PROCESAMIENTO AUDIOVISUAL

Programa de teoría

1. Adquisición y representación de imágenes.
2. **Procesamiento global de imágenes.**
3. Filtros y transformaciones locales.
4. Transformaciones geométricas.
5. Espacios de color y el dominio frecuencial.
6. Análisis de imágenes.
7. Vídeo y sonido digital.

Tema 2. Procesamiento global de imágenes.

2.1. Tipos de operaciones. Histogramas.

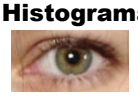
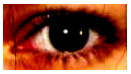
- **Pregunta:** ¿Cuál es la base teórica del procesamiento de imágenes? ¿Qué operaciones aplicar?
- **Recordatorio:** ¡una imagen digital no es más que una matriz, o array bidimensional, de números!

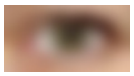
90	67	68	75	78
92	87	73	78	82
63	102	89	76	98
45	83	109	80	130
39	69	92	115	154

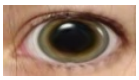
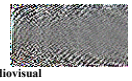
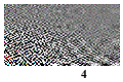
→ Podemos aplicar las mismas operaciones que sobre cualquier número: sumar, restar, multiplicar, dividir, aplicar and, or, máximo, mínimo, integrales, derivadas...

2.1. Tipos de operaciones. Histogramas.

- **Principales tipos de procesamientos de imágenes:**
  - **Operaciones de procesamiento global:** cada píxel es tratado de forma independiente, ya sea con una o con varias imágenes.
  - **Filtros y convoluciones:** se considera la vecindad local de los píxeles.
  - **Transformaciones geométricas:** se modifica el tamaño y forma de las matrices.
  - **Transformaciones lineales:** Fourier, wavelets, etc.

2.1. Tipos de operaciones. Histogramas.

- **Operaciones de procesamiento global:**
  - **Aritméticas**: sumar, restar, multiplicar, máximo, etc.
    - Unarias: una sola imagen y un valor constante.
    - Binarias: con dos imágenes.
  - **Booleanas**: and, or, not, etc.
    - Unarias: una sola imagen y una constante.
    - Binarias: con dos imágenes.
  - **Otras** transformaciones generales:
    - Transformaciones de histograma.
    - Transformaciones de color.
    - Binarización, etc.
- Cada operación tendrá un **significado**, **utilidad** y **aplicaciones** específicos. +

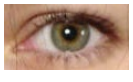
2.1. Tipos de operaciones. Histogramas.

- Supongamos una imagen de **entrada A** y una imagen **resultado R**.
- Una operación global (píxel a píxel) se puede expresar como **una función**:

$R(x,y) := f(A(x,y))$ → El valor del píxel resultante es función de (y sólo de) el píxel correspondiente de entrada.

- **Ejemplo. Invertir.** $R(x,y) := 255 - A(x,y)$

Imagen A → Imagen R



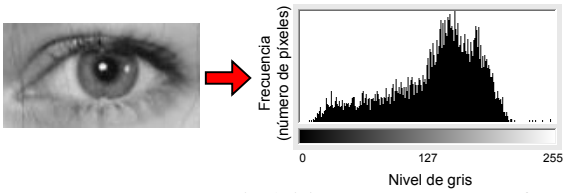

2.1. Tipos de operaciones. Histogramas.

$R(x,y) := f(A(x,y)), \ \forall (x,y)$

- **Comparar con:**
  - **Filtros y convoluciones:** el valor de un píxel depende de la vecindad local de ese píxel:
    $R(x,y) := f(A(x-k,y-k), ..., A(x,y), ..., A(x+k,y+k))$
  - **Transformaciones geométricas:** el valor de un píxel depende de píxeles situados en otras posiciones:
    $R(x,y) := A(f_1(x,y), f_2(x,y))$
  - **Transformaciones lineales:** el valor de un píxel puede depender de todos los píxeles de la imagen:
    $R(x,y) := f(A, x, y)$

2.1. Tipos de operaciones. Histogramas.

- Para comprender el significado de muchas transformaciones y saber cuál conviene aplicar se usan histogramas.
- ¿Qué es un histograma? → Repasar estadística...
- Un **histograma** representa gráficamente una distribución de frecuencias.
- **Histograma de una imagen:** representa las frecuencias de los diferentes valores de gris en la imagen.



2.1. Tipos de operaciones. Histogramas.

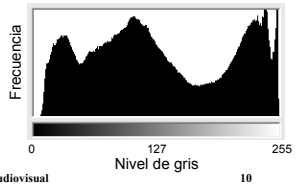
- **Algoritmo.** Cálculo de un histograma.

- **Entrada.** A: imagen de ancho x alto
- **Salida.** Histograma: array [0,...,255] de entero
- **Algoritmo:**

```
Histograma[]:= 0
para y:= 0, ..., alto-1 hacer
    para x:= 0, ..., ancho-1 hacer
        Histograma[A(x,y)]:= Histograma[A(x,y)]+1
```

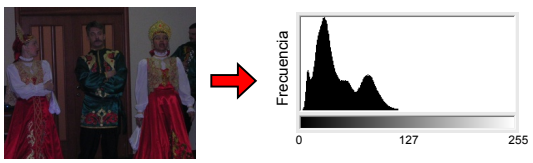

2.1. Tipos de operaciones. Histogramas.

- Los histogramas son una herramienta importante en **análisis de imágenes**: ¿es buena la calidad de una imagen?, ¿sobra luz?, ¿falta contraste?
- Ayudan a decidir cuál es el procesamiento más adecuado para **mejorar la calidad** de una imagen...
  - Tanto **cualitativamente** (qué operación aplicar),
  - Como **cuantitativamente** (en qué cantidad).
- En principio, una buena imagen debe producir un **histograma** más o menos **uniforme** y repartido en todo el rango de valores.

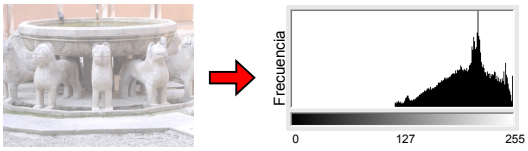


2.1. Tipos de operaciones. Histogramas.

- **Ejemplo 1.** La imagen es muy oscura. Falta luz.

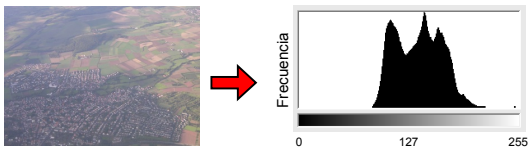


- **Ejemplo 2.** La imagen es muy clara. Sobra brillo.

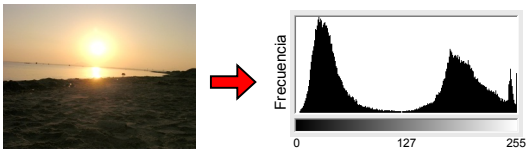


2.1. Tipos de operaciones. Histogramas.

- **Ejemplo 3.** La imagen tiene poco contraste.

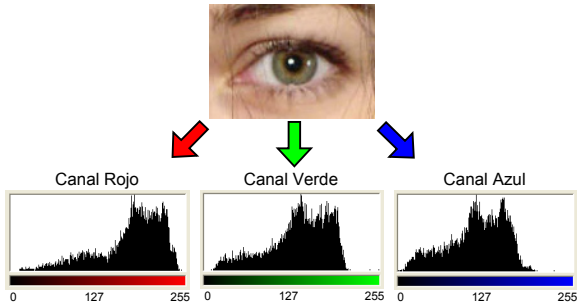


- **Ejemplo 4.** Hay mucho contraste, pocos medios tonos.



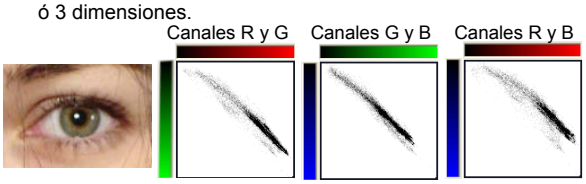
2.1. Tipos de operaciones. Histogramas.

- **Histogramas de color.** En imágenes multicanal podemos obtener un histograma de cada canal por separado.



2.1. Tipos de operaciones. Histogramas.

- O, también, podemos calcular histogramas conjuntos, en 2 ó 3 dimensiones.



- Estos histogramas aportan información sobre los rangos de **colores más frecuentes** en la imagen.
- En teoría, el histograma es de 256x256 celdas (*bins*).
- Pero, para obtener buenos resultados, mejor usar un **número reducido de celdas**. Por ejemplo 64x64 ó 32x32.

2.1. Tipos de operaciones. Histogramas.

- Uso de histogramas para mejorar la calidad de las imágenes.
- **Ejemplo.** El histograma indica tonos muy oscuros.

- **Solución.** Aplicar un operador que "estire" el histograma.

2.2. Operaciones elementales con píxeles.

- **A**: imagen de entrada.
- **R**: imagen resultante (del mismo tamaño que A).

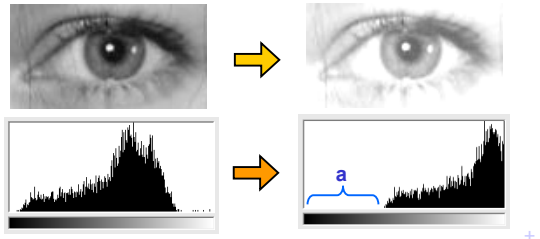
Operaciones unarias:

- Sumar una constante: R(x, y):= A(x, y) + a
- Restar una constante: R(x, y):= A(x, y) - a
- Multiplicar por una constante: R(x, y):= b·A(x, y)
- Dividir por una constante: R(x, y):= A(x, y)/b
- Transformación lineal genérica: R(x, y):= bA(x, y)+a
- Transformación de gama: R(x, y):= A(x, y)^c
- Cualquier función **N**→**N**: R(x, y):= f(A(x,y))

2.2. Operaciones elementales con píxeles.

Sumar una constante: R(x, y):= A(x, y) + a

- **Significado:** incrementar el brillo de la imagen en la cantidad indicada en **a**.
- El histograma se desplaza a la derecha en **a** píxeles.



