Laboratorio sobre sensores y revelado

Ejercicio 0: Se trata de recordar algunas propiedades de un ruido aleatorio con una distribución normal (simulado en MATLAB con la función randn(). randn(N,1) crea un vector con muestras de ruido de media m=0 y desviación standard σ=1.

Crear un vector con N=100000 muestras de "ruido" usando n1=randn(N,1).
Verificar su desviación standard σ (std) y su media m (mean).

Multiplicar por 2 el vector n1 (lo que equivale a amplificarlo). ¿Qué le pasa a su σ?

Crear 2º vector con otras N=100000 muestras (diferentes) de ruido n2=randn(N,1)
Comprobad que su σ es también 1.

¿Cuál es la σ (std) de la suma de n1 y n2? ¿Y de su resta?

Ejercicio 1 (ruido de lectura): En 'black.pgm' tenéis una "foto" con la tapa del objetivo puesta tomada con mi cámara. Dicho archivo se ha obtenido a partir de una imagen RAW usando el programa dcraw con las opciones (dcraw -D -4) que extrae los datos directos del sensor contenidos en la imagen RAW sin "revelarlos". Los datos contenidos en 'black.pgm' son de tipo uint16, adecuados para guardar los valores de 0 a 4095 que el conversor ADC de 12 bits de mi cámara suministra.

El patrón de BAYER usado en esta cámara corresponde al esquema:

R	G1
G2	B

a) Leer los datos usando imread. La matriz resultante es bidimensional, ya que la información de color está multiplexada (repartida) espacialmente.

b) Extraer los 4 conjuntos de píxeles correspondientes a cada color (considerar G1 y G2 como colores distintos). Guardarlos en sendas matrices R, G1, G2, B

Esto es muy sencillo usando el indexado de MATLAB. Si im es la imagen con los datos originales, para sacar el canal rojo bastaría hacer:

```
R = im(1:2:end,1:2:end)   % 1ª,3ª,5ª columna de la 1º,3º,.. filas
```

Repetir para extraer los demás colores. Adjuntar código utilizado.

c) Usando mean2 y std2 (versiones 2D de mean y std) calcular la media y desviación standard de cada canal. Dado que no se han recibido fotones durante esta toma lo que estáis haciendo es estimar las características del ruido de lectura (R) de la cámara. Rellenar la tabla adjunta con los resultados para cada canal:

	R	G1	G2	B
Media (m)				
Sigma (σ)				

d) Visualizar un histograma (usando fc_imhist) de uno de los canales. ¿Creéis que esta cámara está usando un offset para conservar los valores "negativos" del ruido? ¿Cuál sería el valor del cuantificador correspondiente a un negro en esta cámara?

e) Observando las σ's de los 4 canales ¿notáis alguna diferencia? Dado que los "sensels" de cada canal son básicamente idénticos ¿qué puede estar haciendo la cámara para provocar esas diferencias?

Ejercicio 2 (Balance de blancos): La foto de la izquierda se ha tomado bajo una luz incandescente pero se le ha dicho a la cámara que use el balance de blancos correspondiente a la luz diurna. El resultado es una foto claramente naranja debido al color de la luz.

En la foto de la derecha se le ha dicho a la cámara que use una estimación AUTO del balance de blancos. Vemos que el resultado es mucho mejor (el papel se ve mucho más blanco que en la foto de la izquierda).

Sin embargo, todavía queda un cierto toque de color que el algoritmo automático no ha sido capaz de eliminar completamente. Vamos a hacerlo manualmente.

Aunque para corregir el balance de blancos es mejor hacerlo trabajando sobre los datos RAW del sensor también se puede hacer a partir de una imagen JPG ya "revelada".

En el archivo "color.jpg" tenéis la foto de la derecha. Se trata de:

- Ver la imagen usando image(). Usar truesize (o fc_truesize) para verla con el tamaño/relación de aspecto adecuada.

- Extraer un rectángulo de la imagen en la zona del papel, justo encima de las fichas roja/amarilla con coordenadas Y≅(650-700) y coordenadas X≅(650-750) Visualizarlo para ver que habéis escogido bien la zona. Visto aisladamente, ¿se ve blanco?

- Calcular la media de cada plano de esa zona usando mean2. Meter los tres valores en un vector rgb. Idealmente dichos valores deberían ser iguales ya que el papel es un color neutro (gris/blanco).

