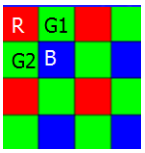
PROYECTO 3 (el sensor y el revelado digital)

Repaso propiedades al combinar ruidos aleatorios: La función randn(N) crea una matriz de tamaño N x N con muestras de un ruido aleatorio con una distribución normal o gaussiana de media m=0 y desviación standard σ=1. Crear una matriz 1000 x 1000 con n1=randn(1000) y verificar que su media (mean2) y desviación σ (std2) son aproximadamente 0 y 1. Crear una segunda imagen (del mismo tamaño) con n2=randn(1000). Verificar que su desviación σ es ~1. ¿Cuál es la desviación σ de la suma de ambos ruidos (n1+n2)? ¿Y de su resta (n1-n2)? ¿Cuál sería la regla para la desviación σ de una suma/resta de ruidos n1/n2 en función de σ1 y σ2?

Dada una imagen de media m_1 y $\sigma = s_1$ para cambiar su media a m_2 y su desviación standard a s_2 haríamos **`new=(im-m1)/s1; new=new*s2+m2;`**. El primer paso nos da una imagen de media m=0 (al restar m1) y σ=1 (al dividir por s1). Luego, al multiplicar por s2 tenemos una imagen con σ=s2 y al sumar m2, nos aseguramos que la media de la imagen sea m2. A partir de n1, cread n3 de forma que su media sea m=5 y su desviación σ=2 y verificarlo. ¿Cuál esperamos que sea la desviación σ de la suma (n2+n3)? Comprobadlo. Calculad la desviación σ de la suma (n1+n3). ¿Se obtiene el mismo resultado? ¿Qué puede estar pasando?

1. Manejo imágenes RAW / Estimación del ruido en el Sensor

a) El fichero "black.pgm" es de una foto tomada con el objetivo tapado y exposición muy corta. El archivo se ha extraído de una imagen RAW con el programa dcraw que extrae (sin revelarlos) los datos del sensor. Los valores del fichero son de tipo **uint16** (enteros 16 bits) y corresponden a los datos (0-4095) que el conversor AD (12 bits) ha obtenido del sensor. El CFA (Color Filter Array) que usa esta cámara se muestra en el esquema adjunto.



1. Leer la imagen de black.pgm con imread(). La imagen leída es bidimensional, ya que la información de color está repartida (multiplexada) espacialmente. Extraer los 4 conjuntos de píxeles de cada color y guardarlos en 4 matrices R, G1, G2 y B (considerando los canales G1 y G2 como colores distintos). Usad para ello el indexado de MATLAB. Si raw es la imagen con los datos RAW originales, para extraer p.e. el canal rojo haríamos **`R=raw(1:2:end,1:2:end);`** De esta forma nos quedaríamos con la 1ª, 3ª, … fila de la 1ª, 3ª,…, columnas. Adjuntar el código para extraer los demás canales. Visualizar el histograma del canal G1 usando **`hist(double(G1(:)),500);`** Como el sensor no ha recibido fotones en esta toma lo que vemos es el ruido de lectura de la cámara. Hacer un zoom "horizontal" en el intervalo [115-140] donde se concentran los valores. Adjuntad figura. ¿Qué tipo de distribución parece tener el ruido?

2. Calcular la media y desviación σ (mean2, std2) de cada canal R, G1, G2 y B. Rellenar la tabla de la hoja de respuestas con vuestros resultados.

Estos resultados muestran claramente que la cámara usa un offset al guardar los datos del sensor. ¿Cuál es su valor?

b) En el apartado a) se usó una toma (*blackframe*) en la que el único ruido presente era el ruido de lectura. En este ejercicio vamos a cuantificar otras fuentes de ruido del sensor (shot-noise y el efecto de la diferente sensibilidad de los "sensels"). Los datos de partida son ahora una serie de 11 frames con exposiciones cada vez más largas (a saltos de 1/2 stop), con un tiempo de exposición desde 1/1000 seg (izquierda) a 1/30 segundos (derecha).