2.2. Operaciones elementales con píxeles.

- **Ojo:** la suma puede ser mayor que 255...
- La operación debería comprobar el overflow:
  si A(x, y) + a > 255 **entonces** R(x, y):= 255
  sino R(x, y):= A(x, y) + a
- Esto se debe hacer también en las demás operaciones, comprobando si el valor es <0 ó >255.
- Coloquialmente, un píxel "por encima" de 255 o por debajo de 0 se dice que está **saturado**.
- La saturación supone una pérdida de información.

Ejemplo de imagen muy saturada

2.2. Operaciones elementales con píxeles.

- En imágenes en color, la suma se realiza sobre los tres canales (R, G y B) y con el **mismo valor**.

R(x, y).R:= A(x, y).R + a R(x, y).G:= A(x, y).G + a

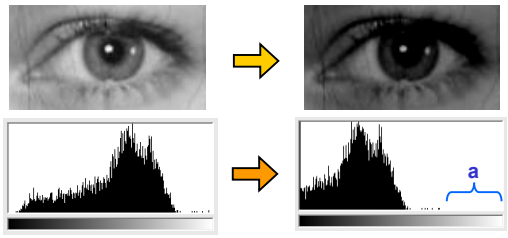
R(x, y).B:= A(x, y).B + a

- ¿Qué ocurre si se suma un valor distinto a cada canal?

2.2. Operaciones elementales con píxeles.

Restar una constante: R(x, y):= A(x, y) - a

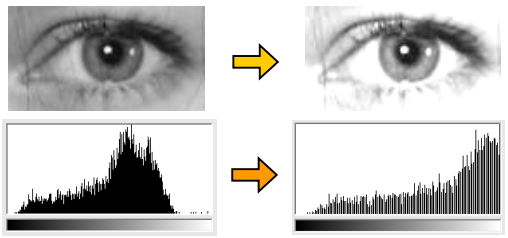
- **Significado:** decrementar el brillo de la imagen en la cantidad indicada en **a**.
- El histograma se desplaza a la izquierda en **a** píxeles.



2.2. Operaciones elementales con píxeles.

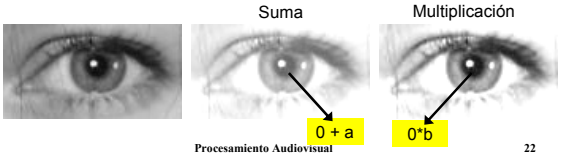
Multiplicar por una constante: R(x, y):= b·A(x, y)

- **Significado:** aumentar la intensidad de la imagen en **b**.
- El histograma se "estira" hacia la derecha.



2.2. Operaciones elementales con píxeles.

- Tanto en la suma como en la multiplicación, se aumenta el nivel de gris de los píxeles, pero de forma distinta.
  - **En la suma**, el parámetro **a** (entero) indica el número de niveles de gris a aumentar: de -255 a 255.
  - **En el producto**, el parámetro **b** (real) indica el factor a multiplicar.
    - b=1 → Ningún cambio
    - b=2 → Se duplica el valor de gris. Los píx. >127 se saturan.
    - b=0,5 → Se "encoge" a la mitad el histograma.

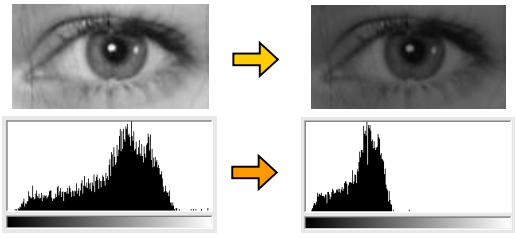


2.2. Operaciones elementales con píxeles.

Dividir por una constante: R(x, y):= A(x, y) / b

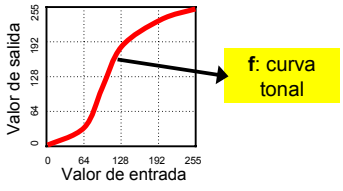
= Multiplicar por 1/b ... ¡obviamente!

- El histograma se "encoge".



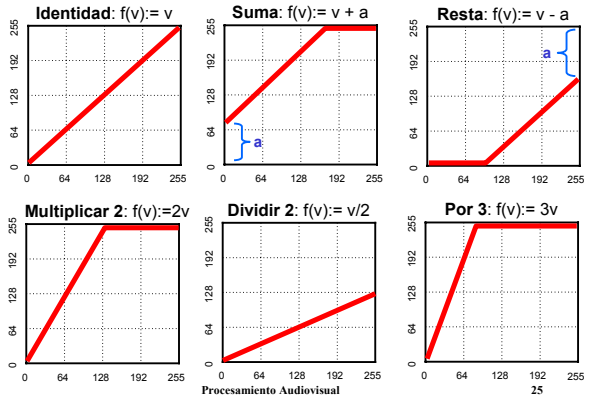
2.3. Transformaciones del histograma.

- Las transformaciones elementales se pueden ver como **funciones f: N → N**.
- **Interpretación:** para cada valor de gris de entrada hay un valor de salida.



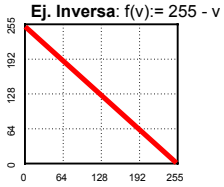
- Se puede usar cualquier función **f**.
- La transformación hace que se modifique el histograma.

2.3. Transformaciones del histograma.



2.3. Transformaciones del histograma.

- En general, podemos definir una **transformación lineal genérica** de la forma:
  f(v):= b·v + a

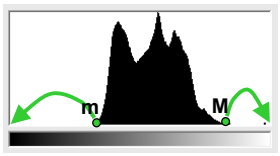


- Pero la transformación también puede ser **no lineal**: cuadrática, polinomial, exponencial, logarítmica, escalonada, etc.
- ¿Cómo decidir cuál es la transformación más adecuada? → Usar el histograma.

2.3. Transformaciones del histograma.

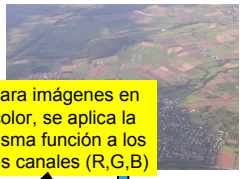
- Normalmente, interesa "estirar" el histograma, para conseguir que aparezca todo el rango de valores.
- **Idea:** definir una transformación lineal tal que el histograma resultante vaya de 0 a 255.
- **Ajuste lineal** o **estiramiento** ***(stretch)*** del histograma:
  - Buscar el valor mínimo del histograma: **m**
  - Buscar el valor máximo: **M**
  - f(v):= (v-m)*255/(M-m)

Nota: Esto es una simple regla de 3



2.3. Transformaciones del histograma.

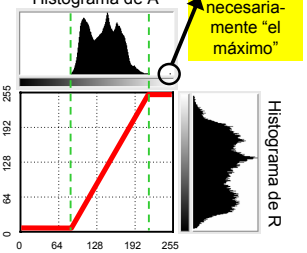
A



Para imágenes en color, se aplica la misma función a los tres canales (R,G,B)

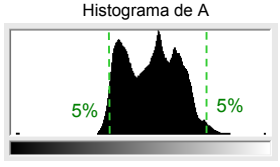
R

- **Ejemplo.** m= 86, M= 214
  R(x,y):= (A(x,y)-86)*1,99



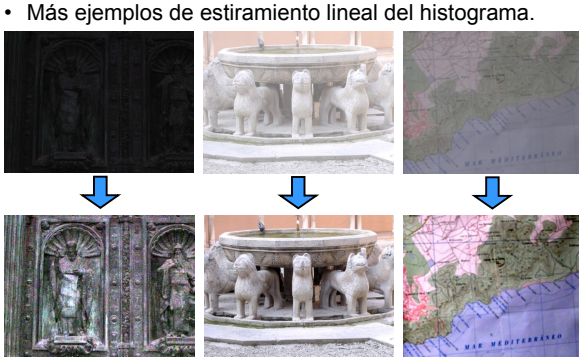
2.3. Transformaciones del histograma.

- **Cuidado:** un simple píxel con valor muy alto o muy bajo puede hacer que el ajuste del histograma sea muy malo.
- Por ejemplo, si hay un píxel con valor 0 y otro con 255, la transformación sería la identidad (la imagen no cambia).
- **Solución:** en lugar de mínimo y máximo, ajustar usando dos percentiles del histograma (p. ej. 10%-90%, ó 5%-95%).



2.3. Transformaciones del histograma.

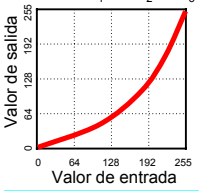
- Más ejemplos de estiramiento lineal del histograma.



2.3. Transformaciones del histograma.

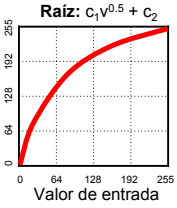
- La **transformación de histograma** puede tomar cualquier forma (no necesariamente lineal).
- **Ejemplos.**

Parábola: $c_1v^2 + c_2v + c_3$



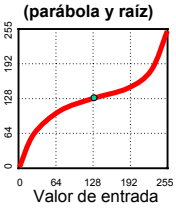
Resultado: oscurecer los medios tonos.

Raíz: $c_1v^{0.5} + c_2$



Resultado: aclarar los medios tonos.

