para estimar log2(R) son los píxeles con valores cercanos a 0 o 255. En estos casos, al estar cerca de la saturación, la estimación de log2(E) a partir de g(Z) no es fiable, lo que da lugar a un nivel alto de ruido en la estimación, que es claramente visible en la imagen anterior. Para evitar el problema promediaremos los valores $\log_2(R)$ obtenidos en cada ecuación para estimar la radiancia definitiva. Al hacer esta media daremos más importancia a los resultados procedentes de valores de Z cercanos a 128 (más alejados de 0 o 255 y por lo tanto más fiables), usando el mismo vector de pesos w definido antes. Todo esto se implementará en una función `function log2R = get_log2R(hdr_data,g,T)`

La función recibe los datos de las 9 tomas para el canal verde (hdr_data), el vector g (256x1) con los valores de g(Z) estimados y el vector de tiempos T. A partir de estos datos la función devolverá una imagen (ancho x alto) con el logaritmo (en base 2) de la radiancia de la escena. El proceso detallado sería:

1. Reservar una matriz log2R (alto x ancho) para guardar la imagen final.

2. Barrer todos los píxeles y para cada uno de ellos:

   - Sacar los datos $\{Z_1,Z_2,\ldots, Z_P\}$ del píxel en las P tomas.
   - Con la información del vector g y del vector de tiempos T, calcular los P valores del lado derecho de las ecuaciones: $g(Z_k) - \log2(T_k)$
   - Calcular los pesos $\{W_1,W_2,\ldots, W_P\}$ correspondientes a $\{Z_1,Z_2,\ldots, Z_P\}$ a partir del vector w de pesos. Normalizar el vector W obtenido de forma que la suma de sus valores sea 1.
   - Hacer la media ponderada de los P valores del lado derecho usando los pesos W. El resultado será la estimación de $\log_2(R)$ para ese píxel. Guardarlo en la correspondiente posición de log2R. La media ponderada de P valores con pesos W_k es:

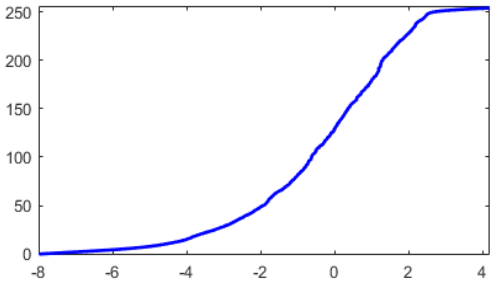
$$\sum_{k=1}^{P} V_k \cdot W_k$$

Adjuntad vuestro código de get_log2R().

Añadid get_log2R() al script anterior para completar el proceso y ejecutadlo usando de la información en N=10000 puntos (con un peso λ=1 para las restricciones).

Adjuntad la imagen resultante. El resultado debe tener mucho menos ruido que la obtenida antes usando una sola toma. ¿Cuál es ahora el rango dinámico de la escena en stops? ¿Es menor o mayor que antes? Justificar.

Una vez que disponemos de los valores de log2R podemos calcular la exposición E en cada toma y hacer una gráfica que muestre en el eje X el log2 de la exposición recibida en un píxel y en el eje Y el valor del píxel registrado por la cámara como consecuencia de esa exposición. Esta gráfica corresponde a la inversa de la función g(Z) que hemos calculada anteriormente y debe tener una forma similar a la de la figura adjunta.



Para no "recargar" el gráfico lo haremos sólo con los datos extraídos de una sola fila de píxeles, la correspondiente a la mitad de la imagen (fila=alto/2). Empezar con la primera toma y estimar log2(E) para los píxeles de esa fila sabiendo que:

$$E = R \cdot T_1 \quad \text{--> } \log2(E) = \log2R + \log2(T_1),$$

donde T_1 es el tiempo de exposición de la toma. En cuanto a (log2R), es común a todas las tomas (ya que radiancia es una propiedad de la escena) y lo tenemos ya calculado (basta extraer de log2R los datos de la fila adecuada).

