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.

(c) Ginés García Mateos, http://dis.um.es/profesores/ginesgm
Dept. de Informática y Sistemas, Universidad de Murcia

Tema 6. Análisis de imágenes.

6.1. Búsqueda de patrones.
6.2. Flujo óptico.
6.3. Integrales proyectivas.
6.4. Análisis del color.
A.6. Análisis de imágenes en OpenCV.

6. Análisis de imágenes.

- **Análisis de imágenes**: procesamiento "inteligente" de las imágenes orientado a la extracción de información de tipo cualitativo (qué hay en las imágenes) o cuantitativo (posiciones, tamaños, distancias, tonos, etc.).
- **Objetivos del análisis**:
  - **Detección de objetos**: encontrar en la imagen las instancias de cierto tipo o clase de objetos.
  - **Reconocimiento de objetos**: distinguir la identidad específica de un objeto que se conoce que pertenece a cierta clase.
  - **Segmentación**: separar los objetos de interés del fondo.
  - **Seguimiento y correspondencia**: encontrar la equivalencia de puntos entre dos imágenes (por ejemplo, imágenes en una secuencia de vídeo o en un par estéreo).
  - **Reconstrucción 3D**: extraer información 3D de la escena, posiciones, ángulos, velocidades, etc.

6.1. Búsqueda de patrones.

- La **búsqueda de patrones** es una técnica de análisis que se puede aplicar en detección de objetos, reconocimiento, seguimiento y correspondencia.
- **Idea de la técnica**: dada una imagen (un **patrón** o **modelo**) encontrar sus apariciones dentro de otra imagen mayor.
- No se buscan sólo las apariciones "exactas", sino permitiendo cierto grado de variación respecto al patrón.
- **Ejemplo**. Buscar el patrón:  en la imagen dada.

Resultado: nº de apariciones, localización de cada una y "verosimilitud"

6.1. Búsqueda de patrones.

- El método más sencillo de búsqueda de patrones es el ***template matching*** (**comparación de plantillas**).
- **Template matching**: sea **A** una imagen (de tamaño WxH), y sea **P** un patrón (de wxh), el resultado es una imagen **M** (de tamaño (W-w+1)x(H-h+1)), donde cada píxel **M(x,y)** indica la "verosimilitud" (probabilidad) de que el rectángulo [x,y] – [x+w-1, y+h-1] de **A** contenga el patrón **P**.
- La imagen **M** se define usando alguna función de diferencia (o similitud) entre dos trozos de imagen.
  M(x,y):= d({A(x,y),...,A(x+w-1,y+h-1)}, {P(0,0),..., P(w-1,h-1)})
- **Ejemplo**. Suma de diferencias al cuadrado:

$$M(x, y) := \sum_{a=0..w-1} \sum_{b=0..h-1} (P(a, b) - A(x+a, y+b))^2$$

Es parecido a una **convolución** (pasar una máscara por toda la imagen)

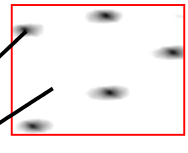
6.1. Búsqueda de patrones.

- **Ejemplo**. *Template matching* con suma de diferencias al cuadrado.

Imagen de entrada **A** (239x156)

P - patrón a buscar (68x37)

Mapa de matching **M**



$6{,}58 \cdot 10^6$

$125{,}3 \cdot 10^6$

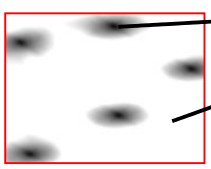
Mapa superpuesto

6.1. Búsqueda de patrones.

- Los **valores bajos** (color oscuro) indican alta probabilidad de que el patrón se encuentre en esa posición (esquina superior izquierda).
- Los **valores altos** (color blanco) indican probabilidad baja.
- ¿Cuánto es alto o bajo? → Normalizar el resultado.
- **Normalización**: dividir el resultado por:

$$\text{sqrt}\left(\sum_{a=0..w-1}\sum_{b=0..h-1} P(a,b)^2 \cdot \sum_{a=0..w-1}\sum_{b=0..h-1} A(x+a, y+b)^2\right)$$

- **Ejemplo**. Diferencias al cuadrado normalizadas.



6.1. Búsqueda de patrones.

- Se pueden usar también otras medidas de distancia.
- **Ejemplo**. Producto escalar de patrones "centrados".

$$M(x, y) := \sum_{a=0..w-1}\sum_{b=0..h-1} (P'(a,b)\cdot A'(x+a, y+b))$$

Esto es lo que se llama la **correlación**

donde $P'(a,b) := P(a,b) - \text{Media}(P)$. Lo mismo para A'.

- El valor (normalizado) está entre -1 y +1. Cuanto mayor (más próximo a +1) más probabilidad.

Imagen de entrada, **A** Mapa de matching, **M**

Patrón, **P**

6.1. Búsqueda de patrones.

- Una de las principales aplicaciones del *template matching* es la detección de objetos.
- **Proceso de detección de objetos** usando búsqueda de patrones.

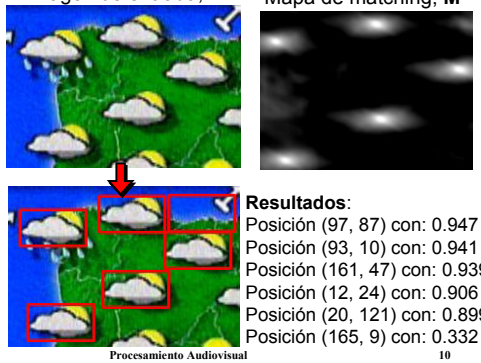
1) Conseguir un patrón, **P**, representativo de la clase de objetos a buscar.
2) Aplicar el *template matching* a la imagen, obteniendo **M**.
3) Buscar los máximos (o mínimos) locales de **M**.
   - 3.1) Buscar el máximo global, (**Ix**, **Iy**)= $\text{argmax}_{\forall x, y}$ **M**(x, y).
   - 3.2) Si **M**(**Ix**, **Iy**) es menor que cierto umbral, acabar.
   - 3.3) Añadir la posición (**Ix**, **Iy**) a una lista de localizaciones resultantes del proceso.
   - 3.4) Poner a cero en **M** el rectángulo [Ix-w,Iy-h]–[Ix+w,Iy+h].
   - 3.5) Volver al paso 3.1.

6.1. Búsqueda de patrones.

- **Ejemplo 1**. Detección de objetos con *template matching*.