- Hacer la media del vector rgb, m=mean(rgb) y calcular el vector de corrección como c = m./rgb (sería lo que le "falta" a cada canal para ser como la media m deseada). Si aplicásemos c a las tripletas rgb de la zona del papel obtendríamos valores RGB similares entre si (grises) e iguales a la media m.

- Aplicar los factores de corrección del vector c a toda la imagen: c(1) debe multiplicar a los valores del canal rojo, c(2) al verde y c(3) al azul. Previamente convertir la imagen a double para poder hacer las cuentas.

- Tras aplicar los factores correctivos, reconvertir la imagen de nuevo a bytes usando im=uint8(im) y visualizarla con image.

Adjuntar vuestro código y los valores obtenidos para el vector rgb y el vector de correcciones.

Adjuntar la imagen original y la resultante.

Ejercicio 3: En el ejercicio 1 estimamos el nivel de ruido de lectura de un sensor a través de una toma de un black-frame. El objetivo de este ejercicio es cuantificar otras fuentes de ruido del sensor (shot-noise, no uniformidad).

Los datos de partida están extraídos de una serie de frames (datos RAW del sensor) obtenidos con 11 exposiciones sucesivamente mayores, con tiempos de exposición entre 1/1000 seg (izquierda) y 1/30 seg (derecha):

Por simplicidad os doy los datos correspondientes sólo al canal verde, de forma que no tenéis que separar los distintos canales. Haciendo >> load datos_ruido veréis los datos como una matriz **datos** de tamaño (50x50) x 11 (11 exposiciones). El área considerada corresponde a una pequeña (50x50) zona homogénea de la foto.

Si no tuviéramos ruido, al ser una zona homogénea, todos los valores del sensor (en unidades ADU del cuantificador) serían iguales entre sí. La desviación de dichos valores respecto a su media nos permitirá estimar el nivel de ruido del sensor con el que se tomaron las fotos.

En las circunstancias en las que se tomaron estas fotos, las fuentes de ruido principales son: ruido de lectura R, ruido shot-noise S y el "ruido" W debido a las diferencias entre el comportamiento de los diferentes "sensels". Las desviaciones estándar σ correspondientes a estos tres tipos de ruido pueden expresarse como:

- Ruido de lectura: $\sigma_R = cte$ (independiente de la exposición E)

- Ruido shot noise: $\sigma_S = \sqrt{G \cdot E}$ (G = ganancia AMP, 1/G = fotones por ADU)

- Ruido de no-uniformidad: $\sigma_W = K \cdot E$

donde E es la exposición mediada en unidades de ADU's (salida del ADC).

Como vimos, al sumar ruidos con desviación $\sigma 1, \sigma 2, \sigma 3$, la desviación del ruido suma es:

$$\sigma^2 = \sigma_1^2 + \sigma_2^2 + \sigma_3^2$$

En este caso, para los tres tipos de ruido considerados podemos escribir que:

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

Lo que vamos a hacer es:

1) Crear sendos vectores columna E y S de tamaño 11 x 1 para guardar los datos de cada frame.

2) Hacer un bucle explorando las 11 exposiciones, cada una de ellas de tamaño 50x50. Cada subimagen podéis extraerla como Q=datos(:,:,k). En cada caso:

a) Calcular la exposición como la media (mean2) de los valores de la imagen Q (acordaros de restar el offset del negro, que en esta cámara es de 128). Guardar el valor de la exposición en E(k).

b) Calcular la desviación standard al cuadrado de cada frame Q usando la función std2() y elevando el resultado al cuadrado. Guardar el valor en S(k)

Una vez que tenemos estos datos podemos separar los distintos tipos de ruido (lectura, shot-noise y no-uniformidad) identificando las constantes que les acompañan.

Podemos hacerlo porque los tres ruidos se comportan de forma distinta según aumenta la exposición (uno es constante, el otro es lineal y el tercero va con el cuadrado de la exposición).

Recordando que $\sigma^2 = \sigma_r^2 + G \cdot E + K^2 \cdot E^2$ nos bastará hacer un ajuste de los datos de ruido S (σ^2) respecto a E usando como la base $\{1, E, E^2\}$.