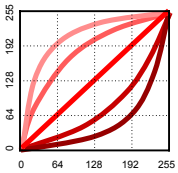
Dos trozos de curva (parábola y raíz)



Resultado: aclarar tonos oscuros y oscurecer los claros.

2.3. Transformaciones del histograma.

- Elevar a 2, elevar a 1/2, ...
- Se define la **transformación de gama** como:
  $f(v) := 255 \cdot (v/255)^{1/GAMA}$



Gama 0,5 Gama 0,75 Gama 1 Gama 2 Gama 4

2.3. Transformaciones del histograma.

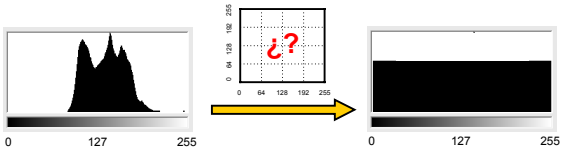
- La diferencia entre diferentes dispositivos (televisores, cámaras, escáneres) se modela con una **transformación de gama**.
- Si el comportamiento del dispositivo fuera perfectamente lineal, Gama = 1.

Blanco Negro

- ¿Dónde está el 50% de gris? ¿Es la escala lineal?
- ¿Dónde estaría si tomáramos una foto?

2.3. Transformaciones del histograma.

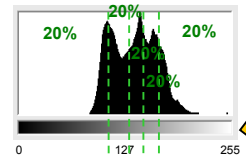
- Otra transformación habitual es la ecualización del histograma (del latín *aequalis* = igual).
- **Ecualización del histograma:** es una transformación definida de forma que el histograma resultante se reparte uniformemente en todo el rango de grises.



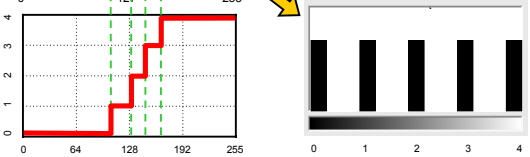
- En este caso se usa una función escalonada:
  f: array [0..255] de byte

2.3. Transformaciones del histograma.

- ¿Cómo definir **f** para conseguir la ecualización?
- **Idea:** suponer que a la salida hay 5 niveles de gris.



para todo píxel (x,y) de R **hacer**
R(x,y):= f[A(x,y)]



2.3. Transformaciones del histograma.

- **Algoritmo.** Cálculo de la función de ecualización del histograma.
- **Entrada.** Histograma: array [0,...,255] de entero
  np: entero (número total de píxeles = $m_x * m_y$)
- **Salida.** f: array [0,...,255] de byte

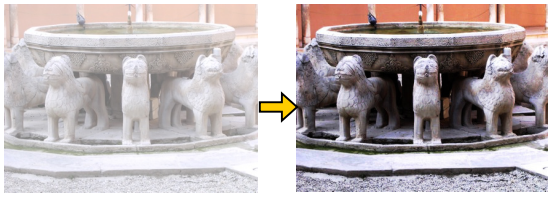
- **Algoritmo:**

```
f[0]:= 0
acumulado:= Histograma[0]
para i:= 1, ..., 254 hacer
    f[i]:= acumulado*255/np
    acumulado:= acumulado + Histograma[i]
finpara
f[255]:= 255
```

La función de ecualización es la **integral del histograma**, escalada por el factor 255/**np**.

2.3. Transformaciones del histograma.

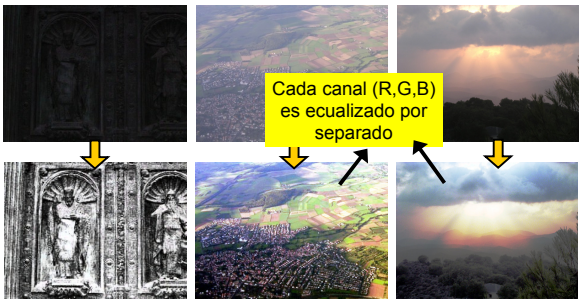
Imagen de entrada (A) Imagen ecualizada (R)



Histograma de A Función f Histograma de R

2.3. Transformaciones del histograma.

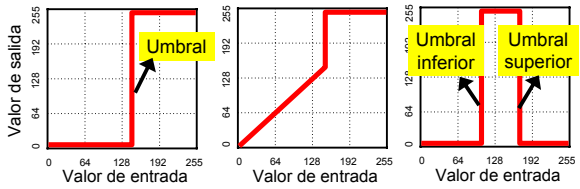
- **Ejemplos.** Ecualización del histograma.



- Cuidado, en algunos casos los resultados pueden ser *artificiosos*.

2.3. Transformaciones del histograma.

- **Umbralización de imágenes.** En algunas aplicaciones puede ser interesante convertir la imagen en binaria, o recortar cierto rango de valores.
- Las funciones tienen las siguientes formas:



Umbralizar la imagen con valor cte. | **Cortar** un rango y mantener el resto | **Seleccionar** un rango

2.3. Transformaciones del histograma.

- Las funciones serán del estilo:
  f(v):= **si** v > umbral1 **entonces** g(v)
  sino h(v)
- **Transformación de binarización** (saturar a 0 ó 255).
  f(v):= **si** v < umbral **entonces** 0 **sino** 255
- **Ejemplo 1**. La binarización se suele aplicar en OCR.

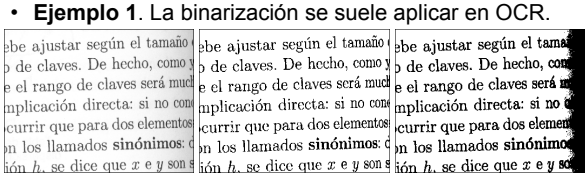


Imagen de entrada (256 grises) | Umbral = 160 | Umbral = 215

2.3. Transformaciones del histograma.

- **Ejemplo 2**. Segmentación de objetos.

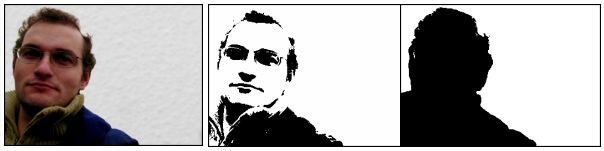


Imagen de entrada | Umbralizar, u = 42 | Umbralizar, u = 180

- La separación del objeto del fondo se llama **segmentación**.
- La umbralización se puede usar para segmentar...
- ... aunque por sí sola no suele funcionar muy bien.

Cortar rango (192, 255)

2.3. Transformaciones del histograma.

Conclusiones:

- Una **transformación elemental** se puede ver desde distintas perspectivas:
  - Como una **función** unidimensional: **f: N → N**
  - Como una **curva tonal**.
  - Como una **modificación** del **histograma**.
- La característica fundamental es que cada píxel se trata **independientemente** de los demás.
- Los **histogramas** son útiles para encontrar la transformación adecuada.
- En imágenes RGB, aplicamos **la misma operación** a los 3 canales para que se mantenga el color.

2.4. Combinación de imágenes.

- **Combinación de imágenes:** utilizar dos o más imágenes de entrada para producir una imagen de salida.
- **Entrada:** imágenes A y B.
- **Salida:** imagen R.

R(x, y):= f(A(x,y), B(x,y))

El valor del píxel resultante es función de los píxeles de A y B en la misma posición

En principio, todas las imágenes deben ser del mismo tamaño

- Posibles operaciones de combinación:
  - **Booleanas**: and, or, xor, not
  - **Aritméticas**: suma, resta, producto/división, media
  - **Relacionales**: máximo, mínimo

2.4. Combinación de imágenes.

- **Operadores booleanos:**
  - R(x, y):= A(x,y) AND B(x,y)
  - R(x, y):= A(x,y) OR B(x,y)
  - R(x, y):= A(x,y) XOR B(x,y)
  - R(x, y):= NOT A(x,y) AND B(x,y)
  - R(x, y):= A(x,y) OR NOT B(x,y)
  - ...
- Estos operadores tienen sentido cuando al menos una de las imágenes es binaria.
  - Negro (0) = FALSE
  - Blanco (1 ó 255) = TRUE

2.4. Combinación de imágenes.

- **Ejemplos**. Operadores booleanos.

Imagen de entrada **A**

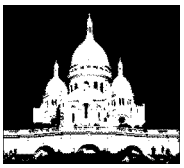
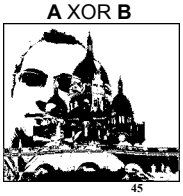


Imagen de entrada **B**

A AND **B**

A OR **B**

A XOR **B**



2.4. Combinación de imágenes.

- En imágenes no binarias no tienen mucho sentido... ¿Cómo se interpretan?

A_{Gris} AND B_{Gris}

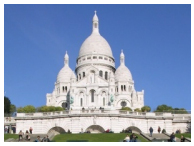
A_{RGB} AND B_{RGB}

- Las operaciones binarias aparecen en análisis de imágenes, y también para trabajar con **máscaras y recortes** de objetos.

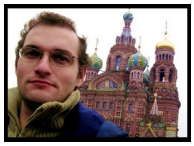
2.4. Combinación de imágenes.

- **Imágenes de entrada.**

A



B



C

- ¿Cómo conseguir el montaje de la página anterior?
- R:= (B AND NOT C) OR (A AND C)

2.4. Combinación de imágenes.

1. T1:= B AND NOT C

B

NOT C

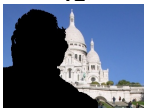
T1

2. T2:= A AND C

A

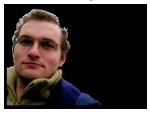
C

T2



3. R:= T1 OR T2

T1



T2

R

2.4. Combinación de imágenes.

- La imagen binaria (**C**) se suele denominar **máscara**.
- La máscara permite segmentar el objeto de interés.