Imagen de entrada, **A** Mapa de matching, **M**

Patrón, **P**



Resultados:
Posición (97, 87) con: 0.947
Posición (93, 10) con: 0.941
Posición (161, 47) con: 0.939
Posición (12, 24) con: 0.906
Posición (20, 121) con: 0.899
Posición (165, 9) con: 0.332

6.1. Búsqueda de patrones.

- Pero, normalmente, el problema no es tan sencillo. Las clases de objetos presentan mayor variabilidad, y pueden haber variaciones de tamaño y rotación.
- El umbral debe bajarse, produciendo **falsos positivos**.
- **Ejemplo 2**. Detección de caras humanas.

Patrón, **P** (29x27)

Patrón ampliado

Imagen de entrada, **A** (640x480)

6.1. Búsqueda de patrones.

- **Ejemplo 2**. Detección de caras humanas con *template matching*.

Mapa de matching, **M** Resultados de la detección

- **Función**: producto vectorial.
- **Umbral** usado: 0,5

6.1. Búsqueda de patrones.

- Obviamente, la técnica es muy sensible a cambios de **escala**, **rotación** o **deformaciones 3D** de los objetos.
- **Ejemplo 1**. Cambio de escala.

Imagen de entrada, **A**

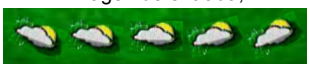
63% 82% 100% 116% 143%

Mapa de matching, **M**

0,523 0,724 0,947 0,761 0,640

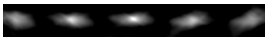
- **Ejemplo 2**. Cambio de rotación.

Imagen de entrada, **A**



20º 10º 0º 15º 25º

Mapa de matching, **M**



0,574 0,756 0,947 0,664 0,507

6.1. Búsqueda de patrones.

- **Soluciones**:
  - Utilizar varios patrones, con distintos tamaños y rotaciones.
  - Hacer una **búsqueda multiescala**. Aplicar el proceso escalando la imagen a: 50%, 75%, 100%, 125%, ...
  - Usar alguna técnica de **atención selectiva**. Por ejemplo, usar color o bordes para centrar la atención en ciertas partes de la imagen.
- Otra aplicación interesante del *template matching* es la **correspondencia**: dado un par de imágenes de una misma escena, encontrar los puntos equivalentes de ambas.
- **Idea**: el patrón se extrae de una imagen y se aplica en la otra. El máximo (o mínimo) *matching* indica la equivalencia de puntos.
- **Ejemplo**: composición panorámica.

6.1. Búsqueda de patrones.

- **Problema**: dadas dos imágenes de sitios adyacentes, obtener una composición panorámica de forma automática.

Imagen **A** (izquierda)

Imagen **B** (derecha)

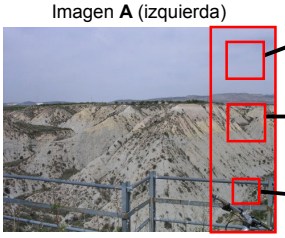
- Como vimos en el tema 4, se usa una transf. geométrica.
- ¿Cómo obtener los parámetros de la transf.? → Encontrar **puntos equivalentes** entre ambas imágenes.

6.1. Búsqueda de patrones.

- **Proceso** de composición panorámica:

1) Escoger dos trozos de la imagen **A** que se espera que aparezcan en **B**. ¿Qué trozos?
   1.1) Deben ser trozos en el solapamiento entre **A** y **B**. Si **A** es la imagen izquierda, un trozo de la derecha.
   1.2) El trozo debe tener elementos claramente definidos.

Imagen **A** (izquierda)



Este patrón no es muy bueno, no es nada significativo... descartarlo

Este patrón es OK

Este patrón también parece OK, pero... ¿y la barandilla?

6.1. Búsqueda de patrones.

2) Para cada patrón escogido, buscarlo en la imagen B.
   2.1) Aplicar *template matching*.
   2.2) Quedarse con el máximo.

Patrón 1

Mapa de matching

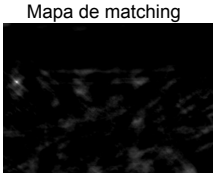


Imagen **B** (derecha)

Localización resultante

6.1. Búsqueda de patrones.

2) Para cada patrón escogido, buscarlo en la imagen B.

Patrón 2

Mapa de matching

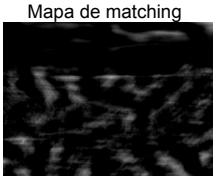


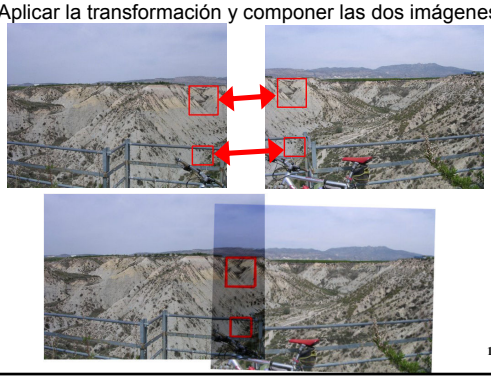
Imagen **B** (derecha)

Localizaciones resultantes

Aquí la cosa no está tan clara, pero podríamos aplicar unas cuantas **heurísticas** sencillas y descartar las localizaciones inviables...

6.1. Búsqueda de patrones.

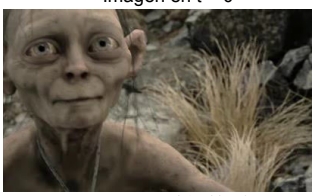
3) Con las localizaciones equivalentes, calcular los parámetros de la transf. geométrica.
4) Aplicar la transformación y componer las dos imágenes.



6.1. Búsqueda de patrones.

- Otra aplicación es el **seguimiento de objetos**: localizar la posición de un objeto a lo largo de una secuencia de vídeo.
- En un vídeo se espera que haya cierta "**continuidad temporal**", los elementos de la escena varían lentamente.
- **Idea**: aplicar *template matching*, usando como patrón el ROI del objeto en el instante *t*, aplicado sobre la imagen en *t*+1.
- **Ejemplo**. Seguimiento de caras. Suponemos una detección inicial.

Imagen en t = 0



Patrón de cara, P_0

6.1. Búsqueda de patrones.

- **Proceso de seguimiento** usando *template matching*:

1) Detectar la posición inicial del objeto, R_0.
2) Repetir para cada frame t= 1...N:
   2.1) Extraer la región P_{t-1} del frame t-1 usando R_{t-1}
   2.2) Aplicar matching del patrón P_{t-1} en la imagen del frame t.
   2.3) Seleccionar la pos. del máximo, poniendo el resultado en R_t.