Hacer un plot (usad el modificador '.') con los valores estimados para log2E en el eje X y los valores de los píxeles de esa fila en el canal verde de la toma en el eje Y.

Repetir con las siguientes tomas superponiendo todos los plot en la misma gráfica (usad colores diferentes para cada toma). La solución encontrada antes para g(Z) es simplemente un ajuste a los puntos representados en esta gráfica. Podéis verlo superponiendo el ajuste obtenido g sobre la gráfica anterior. Adjuntad la gráfica resultante (datos de la fila alto/2 de todas las tomas + solución g superpuesta).

Visualización de la imagen HDR:

En el apartado anterior se calculó log2R para un solo canal, el verde, ya que se trabajó con hdr_data, donde previamente se había guardado el canal verde de todas las exposiciones. En una imagen en color por cada píxel habrá que calcular 3 valores de log2R (uno por cada canal), cada uno obtenido a partir del valor del píxel de ese canal. Por simplicidad usaremos la misma tabla g(Z) (la calculada a partir de los datos del canal verde) para los tres canales. La forma más sencilla de hacerlo con las funciones disponibles es volver a leer todas las imágenes y guardar ahora en hdr_data los datos del canal rojo (en vez del verde).

Llamando luego a **`get_log2R(hdr_data,g,T)`** tendremos el plano de log2R del canal rojo (notad que no recalculamos los valores de g). Repitiendo el proceso para el canal azul completaríamos los tres planos de log2R que juntaremos con cat() para componer una sola imagen (alto x ancho x 3).

Una vez estimado el logaritmo de la radiancia log2(R) para los 3 planos, basta hacer hdr=2.^log2R para obtener la radiancia de la escena, esto es, la imagen HDR. El rango dinámico de una escena es el cociente entre los valores máximo y mínimo de su radiancia. Indicad máximo y mínimo de la radiancia obtenida y rango dinámico.
Vamos a tratar de visualizar hdr directamente con imshow(). Como imshow espera imágenes con valores entre 0 y 1, re-escalar previamente la imagen usando el valor máximo M y mínimo m hallados antes para que su mínimo sea 0 y su máximo 1. Adjuntad código usado para re-escalar e imagen resultante. ¿Cuál es el problema?

El mismo problema tendremos si queremos guardar la imagen HDR en un fichero. Si tratamos de usar imwrite() como si fuera una imagen "normal", durante el proceso la información se vuelve a "colapsar" a 1 byte por pixel y por canal, perdiendo todo nuestro trabajo. Para guardar estas imágenes usad **`hdrwrite(hdr,'im.hdr');`** que guarda la imagen en un formato (extensión .hdr) desarrollado especialmente para este tipo de imágenes HDR.

Para visualizar la imagen HDR con las limitaciones de nuestra pantalla tendremos que aplicarle un algoritmo adicional. Estos algoritmos ("tone-mapping") tratan de ajustar el rango de valores de nuestra imagen HDR al (mucho más limitado) rango de la pantalla, intentando mantener visible el detalle tanto en las zonas claras como en las oscuras (no como se hizo antes con imshow). En ellos se suelen usar técnicas de ajuste local de contraste como la que vimos en un laboratorio anterior.

Afortunadamente, si no queremos trabajar más, MATLAB tiene una función que hace exactamente eso. Se llama tonemap() y recibe una imagen HDR (float) devolviendo una imagen uint8 (en el rango 0-255) lista para ser visualizada. La función admite parámetros opcionales para controlar la saturación y otras características de la imagen de salida. Probad con: **`rgb=tonemap(hdr,'AdjustSaturation',3);`** y adjuntad la imagen resultante usando imshow().

Adjuntad dentro de esta entrega (en un .zip o similar) un script `crear_hdr.m` que genere la imagen HDR y la muestre. Incluid en vuestro script todas las funciones (get_log2R, extraer_datos, solve_G, ...) necesarias para que funcione (no hace falta incluir las imágenes de partida).