Como 11 imágenes RAW ocuparían mucho os voy a dar los datos ya extraídos de las imágenes (del canal verde). Si hacéis **`>>load ruido`** veréis una variable **ruido** que es un array de celdas de tamaño (1x11). Cada celda contiene los valores de exposición (uint16) de una zona de 50x50 píxeles correspondiente a una parte homogénea de una foto. Para acceder al contenido del k-ésimo frame usad **`frame=ruido{k}`**.

En las circunstancias en las que se tomaron estas fotos las 3 fuentes de ruido más importantes eran: ruido de lectura, ruido de tipo *shot-noise* y el "ruido" causado por las inevitables diferencias de sensibilidad entre los "sensels". Con una sola toma no podemos distinguir cada uno de estos ruidos por separado, al haberse sumado en el sensor antes de ser leídos y convertidos a datos RAW. Lo que si podemos estimar es el **nivel de ruido total** (σ) de cada exposición. Como el trozo escogido es una zona homogénea de la foto, si no hubiera ruido todos los valores de la imagen RAW (en ADU's, o unidades del ADC) deberían ser iguales dentro de la misma exposición. La desviación (σ) de dichos valores respecto a su media nos permitirá estimar el nivel total de ruido para esa exposixión. Se trata de:

1. Reservad dos vectores **columna** E y S de tamaño 11 x 1 donde guardaremos los datos obtenidos (exposición y nivel de ruido σ) de cada frame.

2. Haced un bucle barriendo las 11 exposiciones. Extraer la k-ésima exposición y para ese frame calculad:

   a) Su exposición (en unidades ADU) como la media (mean2) de los valores del frame (acordaros de restar previamente el offset/nivel del negro que habéis medido en el ejercicio anterior). Guardar dicho valor en E(k).

   b) Su desviación standard σ (std2) que mide el nivel de ruido total del frame. Guardad el resultado en S(k).

Primero verificaremos que la respuesta del sensor es lineal (la exposición medida es proporcional al tiempo de cada toma) haciendo una gráfica de la exposición E en función de los tiempos T de exposición (milisegundos). Adjuntad la gráfica de (T,E)

Luego, adjuntad la gráfica (semilogx) del nivel de ruido S (eje Y) en función de la exposición E (eje X) en los 11 frames (usad puntos discretos con modificador 'rs').

En cada toma la varianza del ruido total (recordad el ejercicio anterior al hacer la suma de varios ruidos independientes) será la suma de las varianzas de cada uno de los ruidos por separado: $\sigma^2 = S^2 = \sigma_R^2 + \sigma_S^2 + \sigma_W^2$. Con solo una toma sería imposible separar la contribución de cada ruido. Sin embargo, con varias tomas si podremos distinguirlos ya que los diferentes ruidos muestran un comportamiento distinto según cambia la exposición:

- El ruido de lectura σ_R es constante, independiente de los fotones N recibidos.
- El ruido de tipo "shot noise" sigue una estadística de Poisson y corresponde a la raíz cuadrada del nº de fotones recibidos $\sigma_S = \sqrt{N}$.
- El ruido de la no-uniformidad entre píxeles es proporcional a N: $\sigma_W = K \cdot N$

Los datos de σ obtenidos no están en fotones N, sino en ADUs (unidades del ADC). Sin embargo, como la exposición (en ADU's) es E=G·N (ganancia x nº fotones), podemos escribir la dependencia del ruido global S con respecto a la exposición E (todo en unidades de ADU's) como:

$$S^2 = \sigma^2 = \sigma_r^2 + G \cdot E + K^2 \cdot E^2$$

Se ve que la primera fuente de ruido es constante, la segunda es proporcional a la exposición E y la tercera va con E^2. La idea ajustar nuestros datos $S^2=\sigma^2$ usando un polinomio de grado 2 (1, E, E^2):