R_0

Imagen en t = 0

Imagen en t = 1

6.1. Búsqueda de patrones.

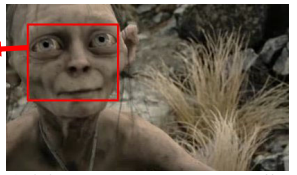
Patrón de cara, P_0

Imagen en t = 1

Mapa de matching

Localización resultante

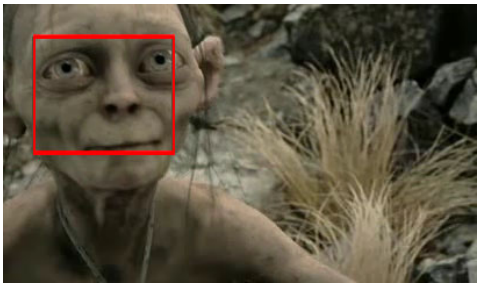
R_1



Ver que aquí el máximo es bastante destacado

6.1. Búsqueda de patrones.

- El proceso se repite para todos los frames de la secuencia.



- Se podrían añadir algunas **heurísticas** adicionales: que el salto no sea muy grande, que el valor de *matching* no baje de un umbral, etc.

6.1. Búsqueda de patrones.

Conclusiones:

- **Template matching**: buscar las apariciones de un trozo de imagen en otra imagen de tamaño mayor.
- El proceso de cálculo es parecido a una convolución.
- **Ventajas**:
  - La idea es muy sencilla, aunque tiene un gran potencial.
  - Aplicación en detección, reconocimiento, seguimiento de objetos, etc.
- **Desventajas**:
  - Es muy sensible a rotaciones, escala, etc.
  - Además, en la vida real encontramos objetos 3D flexibles, lo que supone más variabilidad.
  - La aplicación de la técnica es muy costosa, O(**WHwh**). Cuando la resolución aumenta al doble, el tiempo se multiplica por 16.

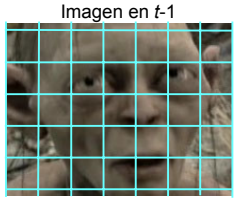
6.2. Flujo óptico.

- El **flujo óptico** es una técnica de análisis de imágenes que se aplica en secuencias de vídeo.
- En concreto, el **flujo óptico** define los vectores de movimiento de diferentes trozos de la imagen.
- **Aplicaciones**: detección de movimiento, seguimiento de objetos por partes, compresión de vídeo, composición, etc.
- **Ejemplo**. Imágenes de entrada Flujo óptico resultante


6.2. Flujo óptico.

- Existen diversas formas de calcular el flujo óptico.
- Una forma sencilla está basada en la **técnica del *template matching***: dividir la imagen en **bloques**, para cada bloque de una imagen buscar la correspondencia en la otra.

Imagen en *t*-1 Imagen en *t*




- Buscar todos los trozos en la otra imagen sería muy costoso...
- Pero normalmente el desplazamiento será pequeño → Buscar sólo en una cierta **vecindad local**.

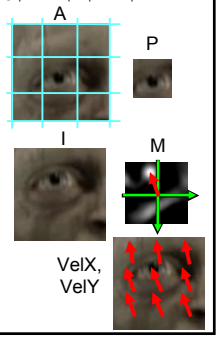
6.2. Flujo óptico.

- **Parámetros** para el cálculo del flujo óptico:
  - **Tamaño de los bloques** a usar.
    - Ni muy pequeños ni muy grandes. Si son pequeños, contienen pocas características y el matching es poco fiable.
    - Si son grandes, perdemos resolución (menos vectores de movimiento). También hay problemas si el bloque se sale de la imagen.
  - **Radio de búsqueda**. Determina el tamaño de la zona, en la imagen *t*, donde se busca el bloque de entrada de la *t*-1.
    - Cuanto más grande, más tiempo de ejecución.
    - Si es muy pequeño y el movimiento es mayor, el resultado será impredecible.
  - **Función de matching** a emplear. Para este problema se podría usar una simple suma de diferencias. Para conseguir invarianza a cambios de iluminación, mejor usar un producto vectorial normalizado.

6.2. Flujo óptico.

- **Proceso de cálculo del flujo óptico**.
  - **Parámetros de entrada**: A, B: imágenes de tamaño WxH; (w, h) tamaño de los bloques; (rx, xy) radio de búsqueda.
  - **Salida**: VelX, VelY: matrices de tamaño $\lceil W/w \rceil$ x $\lceil H/h \rceil$.

1) Para cada posición (i, j) en el rango de VelX y VelY hacer:
   1.1) Sea P el rectángulo $[i \cdot w, j \cdot h]$–$[(i+1)w-1, (j+1)h-1]$ de A
   1.2) Sea I el rectángulo $[i \cdot w-rx, j \cdot h-ry]$–$[(i+1)w-1+rx, (j+1)h-1+ry]$ de B
   1.3) Aplicar matching del patrón P en la imagen I, obteniendo el resultado en M de tamaño $(2 \cdot rx+1, 2 \cdot ry+1)$
   1.4) Buscar el máximo valor de matching: $(mx, my) = \text{argmax}_{\forall a,b} M(a,b)$
   1.5) VelX(i, j)= mx-rx; VelY(i, j)= my-ry

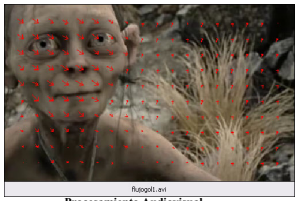


6.2. Flujo óptico.

- **Ejemplo**. Cálculo del flujo óptico por matching de bloques.

Vídeo de entrada

Resolución: 408x240



Flujo óptico resultante

Tamaño de bloque: 21x21
Radio de búsqueda: 21x21

6.2. Flujo óptico.

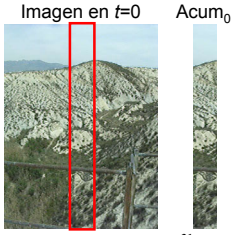
- Otra aplicación interesante es la **composición de vídeo por barrido**.
- **Problema**: dada una secuencia de vídeo donde la cámara gira (o se desplaza lateralmente), componer una imagen con todo el campo de visión disponible.
- **Ejemplo**.

Vídeo de entrada

Panorámica resultante

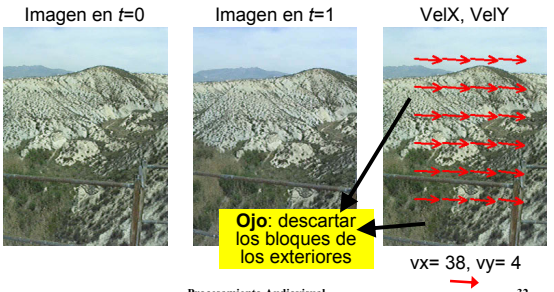
6.2. Flujo óptico.

