


PROYECTO 3 (sensores y revelado digital)

Ejercicio 1 (10%) Balance de blancos: Aunque para corregir el balance de blancos es mejor trabajar directamente sobre los datos RAW del sensor también se puede hacer a partir de una imagen JPG ya "revelada".

La foto de la izquierda está tomada con luz incandescente pero usando el balance de blancos de la cámara correspondiente a luz diurna. El resultado es una foto claramente naranja debido al color de la luz.



En la 2ª foto la cámara estaba en modo AWB (Auto White Balance) para que corrija automáticamente el balance de blancos. El resultado es mejor, pero todavía queda un toque de color que el algoritmo automático no ha sido capaz de eliminar. Vamos a hacerlo manualmente, seleccionando una zona "neutra" de la imagen. En el archivo "color.jpg" tenéis la foto de la derecha. Se trata de:

- Leer la imagen y convertirla a double usando `im2double()`, que convierte a double y además normaliza los valores de la imagen entre 0 y 1. Usando el cursor de datos  de MATLAB pinchad en algún punto del papel y extraer una zona de ± 20 píxeles alrededor de dicho punto. Visualizar la imagen extraída para ver que habéis escogido la zona correcta. Observaréis que, vista de forma aislada, la zona tiene una clara tonalidad no neutra. [Adjuntad imagen](#).
- Calcular la media de cada plano de color de la subimagen con `mean2` y usar esos tres valores para formar un vector **neutro**. Idealmente dichos valores deberían ser iguales al ser el papel de un color neutro. Haced la media m del vector **neutro** y calcular un nuevo vector **comp** (compensación del color) como $m./\text{neutro}$. Los 3 valores de **comp** son lo que le "falta" a cada canal para alcanzar el valor medio m . [Adjuntad valores de neutro y comp](#). Aplicando esta compensación a los canales rgb de la subimagen (multiplicando el canal rojo por `comp(1)`, el verde por `comp(2)`, ...) obtendremos valores RGB similares entre si (grises). [Adjuntad subimagen corregida](#).
- Aplicar ahora los factores de compensación hallados a toda la imagen. Tras aplicar estos factores correctivos, reconvertir la imagen de nuevo a bytes usando `im=uint8(im*255)` y visualizarla con `imshow`.

[Adjuntar código e imagen final resultante.](#)

Ejercicio 2 (10%): Manejo de imágenes RAW / Ruido de lectura

a) En este apartado recordaremos algunas propiedades de un ruido aleatorio con una distribución normal. La función `randn(N,N)` crea una matriz $N \times N$ con muestras de un ruido aleatorio de media $m=0$ y desviación standard $\sigma=1$.

Crear una imagen de 1000×1000 usando `im1=randn(N,N)`. Verificar que su media (`mean2`) y desviación standard (`std2`) son aproximadamente 0 y 1.

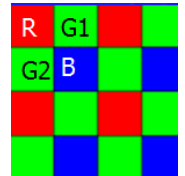
Crear una 2ª imagen (1000×1000) píxeles, `im2=3*randn(N,N)`. Verificar que su $\sigma=3$. ¿Cuál es la σ de la suma de (`im1+im2`)? ¿Y de su resta? ¿Cuál es la regla para la desviación standard σ de la suma de dos ruidos independientes con σ_1 y σ_2 ?

Dada una imagen de media m_1 y $\sigma=\sigma_1$, para cambiar su media a m_2 y su desviación standard a σ_2 :

- 1º) Pasar a imagen con $m=0$ y $\sigma=1$: `im=(im - m1)/ σ_1`
- 2º) Pasar a imagen con $m=m_2$ y $\sigma=\sigma_2$: `im=im * σ_2 + m2`

Modificar `im1` para tener media $m=50$ y desviación standard $\sigma=4$. Comprobadlo. ¿Cuál sería ahora la desviación σ de la imagen (`im1+im2`)? Comprobadlo con `std2`.

b) El fichero 'black.pgm' corresponde a una foto con el objetivo tapado y exposición muy corta. Este archivo se ha obtenido de una imagen RAW usando el programa `dcraw` (opciones `-D -4`) que extrae los datos originales del sensor de la imagen RAW y los guarda en formato `pgm` sin revelarlos. Los valores de 'black.pgm' son de tipo **uint16**, con los valores (de 0 a 4095) que el ADC de la cámara (**12 bits**) suministra. El CFA (Color Filter Array) que usa esta cámara es el patrón de BAYER que se muestra a la derecha.