$$S^2 = \sigma^2 \approx c_1 \cdot 1 + c_2 \cdot E + c_3 \cdot E^2$$

Tras el ajuste, comparando ambas expresiones podemos estimar los parámetros del sensor σ_R , G y K a partir de los coeficientes c_1, c_2 y c_3. Para hacer el ajuste creamos la matriz del sistema sobredeterminado H a partir del vector E con las exposiciones como H=[E^0 E E^2]. El vector b con los datos que queremos ajustar será S^2. Luego se resuelve el sistema H·c=b en el sentido de mínimos cuadrados haciendo c = H\b.

Adjuntar código para calcular el ajuste y los coeficientes c_1, c_2 y c_3 obtenidos
¿Qué valor de σ_R se obtiene para el ruido de lectura?
¿Qué valor tiene la constante K que refleja la no-uniformidad de los receptores?
¿Cuántos fotones admitiría cada elemento de este sensor antes de saturarse?

Cread un vector e=(100:4000) con valores de exposición entre 100 y 4000 (ADU's) y obtener el valor de σ que predice nuestro ajuste:

$$\sigma \approx \sqrt{c_1 + c_2 \cdot e + c_3 \cdot e^2}$$

Luego, usando semilogx como antes hacer una gráfica de σ (eje Y) frente a e (eje X) con una línea azul continua ('b'), superponiéndolo sobre la gráfica anterior de los datos medidos (E,S). Adjuntad la gráfica resultante.

Usando el ajuste podemos también hacer una gráfica del % que supone el ruido (σ) para diferentes valores de la exposición (e). Adjuntad una gráfica del porcentaje de ruido (s/e) (en tanto por ciento) en función de la exposición e.

Esta gráfica es la razón por la que en fotografía digital se recomienda exponer "a la derecha", maximizando la exposición, ya que el nivel relativo del ruido es menor en esa zona. Si luego deseamos una fotografía más oscura, siempre podemos dividir sus valores en la fase de post-proceso (lo que no aumenta el nivel de ruido). Para una exposición ~ 800, ¿qué nivel de ruido tenemos (en %)? ¿Y para una exposición ~ 3200? ¿Por qué el error relativo se dispara para exposiciones muy bajas?

2. Balance de blancos (10%)

El balance de blancos se suele corregir en la cámara sobre la imagen RAW, pero es posible hacerlo también en la fase de post-proceso partir de un JPG ya "revelado".

En este ejemplo, la foto de la izquierda está tomada bajo luz incandescente pero con la cámara usando el balance de blancos ajustado a luz diurna. El resultado es una foto claramente naranja. En la foto de la derecha (fichero 'color.jpg') la cámara estaba en modo Auto White Balance (AWB) que trata de ajustar automáticamente el balance de blancos. El resultado es mejor, pero queda un claro toque de color que el algoritmo automático no ha conseguido eliminar por completo. Para mejorar este resultado seleccionaremos manualmente un punto de la imagen que sepamos que es un tono "neutro" como se haría en un software de post-procesado.

- Cargad la imagen, convertirla a double con im2double() y visualizarla. Usando el cursor de datos elegir un punto sobre el papel y extraer los valores RGB de dicho punto. Indicad coordenadas XY del punto usado y sus valores RGB.

- Si el balance de blancos estuviera correcto los valores R,G,B en ese punto deberían ser iguales ya que el papel es un color neutro. A partir de los valores observados calculad el vector **`div=[R G B]/mean([R G B]).`** Si ahora hacemos **`[R G B]./div`** veréis que los tres valores obtenidos son iguales: el vector div nos da los 3 factores por los que tenemos que dividir cada canal para que el punto elegido se convierta en un gris neutro. Adjuntad valores del vector div.

- Aplicar ahora los factores de compensación hallados para ese punto a la parte derecha de la imagen (columnas 751:1500). Adjuntad la imagen resultante.