- La **composición de vídeo** se puede ver como un proceso de **añadir tiras** de imágenes.
- El tamaño y posición de la tira añadida depende de la cantidad y dirección de movimiento detectado en las imágenes.
- **Proceso de composición por barrido**:
  1) En la imagen inicial **t**= 0, seleccionar una región central (una tira) perpendicular a la dirección del movimiento.
     Inicializar con ella la imagen acumulada (Acum).
  - Por ejemplo, seleccionar el rectángulo [100, 0]–[120, 320].



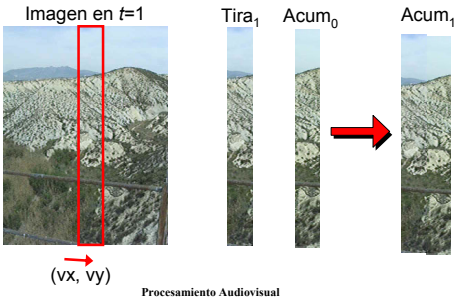
6.2. Flujo óptico.

2) Usando el flujo óptico, detectar la cantidad de movimiento de cada nuevo frame, *t*, respecto al anterior, *t*-1. → VelX, VelY
- Por ejemplo, se puede tomar la media de velocidad en X e Y, vx= Media(VelX), vy= Media(VelY).



6.2. Flujo óptico.

3) Añadir a la imagen acumulada, Acum, la tira correspondiente en función de la velocidad calculada en el paso anterior.
- En el ejemplo, añadir el rectángulo [100, 0]–[100+vx, 320], desplazado en (-vx, -vy) píxeles respecto al último añadido.



6.2. Flujo óptico.

- **Ejemplo**. Composición de vídeo por barrido.

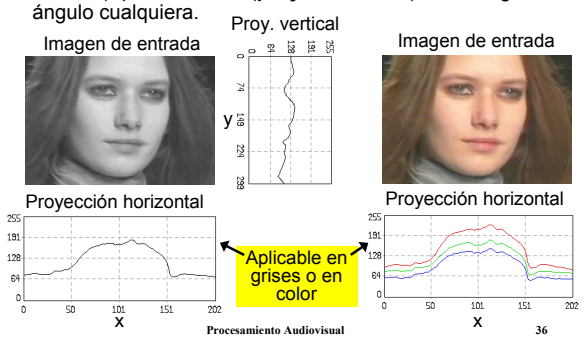
- Otras **cuestiones adicionales**:
  - Compensación del brillo (y tal vez del balance de blancos).
  - Al final puede ser necesario aplicar una rotación de la imagen.
  - ¿Qué ocurre si hay movimiento en la escena?

6.2. Flujo óptico.

Conclusiones:
- **Flujo óptico**: vectores de movimiento entre dos imágenes de una secuencia de vídeo.
- Es una técnica específica de vídeo.
- Además del método básico (utilizando *template matching*) existen otras muchas formas de calcularlo.
- **Ventajas**:
  - Permite comprender mejor la información contenida en un vídeo, la evolución en la escena: detectar si hay cambios en la escena, en qué posiciones, qué cantidad, etc.
- **Inconvenientes**:
  - La técnica es muy lenta. Es inviable aplicarla en tiempo real.
  - Difícil ajustar los parámetros para un funcionamiento óptimo: tamaño de bloques y radio de búsqueda.

6.3. Integrales proyectivas.

- Una **integral proyectiva** (o, simplemente, una **proyección**) de una imagen es la media de los píxeles por filas (**proy. vertical**), por columnas (**proy. horizontal**) o a lo largo de un ángulo cualquiera.



6.3. Integrales proyectivas.

- Las integrales proyectivas se pueden usar en detección, seguimiento y segmentación. Normalmente como fase previa a otros procesos.
- La principal característica es la **reducción de dimensiones** → De imágenes 2D a proyecciones 1D.
  - Más rápidas de procesar.
  - Pero, se puede perder información relevante.
- **Definición**. Sea **A** una imagen de WxH. La **integral proyectiva vertical**, denotada por $\mathbf{P_{VA}}$, es una tabla de tamaño H definida por:

$$P_{VA}(y) = 1/W \cdot \sum_{x=0..W-1} A(x, y); \forall\ y= 0, ..., H-1$$

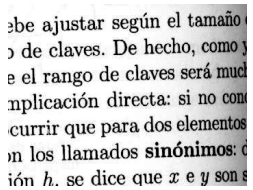
la **integral proyectiva horizontal**, denotada por $\mathbf{P_{HA}}$, es:

$$P_{HA}(x) = 1/H \cdot \sum_{y=0..H-1} A(x, y); \forall\ x= 0, ..., W-1$$

6.3. Integrales proyectivas.

- De forma similar, la **proyección a lo largo de un ángulo cualquiera α**, $P_{\alpha A}$, se puede definir como la integral vertical de la imagen rotada en α.
- El análisis de proyecciones consiste, en esencia, en **localizar los máximos y/o mínimos** de las proyecciones.
- **Ejemplo**. Detección y segmentación del texto en un OCR.
- **Problemas**: 1) rotar la imagen (alinearla horizontalmente), 2) detectar las líneas y 3) detectar las letras en cada línea.

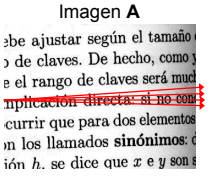
Imagen de entrada **A**



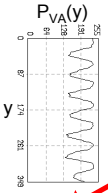
6.3. Integrales proyectivas.

1) **Alinear** la imagen horizontalmente. Calcular el ángulo α para el cual $P_{\alpha A}$ es más "plausible".

Imagen **A**

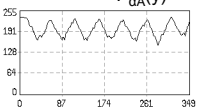


$P_{VA}(y)$

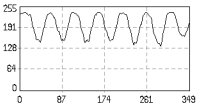


$P_{\alpha A}(y)$

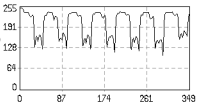
α= 2°



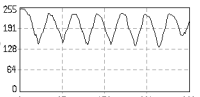
α= 0°



α= -2°



α= -4°



Criterio "plausible": mayor diferencia entre máximo y mínimo, o mayor varianza de la proyección

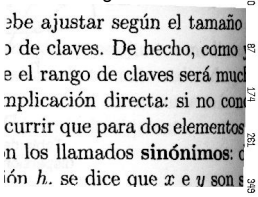
6.3. Integrales proyectivas.

1b) **Rotar** la imagen en el ángulo óptimo.