Los coeficientes del ajuste corresponderán a los valores de σ_r^2 G y K^2 y nos indicarán la importancia relativa de los tres tipos de ruido en el sensor

Para hacer el ajuste crear la matriz H como [E.^0 E E.^2] (siendo E el vector columna con los valores de las exposiciones) y resolver el sistema:

```
c = H \ S
```

Hacer una gráfica de los datos medidos y del ajuste encontrado. Para ello crear un vector ee=(0:4000) con valores de exposición entre 0 y 4000 y hacer una gráfica de la curva ideal obtenida en el ajuste: c(1) + c(2)*ee + c(3)*ee (en azul).
Superponer sobre esta curva un gráfico (con puntos discretos 'ro') con los datos medidos (S en el eje Y frente a E en el eje X).

Adjunta código y gráfica de ajuste

Recordando que $S = \sigma^2 = \sigma_r^2 \cdot 1 + G \cdot E + K^2 \cdot E^2$ los coeficientes obtenidos en el ajuste para 1, E y E^2 están relacionados con los parámetros de los diferentes ruidos de la forma siguiente:

- σ_r = sqrt(c(1))
- G = c(2)
- K = sqrt(c(3))

¿Qué valor de σ se obtiene para el ruido de lectura? ¿Es similar a la que obtuvimos usando la técnica del black–frame para el canal verde?

¿Cuántos fotones necesita esta cámara para incrementar un nivel del ADC?

¿Cuántos fotones pueden caer sobre cada elemento de este sensor antes de que se llene?

¿De qué orden (expresarla en %) es la constante K que indica la no-uniformidad de los receptores?

Consideremos una exposición del orden de 2000 (en unidades ADUs), lo que para esta cámara (con un conversor de 12 bits ≅ 4000 niveles) corresponde a una exposición media ¿cuáles son las σ's de los tres tipos de ruidos considerados? ¿Cuál es el más importante?

Proyecto: para entregar por parejas

Adjuntar código empleado / imágenes / respuestas / comentarios

Revelado digital de una imagen RAW

En 'raw_red.pgm' podéis encontrar una imagen con los datos RAW (uint16) de una fotografía. Este tipo de datos podéis obtenerlo a partir de una imagen RAW de vuestra cámara usando el programa dcraw con las opciones –D -4 (extraer los datos del sensor sin revelar). En este caso yo he hecho un submuestreado adicional de la imagen por un factor 2 para que no tengáis problemas de memoria en MATLAB al manejar una imagen muy grande. La imagen RAW es de unos 1000x1500 datos.

Cargar la imagen (imread) y pasarla a double para poder hacer las cuentas:

1. Normalizar la imagen al intervalo [0,1]. El negro debe ir al 0 y el valor más alto posible al 1. Para esta cámara el nivel del negro es de 128 y el valor máximo posible del sensor es de 4095 (ADC de 12 bits). Poner a cero los valores por debajo del nivel del negro, que saldrán negativos una vez hecho el escalado.

2. Crear 3 imágenes R, G, B llenas de 0's del mismo tamaño que imagen RAW original. Trasladar a cada una de ellas la información del correspondiente canal de la imagen RAW en sus correspondientes posiciones. Recordad como están repartidos los canales Rojo/Verde/Azul en la imagen RAW. Dejar los valores desconocidos de cada canal como ceros.

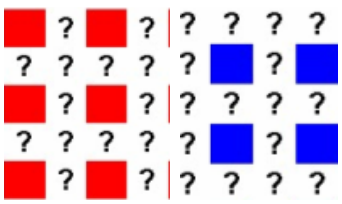
3. Aplicar balance de blancos (WB) a las tres canales. En una primera instancia usar los valores determinados por el AUTO WB de la cámara:

c=[0.75 0.75 1.0].

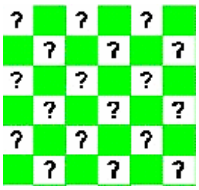
Multiplicar cada canal por el correspondiente factor.

4. Algoritmo de "demosaicing". Vamos a usar un sencillo algoritmo de interpolación linear sin complicaciones.

Para los planos rojo (R) y azul (B) recorrer las filas donde tenemos información (1ª, 3ª … para el rojo, 2ª, 4ª, … para el azul) y calcular los valores que faltan como la media de los dos vecinos a derecha e izquierda. Una vez terminada esta parte, rellenar las filas vacías como la media de los valores que encontramos arriba y abajo.



Escribid vuestros bucles de forma que nunca se necesite un valor que caiga fuera de la matriz conocida. Eso significará que tendremos valores sin rellenar en los bordes, pero no importa, luego los eliminaremos.