4) Fusión de exposiciones usando la pirámide laplaciana

El algoritmo usado en los apartados anteriores se basaba en plantear la relación existente entre la radiancia de la escena y la exposiciones capturadas por el sensor para los distintos tiempos de exposición. Su ventaja es que podía recuperar (salvo un factor) la información sobre la radiancia real de la escena. Sin embargo, si el objetivo es simplemente mezclar fotos con distinta exposición, combinando en una imagen las partes que mejor se ven de unas u otras hay métodos más sencillos. En este apartado implementaremos un método de "fusión de exposiciones", bastante más sencillo de implementar.
Está basado en la pirámide laplaciana, cuya aplicación en la fusión de imágenes se presentó en un tema anterior. Básicamente, el método consiste en calcular la pirámide laplaciana de todas las imágenes de partida para luego combinarlas en una única pirámide. Invirtiendo la nueva pirámide combinada se obtiene la imagen HDR final.

En este caso partiremos de tres imágenes (exp_1, exp_2 y exp_3) con diferentes exposiciones (aquí no necesitamos conocer sus tiempos de exposición). El algoritmo tal como se describe opera con imágenes de tipo double con un **rango entre 0 y 1**.

Pirámide Laplaciana: escribid una función para calcular la pirámide laplaciana que presentamos en el Tema 4. Usaremos el template:

```
function p=lap(im,N)
    p=cell(1,N);
```

La función recibe la imagen im (de tipo double en el rango [0,1]) y el número N de "niveles" de la pirámide: N niveles significa que la pirámide de salida (p) tiene (N-1) niveles de detalle (a diferentes escalas) junto con una versión muy reducida de la imagen original (nivel N).

Los resultados de los diferentes niveles se guardan en el argumento de salida p, que será array de celdas (usamos un array de celdas porque nos permite guardar objetos de diferentes dimensiones en sus casillas). Para acceder a los elementos de este tipo de arrays se usan llaves en lugar de paréntesis: p{1}, p{2},... en lugar de p(1), p(2), ... Si el contenido de p{2} es p.e. una matriz podemos direccionar sus componentes haciendo p{2}(3,4).

Haced primero un bucle desde k=1 a N-1 para calcular los N-1 niveles de detalle. En cada paso:

1. Usando imresize, crear una versión reducida a la mitad de la imagen im.

2. Volver a aplicad imresize() a la imagen reducida, pero ahora ampliándola por un factor 2. La imagen resultante (im2) habrá perdido el detalle de la original.

3. Restar a la imagen original (im) la imagen sin detalle (im2),obteniendo así el detalle correspondiente a ese nivel, que guardararemos en p{k}.

4. Finalmente, guardad en im la versión reducida para que en el siguiente nivel se repita el proceso partiendo de una imagen de tamaño mitad.

Tras terminar el bucle, guardad la última versión de la imagen reducida en el último nivel de la pirámide, p{N}. Adjuntar código de vuestra función lap.m

Probadla con la imagen de "exp_1.jpg" y N=5. Verificar que los 5 niveles de la pirámide tienen tamaños: **{960×1472}, {480×736}, {240×368}, {120×184}, {60x92}**

Visualizar los dos últimos niveles (4º y 5º) de la pirámide con imagesc() y adjuntad una captura de los resultados.

Combinación de las pirámides laplacianas: tras calcular la pirámide laplaciana de las 3 imágenes (con N=5 niveles de profundidad) las combinaremos en una única pirámide siguiendo estas reglas:

- Para el último nivel (N=5) de la pirámide (versión pequeña de las imágenes) promediaremos los correspondientes niveles de las 3 pirámides que tenemos.