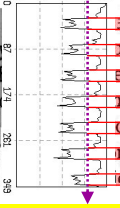
2) Detectar la componente Y de las líneas usando la proyección vertical.

Por ejemplo, fijando un umbral.

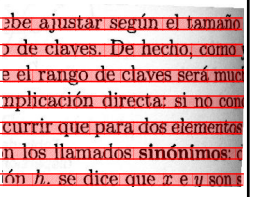
Imagen **A'**



$P_{VA'}(y)$



Líneas detectadas



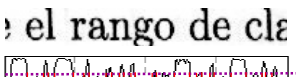
Umbral, prefijado o calculado automáticamente

6.3. Integrales proyectivas.

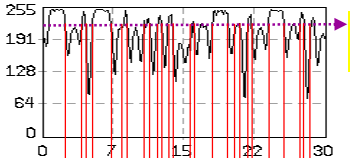
2b) **Segmentar** las líneas (con cierto margen arriba y abajo).

3) Usando la proyección horizontal, detectar cada uno de los caracteres.

Una línea segmentada, **s**



$P_{HS}(x)$



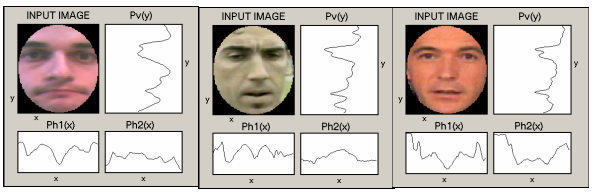
Umbral, aquí ya no está tan claro...

Caracteres detectados

Segmentación incorrecta...

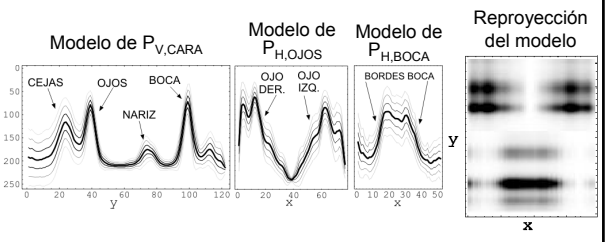
6.3. Integrales proyectivas.

- Las integrales proyectivas también se usan en **detección y seguimiento** de otros tipos de objetos más complejos, como las **caras humanas**.
- **Idea**: al aplicar las proyecciones sobre caras humanas, se obtienen patrones típicos de zonas claras y oscuras.
- **Ejemplo**. **Pv** → proy. vertical de la cara. **Ph1** → p. horizontal de la región de ojos. **Ph2** → p. horizontal de la región boca.



6.3. Integrales proyectivas.

- **Detección de caras humanas** usando proyecciones.
  - Crear un modelo de proy.: $P_{V,CARA}$, $P_{H,OJOS}$ y $P_{H,BOCA}$.
  - Para todas las zonas de la imagen, calcular sus 3 proy. (Pv, Ph1 y Ph2) y comparar con $P_{V,CARA}$, $P_{H,OJOS}$ y $P_{H,BOCA}$. Si la diferencia es pequeña → cara detectada.



6.3. Integrales proyectivas.

- **Ejemplo de aplicación: Interface perceptual**. Los resultados del seguimiento se aplican en el control del movimiento en un entorno virtual.

http://dis.um.es/~ginesgm/fip

6.3. Integrales proyectivas.

Conclusiones:

- **Integrales proyectivas**: acumulación de los valores de los píxeles a lo largo de cierta dirección.
- A partir de una imagen 2D se extraen proyecciones 1D.
- La técnica es aplicable cuando los objetos de interés son **distinguibles por intensidad** (más claros o más oscuros).
- Si esto no es posible, también se pueden aplicar las proyecciones sobre **imágenes de bordes**.
- **Ventajas**:
  - Se trabaja con información simplificada y acumulada. Más eficiencia y menos sensibilidad a ruido.
- **Inconvenientes**:
  - En algunas aplicaciones, la proyección puede suponer perder información relevante.
  - Difícil establecer umbrales máximos y mínimos adecuados.

6.4. Análisis de color.

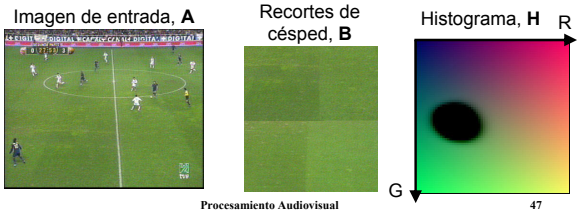
- Las técnicas anteriores se pueden aplicar tanto en imágenes en color como en escala de grises.
- Existen otras **técnicas** que hacen específicamente **uso del color**:
  - **Detección de zonas de color**. Seleccionar y modelar un color objetivo. Encontrar regiones conexas de ese color. Parecido al relleno de color, pero aplicado a toda la imagen.
    → Aplicaciones: sistemas croma-key (segmentación por color), detección de objetos (p.ej. de caras humanas usando color de piel), seguimiento basado en color.
  - **Reproyección del histograma**. Es un caso de la detección de zonas de color. Para cada color se define una verosimilitud (probabilidad), obtenida mediante un histograma de color.
  - **Comparación de histogramas**. Definir y usar una medida de diferencia entre histogramas. Aplicada fundamentalmente en indexación de imágenes (consultas basadas en similitud).

6.4. Análisis de color.

- **Reproyección del histograma**. Permite detectar un color partiendo de una imagen con regiones amplias de ese color.
- **Idea**: usar histogramas para modelar un color objetivo.
- **Proceso**:

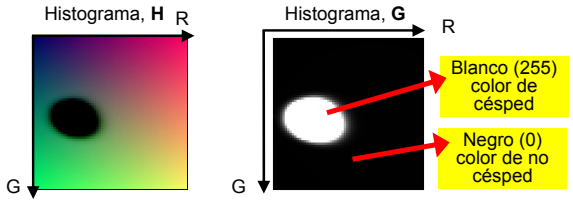
  1) Calcular el histograma de color de una zona de la imagen con el color objetivo.

  - **Ejemplo**. Histograma (R, G), con 64x64 celdas.



6.4. Análisis de color.

2) Usar el histograma como una medida de verosimilitud, probabilidad o distancia en el espacio de color.
Valor alto: césped. Valor bajo: no césped.



- **Ejemplo**. Buscar el máximo de **H** → hmax
  G(r, g) = min {255, 255·10·H(r, g)/hmax}

6.4. Análisis de color.

3) Dada una imagen nueva, **C**, aplicar los valores del histograma para detectar las zonas de césped.

M(x, y) = G(C(x, y).R, C(x, y).G), para todo (x, y)

Esto es lo que se conoce como la **reproyección del hist**.