Para el canal verde rellenar los píxeles vacíos como la media de los 4 vecinos (arriba, abajo, izquierda, derecha). De nuevo, ignorar los píxeles de los bordes que nos pedirían el valor de píxeles que caen fuera de la imagen.

Finalizado el proceso de interpolación linear, eliminar el borde de las imágenes interpoladas (R, G y B). Para ello, basta eliminad la primera y última fila, junto con la primera y última columnas. De esta forma eliminamos las zonas donde había píxeles que no se han interpolados correctamente.

Si hubieseis usado un esquema de interpolación usando más vecinos tendríais más filas/columnas con información incompleta en los bordes. Esta es la razón por la que el tamaño de vuestras fotos es un poco menor que el tamaño del sensor según las especificaciones.

5. Crear una imagen del mismo tamaño que R, G o B, pero ahora con los 3 planos de color. Colocar cada uno de vuestros planos R, G y B en los correspondientes planos de color de la nueva imagen.

Visualizarla con imshow. Adjuntar imagen.

6. Llegado este momento tenéis ya una imagen color, pero todavía no es válida: es muy oscura y está en un espacio de color "RAW" muy peculiar, influenciado por un montón de factores de la cámara (transmitancia de filtros de color, sensibilidad del sensor a las diferentes longitudes de onda, etc.)

Para la cámara de la que he sacado esta imagen RAW, el fabricante da la siguiente matriz para pasar del espacio de color propio RAW al espacio de color canónico XYZ:

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} & M1 & \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix} \qquad M1 = \begin{pmatrix} 0.6844 & 0.1651 & 0.1009 \\ 0.3600 & 0.7634 & -0.1235 \\ 0.0389 & -0.0575 & 1.1072 \end{pmatrix}$$

Una vez en el espacio de color standard XYZ es fácil pasar al espacio sRGB usado por la mayoría de las cámaras. En Wikipedia podemos encontrar la nueva matriz de cambio de XYZ a sRGB:

$$\begin{pmatrix} sR \\ sG \\ sB \end{pmatrix} = \begin{pmatrix} & M2 & \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \qquad M2 = \begin{pmatrix} 3.2406 & -1.5372 & -0.4986 \\ -0.9689 & 1.8758 & 0.0415 \\ 0.0557 & -0.2040 & 1.0570 \end{pmatrix}$$

Como para pasar de RAW a sRGB hay que aplicar primero M1 y luego M2, es más eficiente calcular la matriz M=M1*M2. De esta forma pasamos en un solo paso de RAW a sRGB, evitando los cálculos asociados con la aplicación de una 2ª matriz.

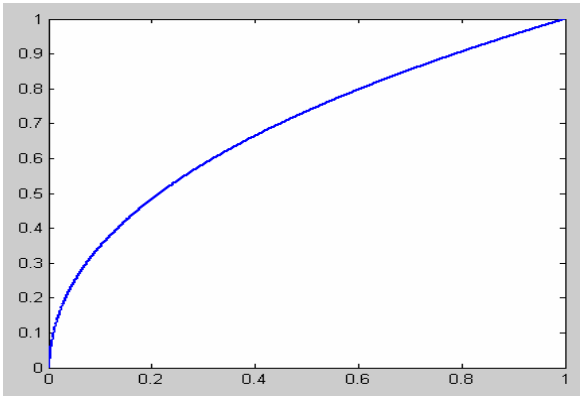
Una vez en el espacio sRGB, poner a 0 aquellos valores negativos y a 1 aquellos que superen la unidad. Se dice que son colores que no están en el rango de colores reproducibles ("gamut") del espacio de destino.

7. Para completar la transformación a sRGB hay que aplicar una no linearidad o función γ. Esta no-linearidad es la responsable de hacer la imagen más clara.

La función transforma el intervalo [0,1] en [0,1] pero expandiendo la zona de negros donde el ojo es más sensible. De esta forma se consigue que la posterior reducción a 8 bits por canal sea perceptualmente óptima. La función no lineal definida en el standard sRGB es:

$$y(x)=\begin{cases} 12.92\cdot x & \text{si} \quad x<0.0031308 \\ (1+a)\cdot x^{1/\gamma}-a & \text{si} \quad x\geq 0.0031308 \end{cases}$$

con a = 0.055 y γ = 2.4. (ver figura adjunta).



Aplicar la función a vuestra imagen. Podéis hacerlo con una sola orden en MATLAB, usando la imagen como una variable.