Volver a ejecutar el código usando como punto neutro un punto escogido sobre la ficha azul. Indicad las coordenadas usadas y los nuevos valores del vector div. Adjuntad la imagen resultante. ¿Qué tonalidad adquiere ahora la imagen corregida?

3. Revelado completo de una imagen RAW

En el fichero 'raw.pgm' tenéis una imagen (2022x3040) con los datos RAW (uint16) de una fotografía. Estos datos los he obtenido usando el programa **dcraw** con las opciones –D -4 (extraer datos del sensor sin revelarlos) y son la respuesta del ADC para los ~6 millones de "sensels" del sensor. El programa dcraw guarda los datos en un fichero de tipo pgm que se puede leer con la función imread() de MATLAB.

El proceso consta de varias etapas intermedias que iremos mostrando paso a paso. Para visualizarlas, usaremos la función: **`fc_pinta_im(im,'XXXXX')`** que muestra una imagen im (junto con su histograma en una esquina). El 2º argumento etiqueta la figura (usadlo para indicar en qué paso del proceso estáis).

3.1) Escalado de los datos. Leer la imagen y pasarla a double (con double). En este caso no usamos im2double porque nos encargaremos nosotros de reescalar sus valores: el valor del offset (128 en esta cámara) debe ir al 0 y el valor máximo de la imagen (M=max(raw(:))) al 1. Tras escalar la imagen, poner a cero aquellos valores que sean negativos (los que estaban por debajo del nivel del negro u offset). En esta práctica trabajaremos siempre con imágenes en el intervalo [0,1]. Adjuntad (fc_pinta_im) la imagen RAW una vez reescalada. La figura adjunta muestra 4 píxeles contiguos de esta imagen correspondientes a una zona del cielo. Justificad cuáles serían los colores del filtro de Bayer (R,G1,B,G2) asociados a cada uno de los cuatro píxeles.

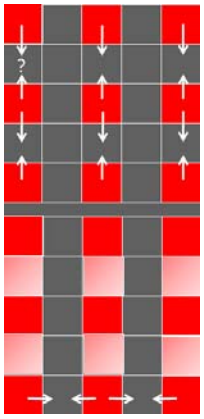
3.2) Demultiplexado: el siguiente paso es demultiplexar los colores muestreados con el filtro de Bayer para obtener los tres canales R, G y B por separado.

Cread 3 imágenes R, G, B (mismo tamaño que la imagen RAW original) inicializadas a cero. Trasladar a cada una de ellas la información del correspondiente canal de la imagen RAW en las mismas posiciones, como se muestra en la figura. Recordando el ejercicio 1, para trasladar los datos del canal rojo de la imagen raw al canal rojo R haríamos:

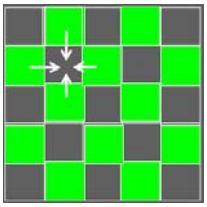
```
R=0*raw;  R(1:2:end,1:2:end) = raw(1:2:end,1:2:end);
```

Hacer lo apropiado para obtener B y G. Notad que los valores desconocidos de cada canal quedan como 0's. Visualizar la imagen RGB obtenida al juntar los tres planos con **`fc_pinta_im(cat(3,R,G,B),'Mosaico').`** La función cat() apilar varias imágenes 2D (en este caso R, G, B) a lo largo de la dimensión indicada por el 1er argumento (en este caso la tercera dimensión) creando así una imagen RGB cuyos planos de color son R, B y G. Adjuntar la imagen. Haced zoom sobre una parte de la imagen con nieve hasta apreciar los píxeles individuales. Como los 3 canales responden al color blanco se apreciará claramente el patrón de Bayer subyacente.

3.3) De-mosaicing: Ahora rellenaremos los valores (que ahora están a 0) de cada plano para los que el sensor no capturó información (la mitad del canal verde y tres cuartas partes del canal rojo y azul). Estos algoritmos se llaman de "demosaicing", porque deshacen el aspecto de mosaico de la imagen anterior. En esta práctica, para recuperar los valores desconocidos, usaremos una sencilla interpolación linear.