Histograma, **G**

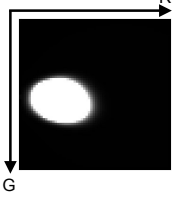


Imagen de entrada **C**

M, reproyección del histograma sobre **C**

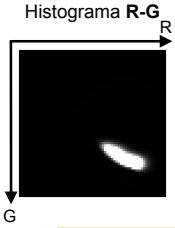
Césped segmentado

6.4. Análisis de color.

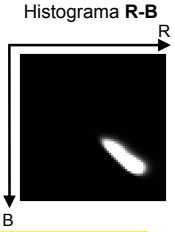
- **Ejemplo de aplicación: Sistema croma-key**. Se toma una imagen del panel de color usado como fondo. Y se calcula su histograma.

Imagen de fondo, **F**

Histograma **R-G**



Histograma **R-B**

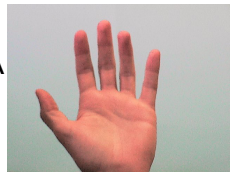


En este caso vamos a usar histogramas 3D de 32x32x32

6.4. Análisis de color.

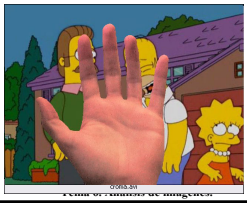
- Después se usa la reproyección para obtener la máscara de segmentación **M**.

Imagen nueva, **A**



M, reproyección del histograma sobre **A**

Imagen compuesta, **R**

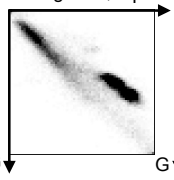


Binarizada y **erosionada** conveniente-mente (3 iteraciones)

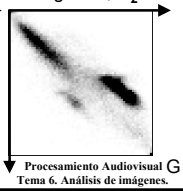
6.4. Análisis de color.

- **Comparación de histogramas**. Permite obtener una medida subjetiva de similitud entre imágenes, basada en color. Es útil en aplicaciones como indexación de imágenes.
- **Idea**: definir una medida de diferencia (o similitud) entre dos histogramas. La diferencia entre dos imágenes se reduce a la diferencia entre los histogramas correspondientes.
- Se pueden usar distintas medidas como: suma de diferencias al cuadrado, producto vectorial o solapamiento.

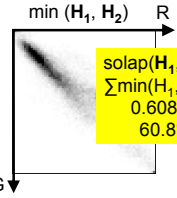
Histograma, $\mathbf{H_1}$



Histograma, $\mathbf{H_2}$



min ($\mathbf{H_1}$, $\mathbf{H_2}$)



solap($\mathbf{H_1}$, $\mathbf{H_2}$) = $\sum$min(H_1, H_2) = 0.608 → 60.8%

6.4. Análisis de color.

- **Ejemplo de aplicación: Indexación de imágenes**. En una base de datos de imágenes queremos añadir consultas basadas en similitud entre imágenes:
  - "Busca la imagen de la BD que más se parezca a una dada".
  - "Busca las *n* imágenes de la BD más relacionadas, en cuanto a su contenido, con una dada".
  - "Busca imágenes que traten sobre el mismo tema o categoría".
  - Por ejemplo, "busca imágenes de fútbol".
- Todas estas consultas se reducen a medidas de **distancia entre histogramas**, entre la imagen dada y las de la BD.
- En los dos últimos ejemplos, además, se deben etiquetar (automáticamente) las imágenes en ciertas **categorías**.

6.4. Análisis de color.

- **Ejemplo 1**. Base de datos de imágenes.

6.4. Análisis de color.

- Añadimos a la BD consultas del tipo: "buscar las imágenes que estén más relacionadas con una dada, **A**".
- **Proceso**:
  - Usar **histogramas 3D** de los canales (R,G,B), con 32 celdas por dimensión.
  - **Normalizamos** los histogramas para que la suma total sea 1.
  - Tomamos como medida de similitud el **solapamiento** entre histogramas. Esta medida irá entre 0 y 1 (entre 0% y 100% de solapamiento).
  1) **Calcular los histogramas** de todas las imágenes de la BD y de la imagen **A**.
  2) **Calcular la similitud** de esos histogramas con el de **A**.
  3) **Resultado**: las imágenes de la BD con mayor valor de similitud.

6.4. Análisis de color.

- **Ejemplo 1**.
- **Consulta**: buscar las 3 imágenes más relacionadas con...

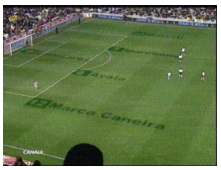


Imagen de entrada **A**

- **Respuestas**:

Respuesta 1
Similitud: 33%

Respuesta 2
Similitud: 32%

Respuesta 3
Similitud: 29%

6.4. Análisis de color.

- **Ejemplo 2**.
- **Consulta**: buscar las 3 imágenes más relacionadas con...

Imagen de entrada **A**

- **Respuestas**:

Respuesta 1
Similitud: 36%

Respuesta 2
Similitud: 35%

Respuesta 3
Similitud: 34%

6.4. Análisis de color.

- **Ejemplo 3**.
- **Consulta**: buscar las 3 imágenes más relacionadas con...

Imagen de entrada **A**

- **Respuestas**:

Respuesta 1
Similitud: 64%

Respuesta 2
Similitud: 63%

Respuesta 3
Similitud: 62%

6.4. Análisis de color.

- **Ejemplo 4**.
- **Consulta**: buscar las 3 imágenes más relacionadas con...

Imagen de entrada **A**

- **Respuestas**:

Respuesta 1
Similitud: 36%

Respuesta 2
Similitud: 33%

Respuesta 3
Similitud: 32%

6. Análisis de imágenes.

Conclusiones:

- Existen otras muchas técnicas de **análisis de imágenes**, muchas de ellas diseñadas para problemas específicos.
- Simplemente hemos dado una **breve perspectiva** de los principales **objetivos** (detección, seguimiento, reconocimiento, etc.) y algunas **técnicas** para conseguirlos (*template matching*, integrales proyectivas, etc.).
- **Disyuntiva**: técnicas genéricas ↔ soluciones *ad hoc*.
- El análisis de imágenes es la base de la **visión artificial**, cuyo objetivo es la comprensión de la información visual.
- Además, la visión artificial incorpora técnicas más propias de la **inteligencia artificial**: reconocimiento de patrones, clasificación supervisada y no supervisada, toma de decisiones, representación del conocimiento, etc.

Anexo A.6.
Análisis de imágenes en OpenCV.

- Búsqueda de patrones
- Flujo óptico
- Integrales proyectivas
- Reproyección y comparación de histogramas

A.6. Análisis de imágenes en OpenCV.