C

R

Cuestiones:

- ¿Cómo crear la máscara de forma automática?
- La zona del pelo no se mezcla bien con el fondo. ¿Cómo evitar este problema?

2.4. Combinación de imágenes.

- **Operaciones aritméticas:**
  - R(x, y):= A(x, y) + B(x, y)
  - R(x, y):= A(x, y) - B(x, y)
  - R(x, y):= (A(x, y) + B(x, y))/2
  - R(x, y):= a·A(x, y) + (1-a)·B(x, y)
  - R(x, y):= A(x, y)·B(x, y)·c

- Se usan en generación y análisis de imágenes.
- Cuidado con los problemas de **saturación**.
- En imágenes binarias son equivalentes (en su mayoría) a los operadores booleanos.

2.4. Combinación de imágenes.

Sumar dos imágenes: R(x, y):= A(x, y) + B(x, y)

- **Significado:** mezclar las dos imágenes.

A

B

R

- **Ojo:** [0..255] + [0..255] = [0..510]

2.4. Combinación de imágenes.

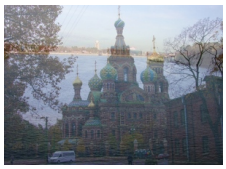
- Para evitar la saturación se puede usar la media.

Media de 2 imágenes: R(x, y):= (A(x,y)+B(x,y))/2

A

B

R



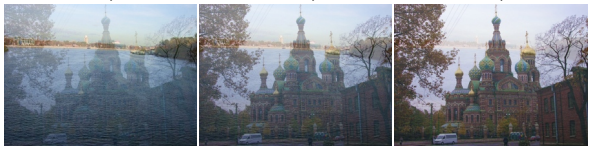
- **Significado:** las imágenes son semitransparentes (al 50%).

2.4. Combinación de imágenes.

- De forma similar, se puede definir la **media ponderada**.

Media ponderada: R(x,y):= a·A(x,y) + (1-a)·B(x,y)

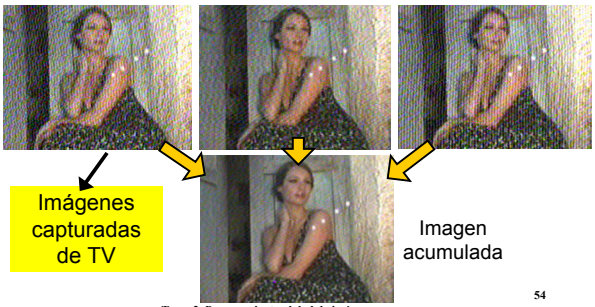
a = 0,25 a = 0,5 a = 0,75



- La media ponderada se puede usar para crear una **transición suave** entre imágenes (o vídeos).

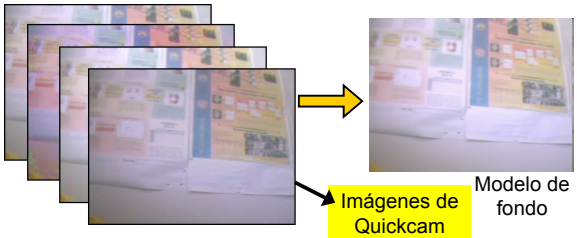
2.4. Combinación de imágenes.

- La media de imágenes se puede usar para **acumular imágenes** de un vídeo.
- **Ejemplo 1.** Combinar imágenes con mucho ruido de una escena, para obtener una mezcla con menos ruido.



2.4. Combinación de imágenes.

- **Ejemplo 2.** Crear un modelo de fondo de una escena, acumulando varias imágenes.

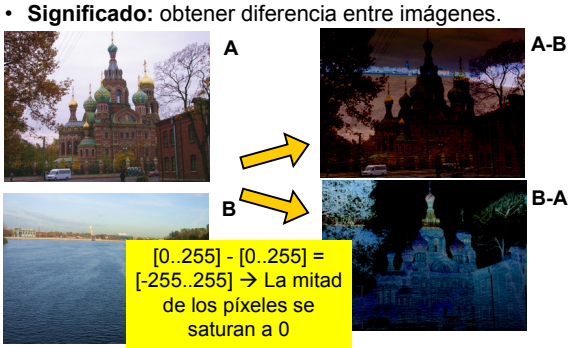


- **Idea:** si además de la media en cada píxel calculamos también la varianza, podríamos tener un modelo gaussiano del fondo (N(μ,σ)).

2.4. Combinación de imágenes.

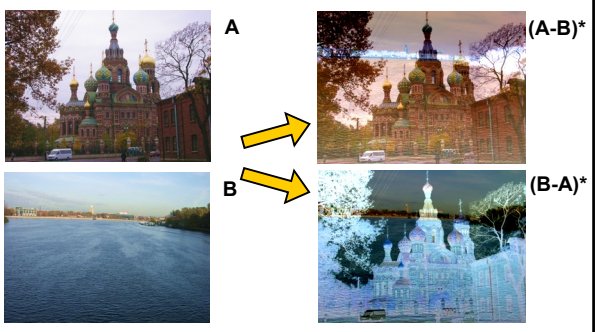
Restar dos imágenes: R(x, y):= A(x, y) - B(x, y)

- **Significado:** obtener diferencia entre imágenes.



2.4. Combinación de imágenes.

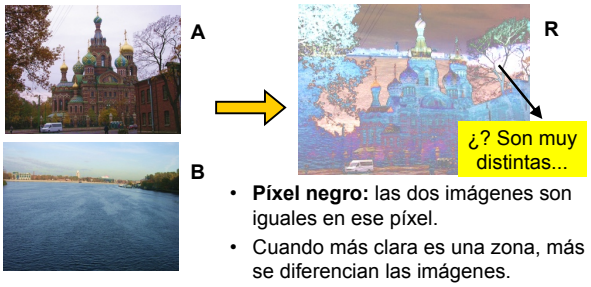
Restar dos imágenes, manteniendo el rango de salida: R(x, y):= (A(x, y) - B(x, y))/2 + 128



2.4. Combinación de imágenes.

- Muchas veces lo que interesa es conocer la diferencia entre las imágenes. → **Solución:** tomar valor absoluto de la resta.

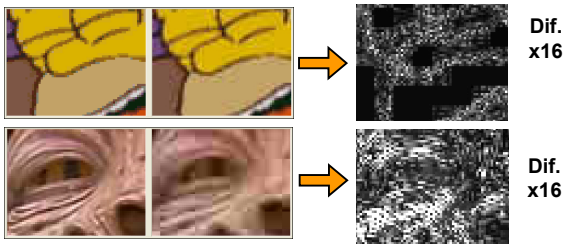
Diferencia: R(x, y):= abs(A(x, y) - B(x, y))



- **Píxel negro:** las dos imágenes son iguales en ese píxel.
- Cuando más clara es una zona, más se diferencian las imágenes.

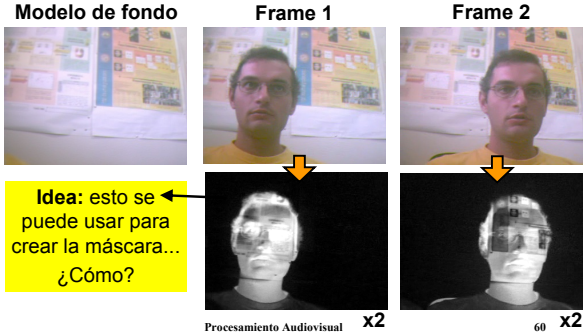
2.4. Combinación de imágenes.

- **Aplicaciones de la diferencia:** encontrar variaciones entre imágenes que, en principio, deberían ser parecidas.
- **Ejemplo 1.** Analizar la pérdida de información al comprimir una imagen. Por ejemplo, con JPEG.



2.4. Combinación de imágenes.

- **Ejemplo 2.** Segmentación del fondo de una escena.
- Tenemos un fondo (imagen media) y una nueva imagen.



2.4. Combinación de imágenes.

- **Proceso.**

1. Obtener el modelo de fondo **M**.
2. Para cada imagen **A** del vídeo.
3. Calcular la diferencia: **D** = abs(M-A).
4. Umbralizar la imagen con un valor adecuado. **U** = umbralizar(D, x).
5. Sea **F** el nuevo fondo.
6. **R**:= (F AND NOT U) OR (A AND U)

¿Cómo arreglar eso?

2.4. Combinación de imágenes.

- **Ejemplo 3.** Detección de movimiento en vídeo.
- Dada una secuencia de vídeo, queremos saber si se ha producido alguna modificación, y en qué zonas de la imagen ("encuentra las 7 diferencias").

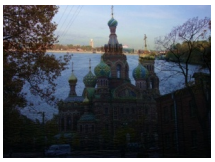
Frame 1 **Frame 2** **Diferencia x2**

- ¿Qué objetos se han movido y en qué dirección?

2.4. Combinación de imágenes.

Producto imágenes: R(x, y):= A(x, y)·B(x, y)/255


A


A·B


B

- Necesario escalar el resultado (dividir por 255).
- Efecto de **mezcla**, similar a la suma, pero conceptualmente más próximo a un AND...

2.4. Combinación de imágenes.

División imágenes: R(x, y):= 255·A(x, y)/B(x, y)


A

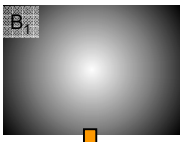

A/B


B

- También es necesario escalar el resultado (multiplicar por 255).
- ¿Cuál es interpretación del resultado?