Para completar el plano rojo (R) recorrer las columnas donde tenemos información (1ª, 3ª, 5ª,…) y calcular los valores que faltan como la media de los vecinos arriba y abajo (ver fig adjunta). Tras completar este paso quedan unas columnas (2ª, 4ª, …) vacías, que rellenaremos haciendo la media de las columnas de cada lado. Escribid los bucles de forma que no necesitemos nunca un valor de fuera de la matriz R. Eso hará que algunos puntos de los bordes se queden sin rellenar (luego los eliminaremos). Repetir con el canal B, procesando primero las columnas con información (2ª, 4ª, etc.) y promediando luego las columnas vecinas como se ha hecho con el canal rojo.



En el canal verde los píxeles vacíos se calculan como la media de los 4 vecinos (arriba, abajo, izquierda, derecha). De nuevo, no calcular los píxeles en los bordes (primera y última filas/columnas) para los que necesitaríamos conocer el valor de píxeles de fuera de la imagen.

Finalizada la interpolación, quitar el borde a las imágenes R, G y B para eliminar los píxeles que han podido quedarse sin rellenar. Para ello, eliminad la primera y última fila y columna. Por ejemplo, para el canal rojo haríamos: `R=R(2:end-1,2:end-1);`

Volver a visualizar la imagen RGB que obtendríamos tras completar la interpolación de los 3 canales, `cat(3,R,G,B),` con `fc_pinta_im().` Adjuntar la imagen resultante.

3.4) Equilibrado de blancos y cambio al espacio de color sRGB:

Aunque ya no tiene el aspecto de mosaico de antes, la imagen obtenida muestra unos colores poco naturales. La razón principal es que no hemos hecho todavía el balance de blancos (WB). Como se trata de reproducir el proceso de revelado que tiene lugar en una cámara corregiremos automáticamente el color (como cuando se escoge en la cámara la opción de Auto White Balance). Queremos repetir lo que se hizo en el ejercicio 2 pero ahora calculando automáticamente los coeficientes de compensación, sin tener que elegir manualmente un punto neutro en la imagen.

Una opción es suponer que la media de la imagen es un gris neutro, pero eso puede dar malos resultados si una imagen tiene un color predominante (no derivado de la iluminación). Otra opción es hacer la media sólo de los píxeles que cumplan algunas condiciones. En este caso nos restringiremos a píxeles con una alta luminosidad (un valor alto en una versión en grises de la imagen). Usad (0.3·R+0.5·G+0.2·B) para crear una versión en niveles de gris de la imagen y volcad su valor máximo. Dividir la imagen por este máximo para tener una imagen BW con un valor máximo de 1. Para hallar la corrección de color usaremos sólo los píxeles cuyos valores (en la imagen B/W normalizada) estén comprendidos entre 0.5 y 0.8. ¿Qué porcentaje de píxeles de la imagen cumplen estas condiciones?

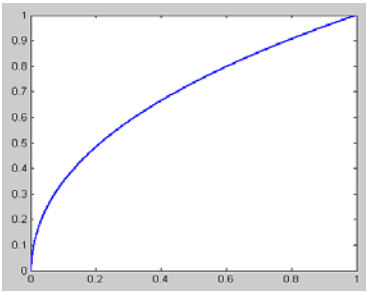
En MATLAB es sencillo implementar estas operaciones sin bucles usando el indexado lógico. El resultado de la comparación ok=(BW>=U0 & BW<=U1) es un array lógico que puede usarse luego para extraer p.e. los elementos de R que verifican dicha condición haciendo RR=R(ok). La media de RR es la media del canal rojo (para los píxeles considerados). Repitiendo esto para los otros canales obtenemos las tres componentes [RR GG BB] que asumimos corresponden a un gris neutro. A partir de [RR GG BB] calcular el vector **div** con los factores a usar en la compensación de color. Adjuntad los valores [RR,GG,BB] así como el vector **div**. Compensar los canales R, G y B como se hizo en el ejercicio 2.