1. Leer la imagen de `black.pgm` con `imread`. La imagen leída es bidimensional, ya que la información de color está multiplexada (repartida) espacialmente.
2. Extraer los 4 conjuntos de píxeles de cada color (considerando G1 y G2 como colores distintos) y guardarlos en 4 matrices `R`, `G1`, `G2`, `B`. Esto es sencillo usando el indexado de MATLAB. Si `im` es la imagen con los datos RAW originales, para sacar el canal rojo bastaría hacer (ver figura anterior):

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

Repetir para extraer los demás colores. [Adjuntar código utilizado.](#)

3. Visualizar el histograma del canal rojo usando `hist(double(R(:)),256)`; [Adjuntar histograma.](#) ¿Usa esta cámara un offset para conservar los valores "negativos" del ruido de lectura? ¿Cuál es el valor en los datos RAW que corresponde a un negro puro en la cámara?
4. Calcular la media y σ de cada canal (usando `mean2` y `std2`). Como el sensor no ha recibido fotones durante esta toma lo que estáis haciendo es estimar los niveles del ruido de lectura (Read Noise) de la cámara. [Rellenar la tabla de la hoja de respuestas con los resultados de cada canal.](#)

Ejercicio 3 (15%): Al tener el objetivo tapado y usar una exposición rápida el único ruido presente antes era el ruido de lectura. En este ejercicio vamos a cuantificar otras fuentes de ruido de este sensor (shot-noise y no uniformidad).

Los datos de partida han sido extraídos de los datos RAW de una serie de frames correspondientes a 11 exposiciones cada vez más largas (a saltos de 1/2 stop), con tiempos de exposición entre 1/1000 seg (izquierda) y 1/30 seg (derecha):

Por simplicidad os doy los ya datos extraídos de un solo canal (el verde), de forma que no hace falta separar los distintos canales como hicimos en el ejercicio anterior. Estudiaremos una pequeña área (50x50) de una zona homogénea de la foto. Con un `>> load ruido` veréis los datos como una matriz **ruido** de tamaño (50 x 50) x 11 (11 exposiciones). En la figura adjunta se muestran estas 11 imágenes:



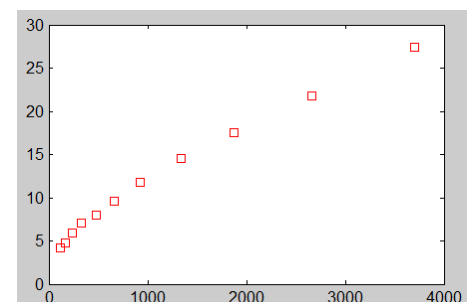
En las circunstancias en las que se tomaron estas fotos, las tres fuentes de ruido más importantes son: ruido de lectura R , shot-noise S y el "ruido" W debido a las diferencias entre la ganancia de los diferentes "sensels". El problema es que, al contrario que en el black-frame anterior aquí no podremos medir cada uno de ellos por separado, ya que en los datos RAW de los que disponemos ya se han sumado.

Lo que si podemos hacer es medir el **ruido total** de cada exposición. Idealmente, al ser una zona homogénea, si no hubiera ningún ruido, todos los valores de la imagen RAW (en unidades del ADC) serían iguales entre sí dentro de cada exposición. Por lo tanto, la desviación (σ) de dichos valores respecto a su media nos permite estimar el nivel de ruido presente en cada exposición. Se trata de:

1. Reservad dos vectores **columna** E y S de tamaño 11 x 1 para ir guardando la exposición y nivel de ruido (σ) que obtengamos de cada frame.
2. Haced un bucle barriendo las 11 exposiciones y en cada paso extraer la k -ésima exposición. Para cada frame de 50x50:
 - a) Guardar en $E(k)$ la exposición (unidades ADU) del frame como la media (`mean2`) de los valores del frame (acordaros de restar el offset o nivel del negro encontrado en el ejercicio anterior).
 - b) Calcular su desviación standard σ con `std2()` y guardad su valor en $S(k)$.