2.4. Combinación de imágenes.

- **Ejemplo 1.** Realizar una transformación de intensidad distinta para cada píxel.



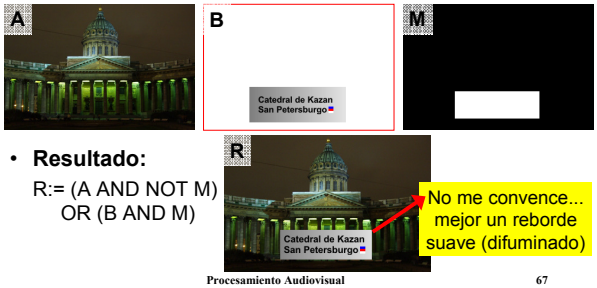
2.4. Combinación de imágenes.

- Estos mismos tipos de imágenes se pueden usar para hacer sumas, restas, divisiones, etc.
- **Ejemplo.** R(x, y):= A(x, y)·B(x, y)/128
  - Si B(x, y) = 128 el píxel de A no cambia.
  - Si B(x, y) < 128 el píxel se oscurece.
  - Si B(x, y) > 128 el píxel se aclara.

- El producto es también la base en la idea de **máscara** o **selección difusa**.
- **Idea:** una imagen se compone de distintos elementos o capas, que tienen definido cierto nivel de transparencia.

2.4. Combinación de imágenes.

- **Ejemplo 2.** Mezcla y combinación de imágenes. Queremos combinar dos imágenes, por ejemplo, para poner una etiqueta descriptiva en una foto. Una imagen binaria sirve de **máscara**: 0 = fondo, 1 = etiqueta.

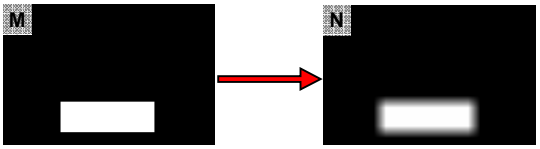


- **Resultado:**
  R:= (A AND NOT M) OR (B AND M)

No me convence... mejor un reborde suave (difuminado)

2.4. Combinación de imágenes.

- **Solución.** Usar una máscara "suave", una imagen en gris: 0 = transparente, 255 = opaco. Combinar: sumas y productos.



- **Resultado:**
  R:= A·(255-N)/255 + B·N/255

Producto de imágenes

2.4. Combinación de imágenes.

- **Indicaciones sobre el ejemplo 2.**
  - La "mascara suave" es la idea del **canal alfa**.
  - RGB → RGBA, donde el canal A indica el **grado de opacidad** de un píxel (0= transparente, 255= opaco).
  - **Uso:** definimos imágenes, con sus canales alfa, y las componemos poniendo unas sobre otras.
  - La **composición de imágenes** con canal alfa es básicamente una media ponderada como hemos visto.
  - En el modo binario, muchas herramientas incorporan las ideas de **máscara**, **selección**, **región de interés** (cuando es rectangular) o **canal de interés** (en multicanal).
  - No necesitamos trabajar con **operaciones booleanas**, aunque implícitamente es lo que hay subyacente.

2.4. Combinación de imágenes.

Otras operaciones no lineales

- **Mínimo de 2 imágenes.** R(x, y):= min(A(x, y), B(x, y))

- **Máximo de 2 imágenes.** R(x, y):= max(A(x, y), B(x, y))

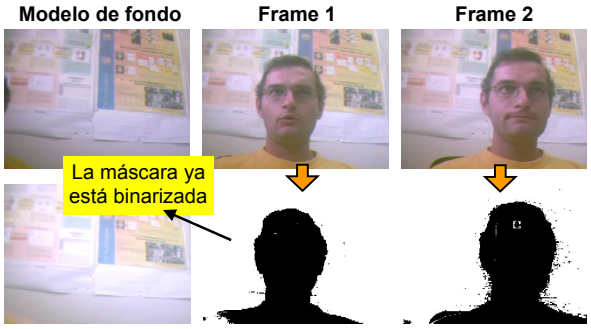
2.4. Combinación de imágenes.

- **Ejemplo.** Una alternativa para crear modelos de fondo es usar máximos y mínimos. En lugar de tener media y varianza, tenemos **máximo** y **mínimo** del **fondo** en cada píxel.

- Dada una imagen nueva, para cada píxel, comprobar si su valor está **entre el máximo y el mínimo**. Si lo está: fondo; si no lo está: objeto.

2.4. Combinación de imágenes.

- Con esto tenemos otra forma de hacer la segmentación de los objetos.



2.4. Combinación de imágenes.

Conclusiones:

- Operaciones de **combinación**: a partir de dos o más imágenes obtener una nueva imagen.
- La operación a aplicar depende de lo que queramos conseguir.
- Operaciones **booleanas**: útiles para trabajar con máscaras de objetos.
- Operaciones **aritméticas**: útiles en vídeo, modelos acumulados, detección de movimiento, transparencias difusas, etc.
- En general, cualquier tipo de operación es posible, ya sean lineales o no lineales.

2.5. Transformaciones de color.

- En los puntos anteriores la transformación era la misma para todos los canales (R, G y B).
- Si es **distinta**, hablamos de **transformación de color**:

  $R(x, y).R := f_1(A(x,y).R, A(x,y).G, A(x,y).B)$

  $R(x, y).G := f_2(A(x,y).R, A(x,y).G, A(x,y).B)$

  $R(x, y).B := f_3(A(x,y).R, A(x,y).G, A(x,y).B)$
- **Posibilidades:**
  - Aplicar **las mismas** transformaciones que antes (suma, producto, ajuste de histograma, etc.), pero con distintos parámetros para cada canal.
  - Transformaciones basadas en **modelos de color**. Cambiar el modelo de color (RGB, HSV, HLS, XYZ, YUV, etc.) y aplicar la función en ese modelo.

2.5. Transformaciones de color.

Conversión color → escala de grises

- **Conversión sencilla:**

  $R(x, y) := (A(x,y).R + A(x,y).G + A(x,y).B)/3$

- **Conversión precisa:**

  $R(x, y) := 0.21A(x,y).R + 0.72A(x,y).G + 0.07A(x,y).B$

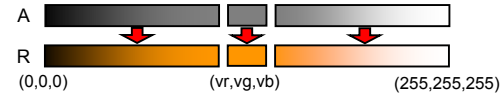
Pero, ¿de dónde salen esos pesos?

Imagen de entrada | Grises (media) | Grises (precisa)

2.5. Transformaciones de color.

Transformación escala de grises → escala de color

- **Idea:** dada un imagen en gris, producir una imagen en escala de cierto color dado.
- Sea **A** una imagen en grises y un color objetivo **(vr, vg, vb)**. La escala se puede descomponer en dos partes:



- **Transformación** (obviamos (x,y)):

  si A<128 **entonces**

  R.R:= vr·A/128; R.G:= vg·A/128; R.B:= vb·A/128

  sino

  R.R:= vr+(255-vr)(A-128)/128; R.G:= vg+(255-vg)(A-128)/128

  R.B:= vb+(255-vb)(A-128)/128

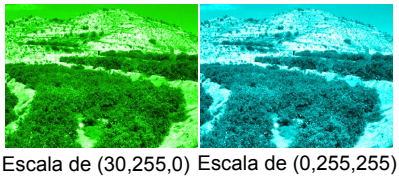
  finsi

2.5. Transformaciones de color.

- **Ejemplo.** Transformación a sepia.

Imagen de entrada | Escala de grises | Escala de sepias

- ¿Cómo conseguir que el punto intermedio sea un valor cualquiera (distinto de 128)?

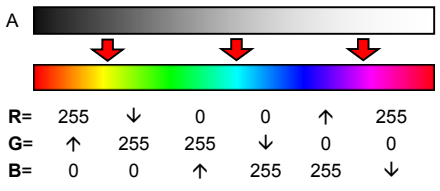


Escala de (30,255,0) | Escala de (0,255,255)

2.5. Transformaciones de color.

Transformación de color falso

- Es una transformación de la misma familia, cuyo objetivo es hacer más visibles las **pequeñas variaciones** del nivel de gris.
- Se define una paleta de salida adecuada y una transformación de cada valor de gris en la paleta.



2.5. Transformaciones de color.

- **Ejemplo.** Transformación de color falso.
  Las transformaciones de este tipo son comunes en imágenes **médicas** y de **satélite**.

  En estas aplicaciones, la profundidad del canal puede ser fácilmente mayor que 1 byte. Al usar sólo 256 grises se pierde información.

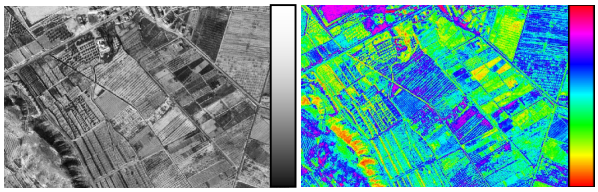


Imagen de entrada Imagen con color falso

2.5. Transformaciones de color.

Transformaciones de agregar color (colorear)

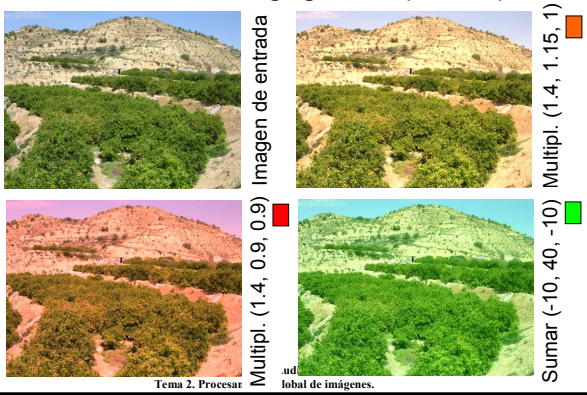
- **Idea:** usar las operaciones de suma, resta y producto, pero con una constante distinta por cada canal.