Paso al espacio de color sRGB: aunque hemos corregido el Balance de Blancos, los planos de color R, G y B obtenidos están todavía en un espacio de color propio de la cámara, que depende de factores específicos como los filtros de color usados, sensibilidad del sensor a las diferentes longitudes de onda, etc. Para esta cámara el fabricante recomienda la siguiente transformación desde el espacio de color propio que hemos calculado (planos R, G, B) al espacio de color sRGB (que es el usado en la mayoría de las cámaras):

```
sR = 1.65·R – 0.61·G – 0.04·B
sG = 0.01·R + 1.27·G – 0.28·B
sB = 0.01·R – 0.21·G + 1.20·B
```

Una hecha la transformación, juntad (usando cat) los tres nuevos canales sR, sG, sB en una imagen. Poner a 0/1 aquellos valores de la imagen que sean menores que 0 o mayores que 1, que corresponden a colores que caen fuera del rango de colores reproducible ("gamut") del espacio de color destino. Hacedlo sin bucles, usando condiciones como antes. Adjuntad la imagen (fc_pinta_im) tras el equilibrado de color y conversión a sRGB.

3.5) Aplicación de la función γ.



Aunque ya tiene los colores esperados, la imagen anterior es bastante oscura. El paso al espacio sRGB se completa con una función no lineal o función γ. Esta no-linearidad nos dará una imagen más clara, repartiendo sus valores de una forma más acorde a la percepción del sistema visual. En la figura adjunta se muestra la función γ. Aunque el rango de la entrada y salida es el mismo (intervalo [0,1]), se observa que la función expande la zona correspondiente a los tonos oscuros, donde el ojo es más sensible. De esta forma, la cuantificación posterior (a 8 bits por canal) que se realiza al guardar la imagen se hace menos perceptible. La función usada en el standard es algo más complicada, pero una buena aproximación a la función γ es simplemente:

$$\gamma(x) = x^{0.4167}$$

Aplicar esta función (sin bucles) a los valores de la imagen sRGB creada en el paso anterior. Adjuntad la imagen (fc_pinta_im) resultante tras aplicar la función γ.

3.6) Ajuste de Contraste y Saturación:

Con el paso anterior se completa el revelado propiamente dicho y tenemos una foto aceptable. Para adaptar el aspecto final de la foto al gusto del usuario, las cámaras tienen opciones que modificar el contraste (rango de valores) y saturación (colores más o menos vivos) de la imágenes.

Para implementar esta funcionalidad trabajaremos en el espacio de color HSV. Los planos del nuevo espacio corresponden a:

1. Hue (H): color o tono (rojo, azul, ...) predominante en cada píxel. Este plano no se modificará, ya que no se desea alterar los colores de la imagen.
2. Saturación (S) de cada píxel. Un valor entre 0 y 1 indica la "pureza" del color del píxel. Valores altos corresponden a colores puros y vibrantes. Al bajar obtenemos tonos pastel, cada vez más apagados hasta llegar a gris si S~0.
3. Value (V): luminancia de la imagen: corresponde a una versión en niveles de gris de la imagen, con valores entre 0 (negro) y 1 (blanco).

Una vez en dicho espacio modificaremos el 2º plano (para alterar la saturación) y el 3er plano (para cambiar el contraste).

Extraer el 3er plano (V) y ver su histograma: **`hist(V(:),1000); xlim([-0.05 1.05])`** Observaréis que el rango de valores no aprovecha todo el rango [0,1] posible. Usando max(V(:)) y min(V(:)) determinar máximo M y mínimo m del canal V.