- OpenCV incluye muchas funciones relacionadas con el **análisis de imágenes** y la **visión artificial**, algunas básicas y otras mucho más avanzadas.
- No obstante, no vamos a profundizar mucho, puesto que se sale de los objetivos de la asignatura. Nos limitaremos a las técnicas **estudiadas en este tema**:
  - Búsqueda de patrones: **cvMatchTemplate**
  - Flujo óptico: **cvCalcOpticalFlowBM**
  - Integrales proyectivas: **cvIntegral**
  - Análisis de color: **cvCalcBackProject**, **cvCompareHist**
- En "cvaux" se puede encontrar alguna funcionalidad adicional y experimental de análisis y reconocimiento de patrones.

A.6. Análisis de imágenes en OpenCV.

Búsqueda de patrones

- La función **cvMatchTemplate** de OpenCV permite realizar la búsqueda de patrones por *template matching*: dado un patrón y una imagen, calcular el mapa de matching.
- Distintas medidas de distancia:
  - CV_TM_SQDIFF: suma de diferencias al cuadrado.
  - CV_TM_SQDIFF_NORMED: diferencias al cuadrado normalizadas.
  - CV_TM_CCORR: producto vectorial de los 2 trozos de imagen.
  - CV_TM_CCORR_NORMED: producto vectorial normalizado.
  - CV_TM_CCOEFF: producto vectorial de los patrones "centrados" (correlación).
  - CV_TM_CCOEFF_NORMED: correlación normalizada.

A.6. Análisis de imágenes en OpenCV.

void **cvMatchTemplate** (const CvArr* img, const CvArr* templ, CvArr* result, int method)

- **Significado**: buscar el patrón **templ** en la imagen **img** usando la medida de distancia dada en **method**, y almacenando el resultado en la imagen **result**.
- **Ojo** con las restricciones de tamaños y tipos de datos:
  - Las imágenes **img** y **templ** deben ser de 1 solo canal, de 8 bits de profundidad o bien reales de 32 bits.
  - La imagen **templ** debe ser más pequeña que la **img**.
  - La imagen **result** debe ser de 1 solo canal y necesariamente de reales de 32 bits.
  - Si **img** es de tamaño WxH y **templ** de wxh, **result** debe ser de tamaño (W-w+1)x(H-h+1).

A.6. Análisis de imágenes en OpenCV.

- En imágenes en color, dos opciones:
  - Convertirlas a escala de grises (**cvCvtColor**) y trabajar en gris.
  - Separar los canales (**cvSplit**), aplicar el matching a cada canal y sumar los resultados (**cvAdd**).
- Con CV_TM_SQDIFF, el mejor valor de matching se encontrará en la posición del mínimo.
- Con CV_TM_CCORR y CV_TM_CCOEFF, el mejor matching será el máximo.
- Para buscar máximos y mínimos puede ser interesante usar la función **cvMinMaxLoc**.

void **cvMinMaxLoc** (const CvArr* A, double* minVal, double* maxVal, CvPoint* minLoc, CvPoint* maxLoc, const CvArr* mask=0)

  - **minVal**, **maxVal**: mínimo y máximo global en **A**, respectivamente.
  - **minLoc**, **maxLoc**: posición (x,y) del mínimo y del máximo en **A**.

A.6. Análisis de imágenes en OpenCV.

Flujo óptico

- Existen muchas funciones relacionadas con el análisis de movimiento en la sección *"Motion Analysis and Object Tracking"* dentro de la parte CV de OpenCV.
- Hay que tener ciertos conocimientos sobre las técnicas para poder hacerlas funcionar adecuadamente, así que nos limitaremos a la estudiada en el tema.
- La función **cvCalcOpticalFlowBM** permite calcular el flujo óptico entre dos imágenes (en escala de grises) aplicando la técnica del matching de bloques.
- Otras funciones para calcular el flujo óptico: cvCalcOpticalFlowHS, cvCalcOpticalFlowLK, cvCalcOpticalFlowPyrLK.

A.6. Análisis de imágenes en OpenCV.

void **cvCalcOpticalFlowBM** (const CvArr* imgA, const CvArr* imgB, CvSize blockSize, CvSize shiftSize, CvSize maxRange, int usePrevious, CvArr* velx, CvArr* vely)

- **imgA**, **imgB**: imágenes de entrada. Deben ser de 8 bits y 1 solo canal. Ambas de igual tamaño.
- **velx**, **vely**: flujo óptico resultante. Son también dos imágenes, de 1 canal con profundidad real de 32 bits.
- **blockSize**: tamaño del bloque que se busca. Recordar que el tipo **CvSize** es un struct con los campos: **width** y **height**.
- **maxRange**: radio de búsqueda del bloque.
- **shiftSize** (no visto en el tema): usar cvSize(1,1) para buscar el bloque en todo el radio de búsqueda; cvSize(2,2) para buscar en saltos de 2 en X e Y; cvSize(3,3) en saltos de 3, etc.
- **usePrevious**: si es true, **velx** y **vely** se toman como parámetros de entrada. Se buscan los bloques partiendo de las posiciones previas.

A.6. Análisis de imágenes en OpenCV.

- Si las imágenes **imgA** e **imgB** son de WxH y el tamaño de bloque es de wxh, las imágenes **velx** y **vely** deben ser de tamaño: $(\lceil W/w \rceil)x(\lceil H/h \rceil)$. Ojo, está mal en la ayuda.

- **Algunas indicaciones**:
  - En general, el cálculo del flujo óptico será un **proceso muy lento**. Puede ser adecuado reducir el tamaño de las imágenes antes de aplicarles la función.
  - **Ajustar los parámetros** requiere un proceso de ensayo y error. Bloques pequeños: matching poco fiable. Bloques grandes: proceso lento. Radio pequeño: error si la velocidad es mayor. Radio grande: proceso más lento.
  - Descartar los **bloques de los extremos**. En general, serán poco fiables si hay movimiento por esa zona.
  - En los ejemplos del tema se usan imágenes de 320x240, bloques de 21x21, radio de búsqueda 21x21 y **shiftSize** 1x1.

A.6. Análisis de imágenes en OpenCV.