  R.R:= vr+A.R; R.G:= vg+A.G; R.B:= vb+A.B

  R.R:= fr·A.R; R.G:= fg·A.G; R.B:= fb·A.B
- **(vr, vg, vb)** y **(fr, fg, fb)** indican el tono de color que se da a la imagen.

2.5. Transformaciones de color.

Transformaciones de agregar color (colorear)



2.5. Transformaciones de color.

- Estas transformaciones están relacionadas con el **balance de blancos**.
- Las salidas de los fotodetectores de cada canal (R,G,B) deberían ser acordes a la apreciación subjetiva del color por parte del humano.
- Esto implica multiplicar cada canal por un factor adecuado.
- **Cuestión:** ¿qué imagen tiene los colores más realistas?
- **Ejemplos:**
  - Priorizar rojos (medio): fr= 1.2, fg= 0.9, fb= 0.9
  - Priorizar verdes (mucho): fr= 0.8, fg= 1.6, fb= 0.8
  - Priorizar amarillos (poco): fr= 1.1, fg= 1.1, fb= 0.8

2.5. Transformaciones de color.

- También es posible **mezclar** y **cambiar** los canales, con transformaciones como las siguientes.

2.5. Transformaciones de color.

- Finalmente, recordar que las operaciones de ajuste y ecualización del histograma se pueden aplicar **conjuntamente** (usando el histograma de gris) o por **separado** (usando el histograma de cada canal).
- La diferencia es que mientras el primero **mantiene** los colores (cambia la intensidad) el segundo no los mantiene.

Imagen de entrada Ajuste conjunto Ajuste separado

2.5. Transformaciones de color.

Conclusiones:

- Las transformaciones globales se pueden realizar **igual** en todos los canales o con valores **distintos**.
- En el primer caso, habrá un cambio en la **intensidad**. En el segundo, puede haber también un **cambio de color**.
- **Balance de blancos:** compensar los canales para obtener los colores más realistas posibles.
- Veremos más cuestiones relacionadas con el color cuando estudiemos **espacios de color**.

2. Procesamiento global de imágenes.

Conclusiones:

- **Procesamiento global:** el valor de un píxel de salida depende del píxel (o píxeles) correspondientes de la imagen de entrada.
- Operaciones aritméticas, lógicas, etc.
- **Distintas aplicaciones:** mejora del histograma, reducción de ruido, composición de imágenes, ajuste del color, etc.
- Normalmente no aparecen solas, sino combinadas con otros tipos de operaciones.

Anexo A.2.
Procesamiento global en OpenCV.

- Operaciones unarias
- Operaciones binarias
- Operaciones con histogramas
- Ejercicios

A.2. Procesamiento global en OpenCV.

Operaciones de procesamiento global:

- **Tipos**: unarias, binarias, histogramas.
  void cvUnaria (const CvArr* A, ..., CvArr* B, ...)
  void cvBinaria (const CvArr* A, const CvArr* B,.., CvArr* C, ...)

Recordar: un CvArr puede ser un IplImage

- Las funciones reciben una (unarias) o dos (binarias) imágenes de entrada, y producen una de salida. Todas ellas **deben estar creadas**, también la imagen de salida.
- La mayoría de las funciones admite **modo inplace**: una imagen de entrada se puede usar también para guardar el resultado.
- Muchas de las operaciones trabajan con **ROI**, **COI** y **mask** → No operan sobre toda la imagen sino sólo sobre una **parte** concreta (rectángulo, canal o máscara de interés).

A.2. Procesamiento global en OpenCV.

- Es importante observar las **restricciones** impuestas sobre las imágenes admitidas en cada función.
- En otro caso ocurrirá un error (se produce una **excepción**).
- **Normalmente**:
  - Todas las imágenes deben ser del **mismo tipo**: profundidad y número de canales.
  - Todas las imágenes deben tener el **mismo tamaño**. Si se utilizan ROI, el tamaño del ROI debe ser el mismo en todas.
  - Algunas funciones requieren imágenes con 1 solo canal.
  - Opcionalmente, algunas admiten el uso de una **máscara** (**mask**), que será otra imagen, de 8 bits y con un 1 canal. Un valor 0 significa que el píxel no se procesa, y ≠ 0 sí se procesa.
  - Ver la **documentación** para cada función.

A.2. Procesamiento global en OpenCV.

- Operaciones **unarias**:
  - cvSet, cvSetZero, cvCopy, cvAddS, cvSubS, cvSubRS, cvAbsDiffS, cvAbs, cvScale, cvConvert, cvConvertScale, cvConvertScaleAbs, cvAndS, cvOrS, cvXorS, cvNot, cvCmpS, cvThreshold, cvAdaptiveThreshold, cvMaxS, cvMinS, cvPow, cvLog
- Operaciones **binarias**:
  - cvAdd, cvAddWeighted, cvSub, cvAbsDiff, cvMul, cvDiv, cvAnd, cvOr, cvXor, cvCmp, cvMax, cvMin
- Operaciones con **histogramas**:
  - cvCreateHist, cvReleaseHist, cvCalcHist, cvQueryHistValue, cvGetHistValue, cvNormalizeHist, cvGetMinMaxHistValue, cvEqualizeHist, cvLUT

A.2. Procesamiento global en OpenCV.

- **Inicializar una imagen con un valor constante**:
  void **cvSet** (CvArr* A, CvScalar S, const CvArr* mask=0)
  → si mask(x,y)≠0 entonces A(x,y):= S
  - **Ejemplo**. Inicializar a verde:
    cvSet(img, cvScalar(0,255,0));
- **Inicializar una imagen con un valor 0**:
  void **cvSetZero** (CvArr* A) / void **cvZero** (CvArr* A)
  → A(x,y):= 0
- **Copiar una imagen en otra**:
  void **cvCopy** (const CvArr* A, CvArr* B, const CvArr* mask =0)
  → si mask(x,y)≠0 entonces B(x,y):= A(x,y)
  - **Ejemplo**. Copiar un trozo de la imagen img en la imagen img2:
    cvSetImageROI(img, cvRect(50,50, 100, 100));
    img2= cvCreateImage(cvSize(100, 100), img->depth, img->nChannels);
    cvCopy(img, img2);
    cvResetImageROI(img);

A.2. Procesamiento global en OpenCV.

- **Sumar a una imagen un valor constante**:
  void **cvAddS** (const CvArr* A, CvScalar S, CvArr* C, const CvArr* mask=0)
  → si mask(x,y)≠0 entonces C(x,y):= A(x,y) + S
  - **Ejemplo**. Sumar un poco de azul: cvAddS(img, cvScalar(40,0,0), img2);
  - Aumentar el brillo (inplace):
    cvAddS(img, cvScalarAll(50), img);
- **Restar a una imagen un valor constante**:
  void **cvSubS** (const CvArr* A, CvScalar S, CvArr* C, const CvArr* mask=0)
  → si mask(x,y)≠0 entonces C(x,y):= A(x,y) – S
  - **Ejemplo**. Disminuir el brillo:
    cvSubS(img, cvScalarAll(50), img);
- **Restar a un valor constante una imagen** :
  void **cvSubRS** (const CvArr* A, CvScalar S, CvArr* C, const CvArr* mask=0)
  → si mask(x,y)≠0 entonces C(x,y):= S – A(x,y)
  - **Ejemplo**. Invertir una imagen (en color o B/N).
    cvSubRS(img, cvScalarAll(255), img);

A.2. Procesamiento global en OpenCV.

- **Valor absoluto de diferencia entre una imagen y un valor constante**:
  void **cvAbsDiffS** (const CvArr* A , CvArr* C, CvScalar S)
  → C(x,y):= |A(x,y) – S|
  - **Ejemplo**. cvAbsDiffS(img, img2, cvScalar(40,128,150));
- **Valor absoluto de una imagen:**
  void **cvAbs** (const CvArr* A, CvArr* C)
  → C(x,y):= |A(x,y)|
  - Tiene sentido cuando la profundidad son números con signo
- **Producto/división de una imagen por una constante**:
  void **cvScale** (const CvArr* A, CvArr* B, double scale=1, double shift=0)
  → B(x,y):= A(x,y) * scale + shift (igual para todos los canales)
  - Se puede hacer cualquier transformación lineal.
  - La función permite hacer conversiones entre imágenes con distintos valores de profundidad (de 8 bits a 16, o a reales de 32 bits, etc).
  - Tiene sinónimos, como cvConvertScale. Ver también cvConvertScaleAbs.
  - **Ejemplo**. Ajuste del contraste: cvConvertScale(img, img2, 1.6, -50);

A.2. Procesamiento global en OpenCV.

- **Ajuste lineal del histograma** (y otros métodos de normalización):
  void **cvNormalize**(const CvArr* A, CvArr* C, double a, double b, CV_MINMAX)
  → hace que los píxeles de C vayan entre **a** y **b**. A y C deben ser de 1 canal
- **Y lógico entre una imagen y un valor constante** (a nivel de bits):
  void **cvAndS** (const CvArr* A, CvScalar S, CvArr* C, const CvArr* mask=0)
  → si mask(x,y)≠0 entonces C(x,y):= A(x,y) AND S
- **O lógico entre una imagen y un valor constante** (a nivel de bits):
  void **cvOrS** (const CvArr* A, CvScalar S, CvArr* C, const CvArr* mask=0)
  → si mask(x,y)≠0 entonces C(x,y):= A(x,y) OR S
- **O exclusivo entre una imagen y un valor constante** (a nivel de bits):
  void **cvXorS** (const CvArr* A, CvScalar S, CvArr* C, const CvArr* mask=0)
  → si mask(x,y)≠0 entonces C(x,y):= A(x,y) XOR S
- **Negación lógica de una imagen** (a nivel de bits):
  void **cvNot** (const CvArr* A, CvArr* C)
  → C(x,y):= NOT A(x,y)
  - **Ejemplo**. Vale para invertir una imagen: cvNot(img, img);
  - Pero, ¿para qué valen las otras operaciones booleanas con constantes?