Una forma automática de aumentar el contraste es re-escalar el plano V como se hizo en el apartado a) para que el valor mínimo m vaya al 0 y el máximo M al 1. Adjuntad el nuevo histograma (que debe ahora cubrir el rango completo de 0 a 1). Si deseamos aumentar más el contraste podemos usar para m un valor mayor que el mínimo de la imagen de partida (m=K·m, con K>1) y para M un valor menor que el máximo original (M/K). En este caso tendremos algunos píxeles saturados (fuera del rango [0,1]). P.e. si el mínimo era 0.1 y decidimos usar m=0.12, los valores de la imagen original entre 0.1 y 0.12 serán negativos tras aplicar la transformación. Dichos valores se pondrán a 0 al final y aparecerán como negros en la imagen. Al aumentar el contraste se suele aceptar que un pequeño porcentaje de píxeles (tanto por arriba como por abajo) queden saturados tras re-escalar la imagen.

Escribir una función **`function P=retoca(P,K)`** para implementar este procedimiento. La función recibe un plano P, calcula su mínimo m y máximo M y los modifica con el 2º parámetro K: m=m·K y M=M/K. Usando los nuevos valores de m y M se reescala el plano P (poniendo a 0/1 los valores que se salgan del intervalo [0,1]). Dentro de la función mostrad el histograma del plano P antes y después de la transformación. Usad subplot(211) y subplot(212) antes de la llamada a hist() para mostrar ambos histogramas en una misma figura. Adjuntad código de vuestra función.

Aplicar la función al plano de luminosidad V usando K=1.05. Adjuntad la imagen del histogramas de V antes y después de la transformación. ¿A qué se debe el pico que aparece en el histograma de después para el valor x=1?

Obtener la imagen recuperada usando la modificación del plano V. Adjuntad la imagen resultante tras el aumento de contraste, visualizándola con fc_pinta_im().

La saturación se modificará con la misma función. Extraer el plano de saturación S y modificarlo usando ahora K=1.25. Mostrad los histogramas de S de antes/después.

Finalmente usar ambos planos modificados (V y S) para reconstruir una imagen con el contraste y saturación incrementados. Mostrarla con fc_pinta_im() y adjuntadla.

Comparar los histogramas (creados por fc_pinta_im) de las dos últimas imágenes, entre las cuales se ha aumentado la saturación. ¿Qué efecto se observa en el histograma de una imagen en color si se aumenta la saturación? Justificar.

3.7) Almacenamiento: tras terminar de procesar la imagen, volver a convertirla a bytes, multiplicando la imagen final por 255 y pasándola a bytes con uint8(). Es en este paso donde se lleva a cabo la reducción a 256 niveles (8 bits) por canal. Ya solo nos queda almacenar la imagen en la tarjeta de memoria usando algún tipo de formato de imágenes. Si nos decidimos por un formato sin pérdidas (p.e. TIF) podemos hacer: `imwrite(im,'foto.tif');` ¿Tamaño en disco del fichero foto.tif? Justificad en función del tamaño de la imagen. Para ver el tamaño de un fichero podéis hacer `>>!dir foto*` en la ventana de comandos.

Para ahorrar espacio lo habitual es seleccionar en el Menú la opción de guardar las fotos como JPG. Al elegir este formato habrá una opción ("Best", "Fine", ...) que determina el parámetro de calidad usado en la codificación JPG (cuanto más alto, hasta 100, mejor calidad):

```
imwrite(im,'foto_99.jpg','Quality',99);
```

¿Qué factor de compresión se obtiene con un factor de calidad Q=99? ¿Y con Q=90? El factor de compresión (S_0/S) es el cociente entre el tamaño en disco que ocuparía la imagen sin comprimir (S_0) y el tamaño de la imagen una vez comprimida (S) y nos dice cuántas veces más pequeña es la imagen comprimida que la original.