Al terminar el bucle, haced una gráfica de la desviación standard S para las diferentes exposiciones E usando puntos discretos 'rs'. Debéis obtener una gráfica como la de la figura adjunta.

Adjuntad código para hallar E y S y gráfica obtenida.



Aunque no hayamos podido medir por separado las distintas contribuciones al ruido total veremos qué es posible separarlas a posteriori. El ruido total S es la suma de 3 ruidos distintos por lo que verificará (ejercicio 2a) que:

$$S^2 = \sigma^2 = \sigma_R^2 + \sigma_S^2 + \sigma_W^2$$

siendo σ_R , σ_S , σ_W , las σ 's de los tres tipos de ruido involucrados:

- Ruido de lectura: σ_R^2 (constante, independiente de la exposición E)
- "shot noise", proporcional a \sqrt{N} , $\sigma_S^2 = (G \cdot \sqrt{N})^2 = G^2 \cdot N$ con G = ganancia.
- Ruido debido a la no-uniformidad entre píxeles: $\sigma_W = (\delta \cdot N)^2$.

Como la exposición (en ADU's) es $E = G \cdot N$, podemos escribir la dependencia del ruido S con respecto a la exposición como:

$$S^2 = \sigma^2 = \sigma_r^2 + G \cdot E + \left(\frac{\delta}{G}\right)^2 \cdot E^2$$

Vamos a separar las contribuciones de los tres ruidos (y estimar sus parámetros σ_r , G , δ) aprovechando que su varianza se comporta de forma distinta al aumentar la exposición (constante, lineal con E , y cuadrática con E).

Se trata de ajustar los datos de S^2 frente a E (exposición): $S^2 = c_1 + c_2 \cdot E + c_3 \cdot E^2$.

Comparando las 2 expresiones anteriores de S^2 , relacionar los coeficientes de ajuste c_1, c_2, c_3 con los parámetros σ_r^2 , G y δ . [Adjuntad una foto de las relación deducida.](#)

Para resolver el ajuste haremos como vimos en Algorítmica Numérica: creamos la matriz $H = [E.^0 \ E \ E.^2]$ (siendo E el vector columna con el valor de las exposiciones) y resolvemos el sistema sobre-determinado haciendo:

$$c = H \backslash (S.^2)$$

Tras resolver, comprobad el ajuste obtenido sobre los datos medidos. Para ello crear un vector $e = (0:4000)$ con valores de exposición entre 0 y 4000 y hallar la desviación standard predicha por el ajuste:

$$s = \sqrt{c(1) + c(2) \cdot e + c(3) \cdot e.^2}$$

Superponer la gráfica (e, s) del ajuste con línea continua azul ('b') sobre la gráfica (E, S) de antes con los datos medidos ('rs'). [Adjuntar código y gráfica de los datos y ajuste obtenido.](#)

[¿Valor de \$\sigma\$ del ruido de lectura obtenido con este método?](#) Debe salir similar al obtenido en el primer ejercicio para el canal verde.

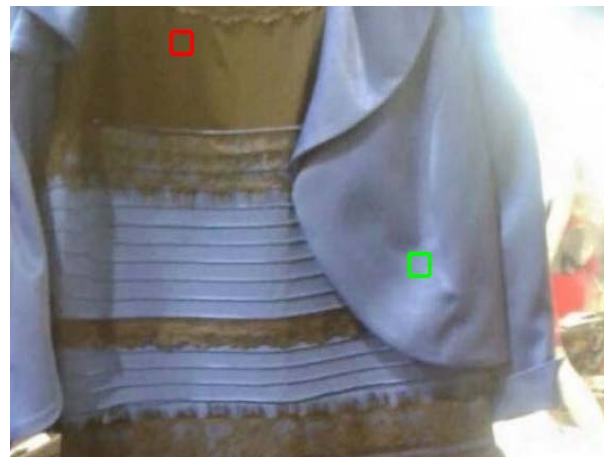
[¿Valor de la ganancia \$G\$ para esta cámara? ¿Cuántos fotones necesita la cámara para incrementar un nivel del ADC? Una vez conocida la ganancia \(relación entre ADU's y # de fotones\), rehacer la gráfica del ajuste para que muestre fotones recibidos \(en el eje X\) y nivel de ruido \(medido en fotones\) en el eje Y.](#)