A.2. Procesamiento global en OpenCV.

- **Comparación entre una imagen y un valor constante**:
  void **cvCmpS** (const CvArr* A, double S, CvArr* C, int cmp_op)
  → C(x,y):= A(x,y) op S, con op ∈ {"=", ">", "<", ">=", "<="}
  - **cmp_op** indica el modo de comparación: CV_CMP_EQ (igual), CV_CMP_GT (mayor que), CV_CMP_GE (mayor o igual), etc.
  - El resultado es una imagen binaria, con 0 ó 255 (todos los bits a 1).
  - Ver también la función **cvInRangeS(img, scalar1, scalar2, res)**.
- **Umbralización/binarización de una imagen**:
  void **cvThreshold** (const CvArr* src, CvArr* dst, double threshold, double maxValue, int thresholdType)
  - Umbraliza la imagen según el método dado en **thresholdType**. y el umbral es **threshold**.
  - P.ej., CV_THRESH_BINARY para binarizar:
  → C(x,y):= si A(x,y) > threshold entonces maxValue sino 0
  - La umbralización se hace con un valor constante. Para un método más avanzado ver **cvAdaptiveThreshold**. El umbral se calcula para cada píxel, usando una vecindad local (adaptativo).

A.2. Procesamiento global en OpenCV.

- **Máximo entre una imagen y un valor constante**:
  void **cvMaxS** (const CvArr* A, double scalar, CvArr* C) *(está mal en la ayuda)*
  →C(x,y):= max {A(x,y), S}
  - Las imágenes deben ser de un solo canal.
- **Mínimo entre una imagen y un valor constante**:
  void **cvMinS** (const CvArr* A, double scalar, CvArr* C) *(está mal en la ayuda)*
  →C(x,y):= min {A(x,y), S}
  - Las imágenes deben ser de un solo canal.
- **Potencia, exponencial y logaritmo de los píxeles de una imagen**:
  void **cvPow** (const CvArr* A, CvArr* C, double p) → $C(x,y):= A(x,y)^p$
  void **cvExp** (const CvArr* A, CvArr* C) → $C(x,y):= e^{A(x,y)}$
  void **cvLog** (const CvArr* A, CvArr* C) → $C(x,y):= \log_e |A(x,y)|$
  - Para evitar saturación y pérdida de información, es conveniente transformar las profundidad a reales de 32 o 64 bits.
  - **Ejemplo**. Transformación de gamma:
    cvConvertScale(img, imr, 1./255, 0); *// imr es de profundidad 32F*
    cvPow(imr, imr, gamma);
    cvConvertScale(imr, img, 255., 0);

A.2. Procesamiento global en OpenCV.

Operaciones Binarias

- **Sumar dos imágenes**:

  void **cvAdd** (const CvArr* A, const CvArr* B, CvArr* C, const CvArr* mask=0)

  →si mask(x,y)≠0 entonces C(x,y):= A(x,y) + B(x,y)
  - Las imágenes deben tener el mismo tamaño (o ROI) y el mismo tipo.
  - **Ojo**: recordar los problemas de saturación. Por ejemplo, para obtener la media de dos imágenes, im1, im2:
    cvScale(im1, im1, 0.5, 0);
    cvScale(im2, im2, 0.5, 0); *// ¿Qué pasa si hacemos primero la suma*
    cvAdd(im1, im2, imr, 0); *// y luego la división por 2?*
- **Suma ponderada de dos imágenes**:

  void **cvAddWeighted**(CvArr* A, double a, CvArr* B, double b, double g, CvArr* C)

  →C(x,y):= a·A(x,y) + b·B(x,y) + g
  - Las mismas restricciones que antes.
  - Es mucho más adecuada para calcular la media de dos imágenes:
    cvAddWeighted(im1, 0.5, im2, 0.5, 0, imr);

A.2. Procesamiento global en OpenCV.

- **Restar dos imágenes**:

  void **cvSub** (const CvArr* A, const CvArr* B, CvArr* C, const CvArr* mask=0)

  →si mask(x,y)≠0 entonces C(x,y):= A(x,y) - B(x,y)
  - Las imágenes deben tener el mismo tamaño (o ROI) y el mismo tipo.
  - Igual que antes, tener cuidado con los problemas de saturación.
  - Esta operación tiene más sentido cuando se usan tipos con signo (16S, 16F, 32F).
- **Diferencia absoluta entre dos imágenes**:

  void **cvAbsDiff** (const CvArr* A, const CvArr* B, CvArr* C)

  →C(x,y):= |A(x,y) - B(x,y)|
  - Más adecuada cuando tenemos imágenes sin signo y solo queremos medir diferencias absolutas entre píxeles.

A.2. Procesamiento global en OpenCV.

- **Multiplicar dos imágenes**:

  void **cvMul** (const CvArr* A, const CvArr* B, CvArr* C, double scale=1)

  →C(x,y):= A(x,y)*B(x,y)*scale
  - El valor **scale** permite evitar problemas de saturación.
  - **Ejemplo**: multiplicar dos imágenes de 8 bits:
    cvMul(im1, im2, imr, 1./255);
- **Dividir dos imágenes**:

  void **cvDiv** (const CvArr* A, const CvArr* B, CvArr* C, double scale=1)

  →C(x,y):= scale*A(x,y)/B(x,y)
  - **A** puede ser nulo, en cuyo caso se supone que todos los píxeles son 1.
  - Igual que antes, tener cuidado con los problemas de saturación. Es más adecuado usar enteros con signo.
  - **Ejemplo**:
    cvDiv(im1, im2, imr, 255.0);

A.2. Procesamiento global en OpenCV.

- **Y lógico, a nivel de bits, entre dos imágenes**:

  void **cvAnd** (const CvArr* A, const CvArr* B, CvArr* C, const CvArr* mask=0)

  →si mask(x,y)≠0 entonces C(x,y):= A(x,y) AND B(x,y)
  - También funciona con números reales, pero ¿para qué puede valer?
  - Para que tenga sentido, al menos alguna de las dos imágenes debería ser binaria (0/255).
- **O lógico, a nivel de bits, entre dos imágenes**:

  void **cvOr** (const CvArr* A, const CvArr* B, CvArr* C, const CvArr* mask=0)

  →si mask(x,y)≠0 entonces C(x,y):= A(x,y) OR B(x,y)
- **O exclusivo, a nivel de bits, entre dos imágenes**:

  void **cvXor** (const CvArr* A, const CvArr* B, CvArr* C, const CvArr* mask=0)

  →si mask(x,y)≠0 entonces C(x,y):= A(x,y) XOR B(x,y)

A.2. Procesamiento global en OpenCV.

- **Comparación de dos imágenes**:

  void **cvCmp** (const CvArr* A, const CvArr* B, CvArr* C, int cmp_op)

  →C(x,y):= A(x,y) op B(x,y), con op ∈ {"=", ">", "<", ">=", "<="}
  - Los mismos modos de comparación que cvCmpS.
  - La imagen de salida es binaria 0/255. Las imágenes deben tener 1 canal.
  - Ver también la función **cvInRange**(img, imMin, imMax, imRes).
- **Máximo de dos imágenes**:

  void **cvMax** (const CvArr* A, const CvArr* B, CvArr* C)

  →C(x,y):= max {A(x,y), B(x,y)}
  - Las imágenes deben tener 1 solo canal.
- **Mínimo de dos imágenes**:

  void **cvMin** (const CvArr* A, const CvArr* B, CvArr* C)

  →C(x,y):= min {A(x,y), B(x,y)}
  - Las imágenes deben tener 1 solo canal.

A.2. Procesamiento global en OpenCV.

- Observar el **estilo de programación** usando estas funciones.
- Por ejemplo, queremos implementar la operación:
  R(x,y):= A(x,y)·(255-N(x,y))/255 + B(x,y)·N(x,y)/255

void **Combina** (const CvArr* A, const CvArr* B, const CvArr* N, CvArr* R);

- **Implementación 1**.

```
cvMul(B, N, B, 1./255);
cvNot(N, N);
cvMul(A, N, A, 1./255);
cvAdd(A, B, R);
```

Esto es más **sencillo** y **eficiente**, porque las operaciones están **optimizadas**

Aunque no del todo correcto, modifica sus parámetros de entrada A, B y N...

- **Implementación 2**.

```
#define CM(a,b,n)   (a*(255-n)+b*n)/255.0
CvScalar pA, pB, pN, pR;
for (int y=0; y<A->height; y++)
   for (int x= 0; x<A->width; x++) {
      pA= cvGet2D(A, y, x);
      pB= cvGet2D(B, y, x);
      pN= cvGet2D(N, y, x);
      pR.val[0]= CM(pA.val[0],pB.val[0],pN.val[0]);
      pR.val[1]= CM(pA.val[1],pB.val[1],pN.val[1]);
      pR.val[2]= CM(pA.val[2],pB.val[2],pN.val[2]);
      cvSet2D(R, y, x, pR);
   }
```

Esto es **menos eficiente** (~5 veces más lento) y menos recomendable

A.2. Procesamiento global en OpenCV.