Integrales proyectivas

- OpenCV no incluye, implícitamente, funciones para el manejo de integrales proyectivas, aunque se pueden usar matrices **CvMat** (o incluso imágenes **IplImage**) para almacenarlas y manipularlas.
- En cuanto al cálculo de las proyecciones, puede ser interesante la función **cvIntegral**.

void **cvIntegral** (const CvArr* I, CvArr* S, CvArr* Sq=0, CvArr* T=0)

- Dada la imagen **I**, calcula en **S** una imagen en la que cada píxel (x,y) es la suma de los píxeles (x',y') con x'<x, y'<y.
- **Sq** contiene la suma de los mismos píxeles al cuadrado.
- Si **I** es de WxH, **S** y **Sq** son de (W+1)x(H+1).
- **Ejemplo**. Para calcular la proyección vertical $\mathbf{P_V}$ haríamos: $P_V(y) = S(W-1, y+1) - S(W-1, y), \forall\ y = 0...H-1$

A.6. Análisis de imágenes en OpenCV.

Análisis de color

- Repasar el manejo de histogramas en OpenCV (anexo A.2).
- Algunas funciones adicionales interesantes:
  - **cvCompareHist**: comparar dos histogramas del mismo tipo, usando diferentes medidas de distancia.
  - **cvCalcBackProject**: aplicar la reproyección del histograma sobre una imagen dada.
  - **cvNormalizeHist**: normalizar un histograma. Hace que la suma de todas las celdas sea un valor dado. Es adecuado aplicarlo antes de llamar a las dos funciones anteriores. Por ejemplo, normalizar a valor 1.
  - **cvThreshHist**: umbralizar un histograma, con cierto valor **u**. Todas las celdas que sean menores que **u** se ponen a 0.
  - **cvGetMinMaxHistValue**: obtener el valor máximo y mínimo del histograma.

A.6. Análisis de imágenes en OpenCV.

double **cvCompareHist** (CvHistogram* H1, CvHistogram* H2, CvCompareMethod method)

- Compara los histogramas **H1** y **H2** y devuelve la medida de distancia o similitud entre ellos.
- Los dos histogramas deben ser del mismo tipo (número de dimensiones y celdas por dimensión).
- El parámetro **method** indica el tipo de medida de distancia:
  - CV_COMP_CORREL: correlación entre histogramas.
  - CV_COMP_CHISQR: suma de diferencias.
  - CV_COMP_INTERSECT: solapamiento o intersección entre los histogramas.
- Con CV_COMP_CORREL y CV_COMP_INTERSECT buscar máximos. Con CV_COMP_CHISQR buscar mínimos.

A.6. Análisis de imágenes en OpenCV.

void **cvCalcBackProject** (IplImage** img, CvArr* backProject, const CvHistogram* hist)

- Calcula la reproyección de la imagen dada en **img** con el histograma **hist**, guardando el resultado en **backProject**.
- La imagen de entrada, **img**, está dada como un array de IplImage*, de imágenes con un solo canal. Deben haber tantas imágenes en **img** como número de dimensiones del histograma.
- El resultado, **backProject**, debe ser también de un solo canal.
- Antes de aplicar la función, puede ser conveniente normalizar y/o umbralizar el histograma.

A.6. Análisis de imágenes en OpenCV.

- **Ejemplo 1**. Detectar y señalar caras humanas en imágenes capturadas de cámara (o bien de archivo AVI).

```
char *archivo= "C:\\OpenCV2.0\\data\\haarcascades" \
                "\\haarcascade_frontalface_default.xml";
  CvCapture* capture= cvCaptureFromCAM(0);  // O bien: cvCaptureFromFile("nombre.avi");
  CvHaarClassifierCascade* cascade= (CvHaarClassifierCascade*) cvLoad(archivo);
  if (!capture || !cascade) return;
  CvMemStorage* storage= cvCreateMemStorage();
  IplImage *img= cvQueryFrame(capture), *img2= NULL;
  while (img && cvWaitKey(5)==-1) {
      if (!img2) img2= cvCreateImage(cvGetSize(img), img->depth, img->nChannels);
      if (img->origin==0) cvCopy(img, img2);
      else cvFlip(img, img2);
      CvSeq* caras= cvHaarDetectObjects(img2, cascade, storage, 1.1, 2,
                        CV_HAAR_DO_CANNY_PRUNING, cvSize(40, 40));
      for (int i= 0; i<(caras?caras->total:0); i++) {
          CvRect* r= (CvRect*)cvGetSeqElem(caras, i);
          cvRectangle(img2, cvPoint(r->x,r->y), cvPoint(r->x+r->width,r->y+r->height),
                        CV_RGB(255,0,0), 3);
      }
      cvNamedWindow("Resultado", 0);
      cvShowImage("Resultado", img2);
      img= cvQueryFrame(capture);
  }
  cvReleaseImage(&img2);
  cvReleaseCapture(&capture);
  cvReleaseHaarClassifierCascade(&cascade);
```


A.6. Análisis de imágenes en OpenCV.

- **Ejemplo 2**. Buscar por *template matching* las apariciones más probables de un patrón (“patron.bmp”) en una imagen (“img.jpg”).

```
int umbral= 50;
IplImage *img= NULL, *pat= NULL, *map= NULL;

void ontrack (int pbar)
{
  double minVal, maxVal;
  CvPoint minLoc, maxLoc;
  IplImage *img2= cvCloneImage(img);
  IplImage *map2= cvCloneImage(map);
  cvMinMaxLoc(map2, &minVal, &maxVal,
              &minLoc, &maxLoc);
  while (maxVal>umbral/100.0) {
    cvRectangle(img2, maxLoc,
          cvPoint(maxLoc.x+pat->width,
          maxLoc.y+pat->height), CV_RGB(255,0,0),
3);
    cvRectangle(map2, cvPoint(maxLoc.x-pat->width,
          maxLoc.y-pat->height), cvPoint(
          maxLoc.x + pat->width, maxLoc.y+
          pat->height), cvScalarAll(0), -1);
    cvMinMaxLoc(map2, &minVal, &maxVal,
          &minLoc, &maxLoc);
  }
  cvShowImage("Deteccion", img2);
  cvReleaseImage(&map2);
  cvReleaseImage(&img2);
}
```

Se supone que hemos creado un botón **pushButton** y definimos su slot

```
void MainWindow::on_pushButton_clicked()
{
   IplImage *imgris= cvLoadImage("imagen.jpg", 0);
   img= cvLoadImage("iamgen.jpg", 1);
   pat= cvLoadImage("pat.jpg", 0);
   map= cvCreateImage(cvSize(imgris->width-
            pat->width+1, imgris->height-pat->height+1),
            IPL_DEPTH_32F, 1);
   cvNamedWindow("Deteccion", 0);
   cvShowImage("Deteccion", img);
   cvMatchTemplate(imgris, pat, map,
            CV_TM_CCOEFF_NORMED);
   cvCreateTrackbar("Umbral", "Deteccion", &umbral,
100, ontrack);
   ontrack(50);
   cvReleaseImage(&imgris);
}
```