El parámetro usado en la compresión JPG especifica un objetivo de calidad, no del factor de compresión. Cargad la imagen 'foto2.tif' y guardarla como un JPG también con Q=99 y Q=90. ¿Cuáles son los ratios de compresión obtenidos para la nueva imagen? ¿Cuál de las dos imágenes es más "difícil" de comprimir? ¿Por qué?

En la entrega subid junto con esta hoja de respuestas un script revelado.m con TODO el CÓDIGO USADO en EL PROCESO DE REVELADO (problema 3).

El fichero debe incluir todas las funciones auxiliares que hayáis escrito para que el script funcione. No hace falta incluir las que yo os he dado como fc_pinta_im() ni los ficheros de datos de partida.

Referencia: para verificar vuestros pasos os adjunto el aspecto de las imágenes que deberíais ver al final de las diferentes etapas.

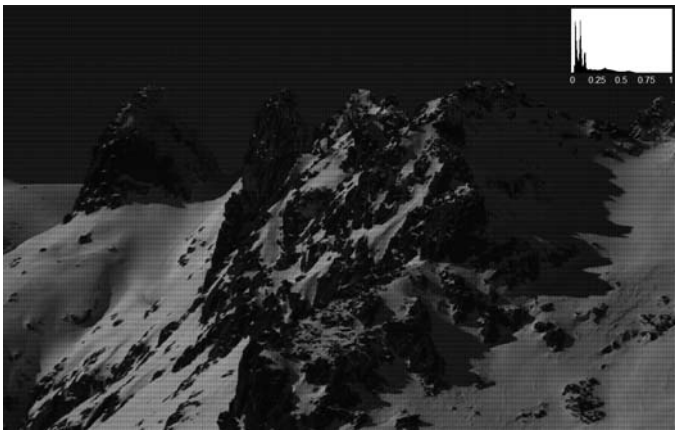
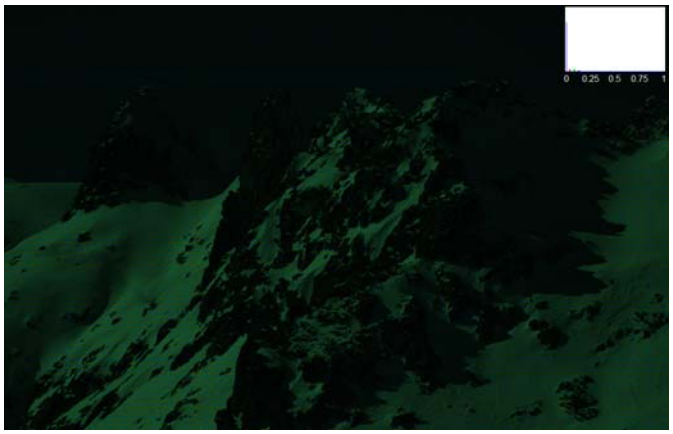
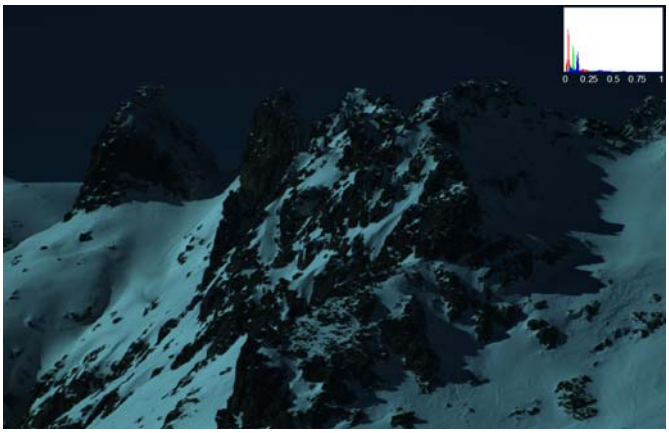


Imagen RAW



Mosaico



Mosaico Interpolado

Balance de Blancos + paso a sRGB

Aplicación de función γ

Aumento de contraste y Saturación