La constante δ mide la no-uniformidad de la ganancia entre los distintos receptores. El valor significativo es el cociente (δ/G) que, expresado como un porcentaje, indica el % de desviación que podemos esperar entre la ganancia de diferentes sensels. **Dar el cociente (δ/G) obtenido como un %.**

Ejercicio 4 (15%) El proceso del equilibrado de blancos trata de lograr que la fotografía recupere el color "original" disminuyendo posibles influencias externas como la iluminación, etc. Una forma de hacerlo (ejercicio 1) es identificar una zona que pensamos que es neutra y forzarla a que aparezca como un gris. Al aplicar la misma transformación al resto de la imagen esperamos recuperar los colores reales.

El cerebro hace algo similar al mirar una escena con una iluminación no neutra. Se fija en objetos cuyo color cree conocer y les "asigna" el color que "deberían" tener, lo que puede alterar nuestra percepción del resto de la escena.

La imagen adjunta (dress.jpg), tomada con una mala iluminación ilustra este problema. En esta situación, el cerebro intenta identificar zonas de color neutro para detectar el color de la iluminación, y poder corregirla para percibir correctamente el resto de los colores de la escena. Lo malo es que no parece haber consenso entre los distintos cerebros sobre la zona a elegir. El resultado es que algunas personas aprecian claramente que los colores del vestido son blanco + oro, pero otros juran que lo ven como azul + negro.



Vamos a tratar de simular lo que pasa en el cerebro de estos dos tipos de gente.

Antes de seguir convertir la imagen original a double para hacer operaciones, pero manteniendo sus valores entre 0 y 255. **No reescalar a [0,1].**

a) En el primer caso extraer una zona del orden de +/- 20 píxeles alrededor de la zona del cuadrado verde y para cada canal calcular su valor medio (r, g, b). Para simular a la gente que decide que esa zona es el color neutro, aplicaremos una transformación para convertir ese valor en un blanco brillante, correspondiente por ejemplo a (220,220,220). Para ello, cada canal se transformará de forma similar a como se indica aquí para el canal rojo R:

$$R = 220 * (R/r);$$

Con esta transformación el valor de r medido en la zona neutra se convierte en 220. Repetir la operación para los otros dos canales (evitar usar bucles). Tras aplicar la transformación, convertir de nuevo la imagen a bytes usando uint8() y visualizarla con imshow(). La zona de la que hemos extraído los valores (la marcada en verde en la foto) debería verse ahora como un blanco brillante. **Adjuntad los valores (r,g,b) medidos en la zona, el código y la imagen resultante.**

b) Veamos ahora que puede estar pasando en el cerebro de alguien que piensa que la zona marcada en rojo es un tono neutro. Medir los valores medios de cada canal (r, g, b) en esa zona (+/-20 píxeles). *¿Es más clara u oscura que la anterior?*

De nuevo se trata de convertir esa zona en un neutro (con los valores de los tres canales iguales). Pero como esta zona es más oscura que la anterior, el cerebro no tiende a interpretarla como un neutro "luminoso" (blanco), sino como un neutro "oscuro". Por eso, aplicaremos ahora una transformación que convierta el valor (r,g,b) medido a un negro oscuro, p.e. (20,20,20). Para el canal rojo R haríamos:

$$R = 235 * (R-r) / (255-r) + 20$$

Con esta transformación el valor de r se lleva al 20, mientras que valores R ~ 255 mantienen su valor. Repetir para los otros canales, convertir a bytes y visualizar el resultado. La zona marcada en rojo debe verse ahora como un negro oscuro.