Operaciones con histogramas

- En OpenCV se define el tipo **CvHistogram** y las operaciones para manejarlo: cvCreateHist, cvReleaseHist, cvCalcHist, cvQueryHistValue, cvGetHistValue, cvNormalizeHist, cvThreshHist, cvGetMinMaxHistValue.
- Tenemos también una operación de ecualización del histograma: cvEqualizeHist.
- Otra cuestión relacionada son las **tablas de transformación** (*look-up table*, o *LUT*), para realizar una transformación de curva tonal arbitraria.

A.2. Procesamiento global en OpenCV.

- **Propiedades** de un histograma:
  - **Número de dimensiones**. Normalmente tendremos 1 dimensión (escala de grises), 2 dimensiones (histogramas conjuntos de dos canales) o como mucho 3.
  - Para cada dimensión, **número de celdas** (*bins*). Normalmente será una potencia de 2, como 256, 64, 32...
  - **Rango de valores** correspondientes a cada celda, en el caso de haber menos celdas que valores (normalmt. será uniforme).
- **Ejemplos.**

Histograma de 1 dimensión y 4 celdas

Bin 0 (0-63)	Bin 1 (64-127)	Bin 2 (128-191)	Bin 3 (192-255)

Histograma de 2 dimensiones

	Bin 0 (0-127)	Bin 1 (128-255)
Bin 0 (0-85)		
Bin 1 (86-170)		
Bin 2 (171-255)		

A.2. Procesamiento global en OpenCV.

- **Crear un histograma**:

  CvHistogram* **cvCreateHist** (int dims, int* sizes, int type, float** ranges=0, int uniform=1)
  - **dims**: número de dimensiones del histograma.
  - **sizes**: número de celdas en cada dimensión.
  - **type**: tipo, usar siempre CV_HIST_ARRAY.
  - **uniform**: poner a 1 para que el rango de valores en cada celda sea uniforme.
  - **ranges**: rango asociado a cada celda de cada dimensión. Para histogramas uniformes, es un array de pares (r_{min}, r_{max}) y las celdas se reparten uniformemente este rango. Si la imagen es de 8 bits y el rango es (0, ..., 255), se puede poner simplemente **ranges**=NULL.
- **Ejemplo.** Crear un histograma con 1 dimensión y 256 celdas:

```
CvHistogram* hist1;
int celdas= 256;
hist= cvCreateHist(1, &celdas, CV_HIST_ARRAY);
```


A.2. Procesamiento global en OpenCV.

- **Ejemplo.** Crear un histograma con 2 dimensiones y 32 celdas en cada dimensión:

```
CvHistogram* hist2;
int celdas[2]= {32, 32};
hist2= cvCreateHist(2, celdas, CV_HIST_ARRAY);
```

- **Liberar la memoria ocupada por un histograma**:

  void **cvReleaseHist** (CvHistogram** hist)
- **Calcular el histograma de una imagen**:

  void **cvCalcHist** (IplImage** img, CvHistogram* hist, int acumulate=0, const CvArr* mask=0)
  - **img**: array de imágenes de 1 canal. Deben haber tantas como número de dimensiones del histograma.
  - **acumulate**: si se pone a TRUE, no borra el contenido anterior de las celdas. Esto permite obtener histogramas acumulados de varias imágenes.
  - **mask**: máscara sobre la que se calculará el histograma.

A.2. Procesamiento global en OpenCV.

- Esta forma tan particular de calcular el histograma, hace que sea necesario **separar los canales** de una imagen.

- **Ejemplo 1.** Calcular el histograma unidimensional del nivel de gris:

```
IplImage *gris= cvCreateImage(cvGetSize(img), 8, 1);  // Ver abajo
cvCvtColor(img, gris, CV_RGB2GRAY);
cvCalcHist(&gris, hist1, 0);
```

- **Ejemplo 2.** Calcular el histograma bidimensional de los canales R y G:

```
IplImage *planos[3];
planos[0]= cvCreateImage(cvGetSize(img), 8, 1);
planos[1]= cvCreateImage(cvGetSize(img), 8, 1);
planos[2]= cvCreateImage(cvGetSize(img), 8, 1);
cvSplit(src, planos+0, planos+1, planos+2, 0);
cvCalcHist(planos, hist2, 0);
```


A.2. Procesamiento global en OpenCV.

- Una vez calculado... **consultar las celdas del histograma**:

  float **cvQueryHistValue_1D** (CvHistogram*hist, int idx0)
  - Para el caso de histogramas **unidimensionales**.
  - Realmente no es una función sino un macro.

  float **cvQueryHistValue_2D** (CvHistogram*hist, int idx0, int idx1)
  - Para el caso de histogramas bidimensionales.
  - De forma similar para histogramas 3D y ND.
- **Obtener un puntero a una celda del histograma**:

  float * **cvGetHistValue_1D** (CvHistogram*hist, int idx0)
  - Para el caso de histogramas **unidimensionales**.
  - Devuelve un puntero. Esta función será útil si lo que queremos es poder modificar a mano el valor de las celdas.

  float * **cvGetHistValue_2D** (CvHistogram*hist, int idx0, int idx1)
  - Para el caso de histogramas bidimensionales.
  - De forma similar para histogramas 3D y ND.

A.2. Procesamiento global en OpenCV.

- Existen otras operaciones interesantes de consulta y manipulación de histogramas.
- **Normalizar un histograma**:

  void **cvNormalizeHist** (CvHistogram* hist, double factor)
  - Hace que la suma de todas las celdas del histograma sea **factor**.
  - Puede ser interesante para visualizar o comparar histogramas.
- **Obtener máximo y mínimo de un histograma**:

  void **cvGetMinMaxHistValue** (const CvHistogram* hist,
  float* minVal, float* maxVal, int* minIdx =0, int* maxIdx =0)
  - Dado el histograma, calcula el mínimo (**minVal**), el máximo (**maxVal**), el índice de la celda mínima (**minIdx**) y máxima (**maxIdx**).
  - Interesante, p.ej., para escalar todas las celdas entre 0 y 1.
- **Ecualizar el histograma de una imagen**:

  void **cvEqualizeHist** (const CvArr* src, CvArr* dst)
  - **Ojo**: la imagen debe ser de 1 solo canal y 8U. No se puede usar para hacer una ecualización conjunta de RGB.

A.2. Procesamiento global en OpenCV.

- Las **tablas de transformación** (*look-up table*, *LUT*) son tablas que definen funciones discretas de la forma:

  f: [0...255] → R
- Esto nos permite construir cualquier **curva tonal** arbitraria.
- En OpenCV, una LUT es una **matriz** de tipo **CvMat**:

  CvMat* lut= cvCreateMat(1, 256, CV_8UC3);
  - 1 fila y 256 columnas. Tantos canales como la salida (C1, C3).
  - La profundidad puede cambiar (8S, 16U, 32F, ...)
- **Aplicar una transformación de tabla LUT**:

  CvMat* **cvLUT** (const CvArr* A, CvArr* B, const CvArr* lut)

  B(x,y):= lut(A(x,y))
  - La imagen **A** puede tener 1 o varios canales. **B** y **lut** deben tener el mismo número y la misma profundidad.
  - La profundidad de entrada, **A**, debe ser 8 bits (con o sin signo), y la de salida (en **B** y en **lut**) puede ser cualquiera.

A.2. Procesamiento global en OpenCV.

- **Ejemplo 1**. Aplicar una ecualización conjunta del histograma a una imagen **img** (existente) usando CvHist y LUT:

```
IplImage *gris= cvCreateImage(cvGetSize(img), img->depth, 1);
cvCvtColor(img, gris, CV_RGB2GRAY);
int celdas= 256;
CvHistogram* hist1= cvCreateHist(1, &celdas, CV_HIST_ARRAY);
cvCalcHist(&gris, hist1);
cvNormalizeHist(hist1, 256.0);
CvMat *lut= cvCreateMat(1, 256, CV_8UC3);
float acum= 0.0;
for (int i= 0; i<255; i++) {
  cvSet1D(lut, i, cvScalarAll(acum));
  acum+= cvQueryHistValue_1D(hist1, i);
}
cvSet1D(lut, 255, cvScalarAll(255));
cvLUT(img, img, lut);
cvReleaseHist(&hist1);
cvReleaseMat(&lut);
cvReleaseImage(&gris);
```


A.2. Procesamiento global en OpenCV.

- **Ejemplo 2**. Media ponderada entre dos imágenes. En un proyecto nuevo, crear un botón y añadir el siguiente código.

```
int peso= 50;
IplImage *ent1, *ent2, *sal;

void on_trackbar(int nada)
{
   cvAddWeighted(ent1, peso/100.0, ent2, 1.0-peso/100.0, 0, sal);
   cvShowImage("Imagen", sal);
}

void MainWindow::on_pushButton_clicked(){
   ent1= cvLoadImage("encolor1.bmp", 1);
   IplImage *tmp= cvLoadImage("encolor2.bmp", 1);
   if (!ent1 || !tmp) return;
   ent2= cvCloneImage(ent1);
   cvResize(tmp, ent2);
   cvReleaseImage(&tmp);
   sal= cvCloneImage(ent1);
   cvNamedWindow("Imagen", 0);
   cvCreateTrackbar("Peso", "Imagen", &peso, 100, on_trackbar);
   on_trackbar(50);
   cvWaitKey(0);
   cvReleaseImage(&ent1);
   cvReleaseImage(&ent2);
   cvReleaseImage(&sal);
   cvDestroyWindow("Imagen");
}
```

Fuera del slot del botón

Dentro del slot del botón