- Los otros niveles (1:N-1) guardan el detalle de las imágenes. La idea de este algoritmo es que las zonas muy oscuras o demasiado claras tendrán menos detalle que las correctamente expuestas. Se trata pues de conservar en cada nivel la información de aquella imagen con el máximo detalle. Dentro de cada escala, la imagen con más detalle tendrá valores más altos en el nivel de la pirámide correspondiente. Como nuestras imágenes son en color, para decidir cuál es el detalle más alto se usará el criterio de la norma del vector rgb.

- Barrer los niveles de 1 a N-1 y dentro de cada nivel barrer todos los píxeles. Para cada píxel determinar cuál de las 3 pirámides tiene un mayor detalle (comparando la norma del vector rgb del detalle para las 3 imágenes en ese nivel). Una vez determinado el píxel con más detalle, usadlo para rellenar el correspondiente píxel de ese nivel en la pirámide combinada.

Adjuntar código usado para combinar las 3 pirámides

Inversión de la pirámide mezcla: tras obtener la pirámide combinada (pm) solo nos falta “colapsarla” de vuelta a una imagen para obtener la imagen HDR:

- Inicializaremos la imagen HDR a recuperar con el último nivel HDR=pm{N}

- Hacemos un bucle desde k=N-1 a 1, barriendo el resto de los niveles y en cada paso: 1. Ampliamos HDR por un factor 2 usando imresize().
  2. Sumamos a HDR el detalle de la pirámide de ese nivel.

Al final terminamos con una imagen en HDR del mismo tamaño que las originales.

Como comprobación, aplicad el proceso a la pirámide de la primera imagen (sin modificar), restad la imagen original y la recuperada y calcular su valor absoluto. Adjuntad el valor máximo de la diferencia entre ambas, que debería ser muy bajo. Adjuntar código usado para invertir la pirámide laplaciana.

Retoques finales: un problema de la imagen HDR recuperada es que, al ser la pirámide invertida una combinación de varias imágenes, no está garantizado que sus valores se mantengan entre 0 y 1. Lo primero es calcular sus valores mínimo v0 y máximo v1 y usarlos para reescalar la imagen final resultante entre 0 y 1.

En la imagen final hemos combinado el detalle de las diferentes imágenes, tanto en las zonas de sombra como en los claros. Como hemos visto en el apartado anterior, este detalle puede perderse de nuevo al tratar de visualizar la escena. Una forma de evitarlo es hacer una ecualización local del histograma. Una ecualización aumenta el contraste de una imagen, y el hecho de ser local hace que lo resalte tanto en las zonas claras como oscuras (algo similar a la técnica de resalte local del contraste implementada en el LAB4). En MATLAB podemos usar la función:

```
I=adapthisteq(I,'ClipLimit',clip);
```

que implementa el algoritmo sobre una imagen 2D. El parámetro clip es la fracción de píxeles saturados (por debajo de 0 o por encima de 1) admitidos en la imagen final. Un valor adecuado puede ser del orden del 1% o 0.01. Cuanto más alto sea dicho porcentaje más "dramático" (y menos realista) será el resultado.

Esta ecualización se suele aplicar solo a la luminancia (versión "BW" de la imagen). Para ello, convertir la imagen obtenida (ya re-escalada entre 0 y 1) al espacio HSV (Hue, Saturación, Value) y aplicar la ecualización indicada al canal de luminancia V.

Ya que estamos en el espacio HSV aumentad también un poco la saturación, aplicando la función $T(x)=x^{0.75}$ al canal de saturación. Adjuntad el código usado para re-escalar la imagen, aplicar la ecualización adaptativa y el aumento de saturación.

Adjuntad imagen final resultante.

Adjuntad dentro de la entrega un 2º script fusion.m que implemente los pasos descritos en este algoritmo y muestre la imagen resultante. El script debe contener todas las funciones auxiliares para que funcione de forma autónoma (no hace falta incluir las imágenes de partida).

Volver a aplicad este algoritmo usando ahora como imágenes de entrada tres de las imágenes del apartado anterior, en particular: belg_3.jpg, belg_5.jpg y belg_7.jpg. Adjuntad la imagen resultante.