Adjuntad el valor medido para (r,g,b), vuestro código y la imagen resultante.

Si todo ha ido bien habréis obtenido dos imágenes con colores "exagerados" que intentan reflejar lo que diferentes personas ven cuando observan la foto inicial.

Ejercicio 5 (50%): REVELADO DIGITAL DE UN IMAGEN RAW

El fichero 'raw.pgm' es una imagen (~ 2000x3000) 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** (que extraen los datos del sensor sin revelarlos). Cada punto en la imagen RAW es la respuesta de cada uno de los ~6 millones de "sensels" del sensor de esta cámara. El programa dcraw guarda los datos del sensor en un fichero gráfico con formato pgm que luego podemos leer con la función `imread()` de MATLAB.

Os iré pidiendo que me mostréis las etapas intermedias. Para visualizarlas, usad la función suministrada: `pinta_im(im, 'Imagen tras balance de blancos')` que muestra la imagen im junto con su histograma en su esquina superior derecha. El 2º argumento etiqueta la figura indicando el paso en el que estemos.

a) Preparación y escalado de los datos: pasar la imagen a double para hacer cuentas y reescalar sus valores en el intervalo [0,1]. El negro debe ir al 0 y el valor más alto posible al 1. Recordad que para esta cámara (ejercicio 2) el nivel del negro es el encontrado en el ejercicio 2 y el valor máximo del sensor (ADC de 12 bits) es de 4095. Aseguraros de que la imagen final no tiene valores fuera del intervalo [0,1]. Debido a estas consideraciones aquí NO es posible usar `im2double()`.

Visualizar la imagen RAW escalada con `pinta_im()`. *Adjuntad imagen resultante. Haced un zoom en la parte del cielo hasta que observéis los "sensels" individuales. Indicad cuáles serían los píxeles "azules" del filtro de Bayer.*

b) Demultiplexado: el siguiente paso es demultiplexar los colores del CFA (filtro de Bayer) para crear los tres planos de la imagen RGB. Normalmente, los fabricantes usan interpolación para conocer el valor de los canales RGB en aquellos puntos donde no hay información. Nosotros haremos algo más sencillo: considerar que la información para rellenar cada uno de nuestros píxeles viene de un grupo de 4 elementos del sensor: dos verdes (G1,G2), uno rojo (R) y uno azul (B). El valor del píxel para ese bloque 2x2 será pues:



--> (R, (G1+G2)/2, B)

Notad que con este esquema se pierde la relación 1:1 entre píxeles y "sensels": un sensor de N Megasensels nos da una imagen con sólo N/4 Megapíxeles.

Extraer los 4 canales (ejercicio 2) y promediar los dos canales verdes G1 y G2. Terminaremos con tres imágenes R, G y B cada una con la información del color rojo, verde y azul de los píxeles, cuyo tamaño será la cuarta parte de la imagen RAW inicial. Juntar las tres imágenes R, G y B para formar una imagen en color usando la función `im=cat(3,im1,im2, ...)` que apila varias imágenes 2D (en este caso R, G y B) a lo largo de la tercera dimensión. [Adjuntar imagen resultante.](#)

c) Equilibrado de color automático: La imagen anterior tiene unos colores raros debido a que todavía no se ha corregido el balance de blancos (WB).

Como tratamos de reproducir el proceso de revelado que tiene lugar en una cámara vamos a corregir automáticamente el color (como hace la máquina con la selección de AWB o Auto White Balance). Una opción muy sencilla es suponer que la media de toda la imagen es un gris neutro. Obviamente usar AWB siempre será peor que seleccionar nosotros una zona gris conocida (como en el ejercicio 1), ya que la media de los colores de la imagen no tiene porque ser un tono neutro.

Se trata de repetir el ejercicio 1, pero usando la media de cada canal (R, G y B) al completo para formar el vector **neutro**, en vez de usar solamente un trozo de la imagen. Al igual que entonces, a partir de neutro, calcular el vector compensación de color **comp**. [Adjuntad los valores obtenidos para los vectores neutro y comp.](#) Los valores de comp os deben salir del orden de [1.6 0.9 0.8].