8. Ya casi hemos acabado. Volver a escalar la imagen entre 0 y 255 (multiplicando por 255) y volver a pasar a bytes con la función uint8().

Ahora sería el momento de retocar el brillo/contraste de la imagen. Para esta imagen multiplicar todos sus valores por un factor adicional de 1.2 aumenta el brillo y da una imagen más luminosa.

Visualizar la imagen final con image(). Adjuntar la imagen resultante.

9. A pesar de haber pasado a bytes vuestra imagen sigue siendo muy grande. ¿Cuánto ocuparía en disco si la guardaseis sin comprimir? Recordad que la imagen original ocupaba 4 veces más que ésta con la que estáis trabajando.
Guardar la imagen comprimida como un JPG con calidad 90 usando:

```
imwrite(im,'xxxxx.jpg','Quality',90);
```

¿Qué tamaño tiene la imagen guardada en disco? Esta sería la imagen que vuestra cámara guardaría en la tarjeta de memoria si la cámara no tiene opción de guardar RAWs o si habéis escogido la opción de JPGs.

ADJUNTAR CÓDIGO de TODO EL PROCESO.

Extras:

- La imagen obtenida es correcta pero el AUTO WB de la cámara no terminado de corregir por completo la iluminación.

  En la imagen, entre las fichas de parchís y la figurita del esquiador podéis extraer una zona de papel que debería verse gris. Siguiendo el ejercicio 3, determinar los valores medios de la zona para los valores del sensor en los tres canales (las imágenes R, G y B) y usarlos para encontrar las correcciones adecuadas. ¿Valores de los multiplicadores encontrados?

  Volver a correr todo el proceso con el nuevo balance de blancos y adjuntar la nueva imagen. Notad que esta vez (al contrario que en el ejercio 3) hallamos los factores de corrección trabajando sobre los datos RAW del sensor (y usamos las correcciones directamente sobre los datos RAW).

- Otro procesado adicional que suelen hacer las cámaras es alterar la saturación de color de la imagen para obtener colores más o menos vivos. Suele ser una opción de menú donde se indica el nivel de saturación deseado como una corrección: …,-1,0,+1,… El nivel 0 indica dejar los colores como salen del proceso. Valores positivos incrementan la saturación y viceversa.

  Para variar la saturación de una imagen lo mejor es pasar a un espacio de color (como el HSV) que separa la saturación en un plano de color diferente.

  Convertir la imagen a HSV usando rgb2hsv() (disponible si habéis instalado en vuestra casa el nuevo MATLAB con la nueva licencia campus UPM).
  Hacerlo **entre los pasos 7 y 8**, antes de reescalar a 255 y pasar a uint8, cuando vuestra imagen es todavía de tipo double con valores entre 0 y 1.

  Una vez en el nuevo espacio de color, podéis acceder a la saturación (un valor entre 0 = gris y 1 = color saturado) en el 2º plano de color de la imagen.
  Probad a hacer sat = 0.1 o sat = 0.9 para toda la imagen. Volver al espacio de partida (hsv2rgb) y visualizar las imágenes resultantes. Adjuntar vuestros resultados.

  Lo anterior son los casos extremos. Nosotros deseamos modificar ligeramente los valores de saturación, aumentándolos un poco si deseamos incrementar la saturación y viceversa. Lo más sencillo es aplicar una función a la saturación para modificarla. La función debe seguir convirtiendo el intervalo [0,1] en [0,1], respetando el rango de valores esperado para la saturación.

  Si aplicamos la función y=x a los valores de la saturación la dejamos igual que antes. Por otra parte, funciones que queden por encima de y=x en el intervalo [0,1] aumentarán la saturación y las que queden por debajo la disminuirán.

  Una elección sencilla es usar la familia $y = x^P$. Dependiendo de si p>1 o p<1 podremos alterar la saturación en la dirección deseada.

Usar esta aplicación con p=0.8 para incrementar la saturación de la imagen final. Tras modificar la saturación, volver al espacio RGB para visualizar y adjuntar imagen resultante.

- ¿En que zonas en la imagen pueden aparecer efectos debidos al algoritmo de "demosaicing" ingenuo que hemos usado?

  Detectar alguna de esas zonas y hacer zoom sobre ellas.
  Adjuntar las imágenes obtenidas, indicando los efectos del "demosaicing".

  En esta imagen los efectos están reforzados al ser un submuestreo de una imagen RAW, por lo que el muestreo de color es aún más inadecuado.