Usar los valores obtenidos en **comp** para corregir los tres canales de la imagen anterior. Visualizar la imagen resultante con `pinta_im`. [Adjuntar imagen.](#)

d) Paso a espacio de color sRGB y aplicación de la no linealidad γ .

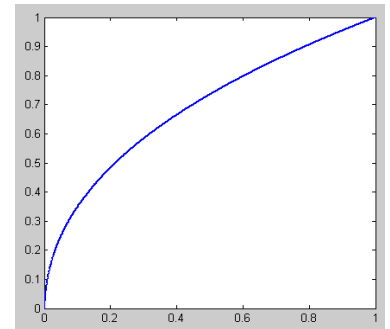
Aunque ya no tiene colores raros, la imagen anterior todavía no es válida: es muy oscura y está en un espacio de color "RAW" muy peculiar, influenciado por varios factores específicos de la cámara (filtros de color, sensor específico usado, etc.). Para esta cámara, el fabricante recomienda la siguiente matriz M para pasar del espacio de color propio al espacio de color sRGB usado por la mayoría de las cámaras como salida:

$$\begin{pmatrix} sR \\ sG \\ sB \end{pmatrix} = \begin{pmatrix} & & \\ & M & \\ & & \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix}_{raw} \quad M = \begin{pmatrix} 1.645 & -0.610 & -0.035 \\ 0.014 & 1.270 & -0.284 \\ 0.006 & -0.207 & 1.201 \end{pmatrix}$$

Usar la función `fc_transform()` del LAB1 para transformar la imagen anterior al nuevo espacio de color sRGB usando la matriz de transformación M dada.

Una vez en el espacio sRGB, poner a 0 o a 1 aquellos valores de la imagen que se sean menores que 0 o mayores que 1. Se dice que son colores que no están dentro del rango de colores reproducibles ("gamut") del espacio de color (sRGB) destino.

El paso al espacio sRGB se completa aplicando una función γ no lineal a los valores de la imagen. Esta no-linearidad es la que se encarga de hacer la imagen más clara y repartir mejor sus valores en la zona en la que el ojo es más sensible. La función a usar mantiene el intervalo $[0,1]$ en $[0,1]$ pero expande la zona de tonos oscuros (niveles bajos) donde el ojo es más sensible. De esta forma la posterior cuantificación a 8 bits por canal será menos perceptible. La función no lineal γ definida para el standard sRGB es:



$$\gamma(x) = \begin{cases} 12.92 \cdot x & \text{si } x < 0.0031308 \\ 1.055 \cdot x^{1/\gamma} - 0.055 & \text{si } x \geq 0.0031308 \end{cases} \quad \text{con } \gamma = 2.4.$$

Aplicar la función anterior a los valores de vuestra imagen (tras la aplicación de la matriz M y posterior limitación entre 0 y 1). Todas estas operaciones serán mucho más rápidas sin bucles, aplicando las condiciones y fórmulas sobre toda la imagen. Se valorará que uséis los menos bucles posibles en vuestro código.

Adjuntad imagen (`pinta_im`) resultante tras aplicar el cambio a sRGB + función γ .

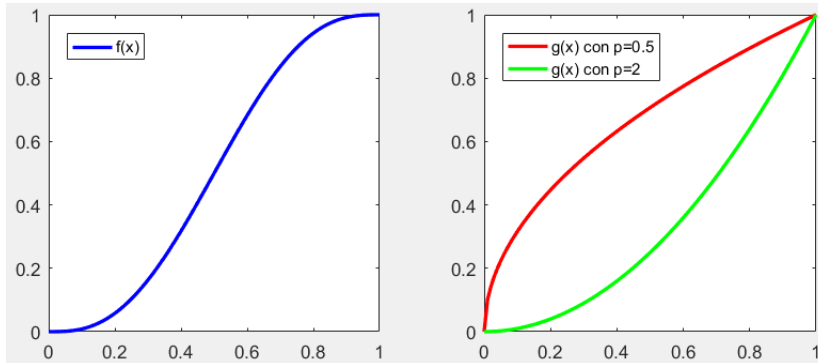
e) Retoques de Brillo / Contraste / Saturación:

La mayoría de las cámaras tienen opciones en el Menú para modificar el aspecto final de la foto alterando aspectos como su brillo (mayor o menor luminosidad), contraste ("ancho" del rango de valores) y saturación (colores más o menos vivos). Para variar estos parámetros es conveniente pasar previamente a alguno de los espacios de color que separan la información de luminancia de la de color. Uno de estos es el espacio HSV (H = Hue, tono de color, S=Saturación, V = luminancia), al que podéis llegar con la función `rgb2hsv()` de MATLAB. Una vez en el nuevo espacio podéis acceder por separado a la luminancia V (3er plano) o a la saturación S (2º plano). Tanto V como S son valores entre 0 y 1, cuya interpretación corresponde a:

- V=Luminancia: versión B/W de la imagen. 0 es negro (valores oscuros) y 1 es blanco (píxeles brillantes). Modificando este canal podemos cambiar el brillo o el contraste de la imagen.

- S=Saturación: S=0 indica un gris neutro (sin un color predominante) y S=1 un color puro saturado (p.e. un rojo o azul puro). Incrementar/disminuir los valores de este canal corresponde a aumentar/bajar la saturación de color.

Vamos a aumentar el contraste y la saturación de la imagen aplicando las siguientes funciones $f()$ y $g()$ a los canales V y S respectivamente:



$$f(x) = (x^3) \cdot (1 - 3(x-1) + 6(x-1)^2)$$

$$g(x) = x^p$$

Estas funciones (figura adjunta) aplicadas al intervalo $[0,1]$ dan valores también entre 0 y 1.

La función $f(x)$ (azul) estira los valores de entrada hacia los extremos 0 y 1, por lo que aplicada al canal V aumenta el contraste de la imagen. El efecto de $g(x)$ cuando se aplica al canal S depende de p . Si $p > 1$ (verde) los valores de S bajan y la saturación de color disminuye. Valores de $p < 1$ (rojo) tienen el efecto contrario.

Aplicar la función $f()$ al canal V y la función $g()$ al canal S probando con varios valores de p que aumenten la saturación de color, eligiendo el que creáis más adecuado. Evitar usar bucles. [Adjuntad el código usado para modificar V y S, indicando el valor de \$p\$ usado. Adjuntad la imagen reconstruida a partir de los canales modificados.](#)

f) Almacenamiento: tras el procesado vamos a reconvertir la imagen double $[0,1]$ a una imagen de bytes $[0,255]$, multiplicándola por 255 y pasándola a bytes usando `uint8()`. Es en este punto donde se lleva a cabo la reducción a 256 niveles (8 bits) por canal. [¿Qué tamaño \(en bytes\) ocupa la imagen en memoria?](#)

Ya solo queda guardar esta imagen en disco. Si usamos un formato sin pérdidas (p.e. TIF) podemos hacer: `imwrite(im, 'foto.tif');` [¿Qué tamaño \(en bytes\) tiene el fichero .tif resultante? ¿Coincide con el tamaño de la imagen en memoria? Justificar.](#)

Para ahorrar espacio en la tarjeta de memoria es común usar formatos comprimidos tipo JPG (con diversas calidades). En Matlab: `imwrite(im, 'foto99.jpg', 'Quality', 99);`

Esta sería la imagen finalmente guardada en la tarjeta de memoria si habéis elegido la opción de JPGs en el Menú. En este caso seleccionaremos también la calidad de la imagen ("Best", "Fine", ...), lo que corresponde a usar diferentes calidades en la compresión JPG. [¿Qué ratio de compresión \$r = \(1 - \text{tamaño_JPG} / \text{tamaño_original}\)\$ \(expresado en %\) se alcanza respecto a la imagen original usando factores de calidad \$Q = 99, 95\$ y \$80\$.](#)

Para este ejercicio, además de la hoja de respuestas entregad el script implementando todos los pasos del revelado digital (en un .rar o similar).

Referencia: para que vayáis viendo si lo vais haciendo bien, os adjunto una copia de imágenes similares a las que os deberían ir saliendo en las diferentes etapas.

