# Heuristic-based Automatic Face Detection

Oscar Mañas Sanchez
Víctor Antón Domingas

## Problema a resolver

El problema que tratamos de resolver es el de detectar caras humanas en una imagen digital. La detección de caras es útil en numerosos campos, y es el primer paso de aplicaciones muy importantes como: reconocimiento facial, interfaz hombre-computador, y vigilancia. El rendimiento de estas aplicaciones está limitado por su capacidad de detectar caras de manera rápida y precisa. Esta tarea es muy costosa ya que cada cara es diferente y pueden variar muchos factores, como la expresión, la postura, las condiciones de iluminación o incluso la capacidad del dispositivo usado para obtener la imágen. Ahora bien, si conseguimos implementar este primer paso de manera eficiente, esto permitirá que otros algoritmos que necesiten este primer paso de detección facial se ejecuten más rápidamente, de modo que podrán dedicar más tiempo a partes más importantes del algoritmo.

Para implementar el algoritmo hemos seguido un artículo científico del año 2003, en el que se explica un método de bajo coste para detectar caras en una imagen. Este método se basa en varios heurísticos para detectar caras en imágenes en escala de grises con fondos complejos. El método se divide en dos etapas; la primera etapa busca posibles regiones de la imagen que podrían ser caras y la segunda etapa determina si una posible cara es realmente una cara usando dos discriminadores. Des estos dos discriminadores, nosotros solo hemos implementado el segundo, ya que el paper no especifica detalles de implementación y con la información de la que disponíamos no pudimos llegar a hacer que el primero funcionara correctamente. Usando dos pasos, uno simple y otro complejo, se busca optimizar el uso de los recursos, y hacer que se apliquen sólo allí donde es necesario.

Cabe destacar que el método que presentamos funciona con imágenes en escala de grises y detecta caras en posición frontal y con unas condiciones uniformes de iluminación (sin sombras duras). Ahora bien, la entrada del algoritmo puede ser perfectamente una imagen en color, ya que también hemos implementado la conversión a escala de grises.

Como hemos avanzado, el método usa dos etapas. La primera etapa busca posibles caras en la imágen usando un heurístico sencillo: *En una cara, la intensidad promedio de los ojos es menor que la intensidad de la parte de la nariz que queda entre los ojos.* La segunda etapa toma como entrada la salida de la primera etapa, es decir, toma las regiones con posibles caras y determina si una posible cara es realmente una cara. El discriminador que implementamos se basa en la detección de aristas de la posible cara.

El algoritmo que hemos implementado aplica el método anterior a una región de la imagen (de ahora en adelante window). La idea básica es ir aplicando los diferentes filtros y heurísticos del método explicado anteriormente a la window, de modo que si consigue pasar todas las fases se considerará que ahí hay una cara. De lo contrario, la window se descarta.

Aplicado a una imagen entera, el procedimiento consiste en dividir la imagen en muchas windows (cuantas más, mayor precisión), y aplicar el método a cada una de las windows. También lo podríamos imaginar como una “sliding window”; es decir, tenemos una

window de un determinado tamaño y la vamos desplazando por toda la imagen determinando si lo que hay debajo es una cara. Y esto lo repetimos para cada tamaño de window. Finalmente, dibujamos en color verde todas aquellas windows que han pasado la criba, es decir, que supuestamente contienen una cara.

A continuación, profundizaremos en el funcionamiento de cada etapa del método.

## La primera etapa

Los ojos son una parte muy importante de una cara, muestran cada emoción, pero no cambian drásticamente, como sí pasa con la boca. Usamos este principio para centrar toda nuestra atención en los ojos porque tiene una baja variabilidad. Sería difícil hacer una búsqueda de los ojos en la imagen de una cara en alta resolución; eso es más bien una tarea de reconocimiento de objetos que necesita muchos recursos. Por este motivo, es conveniente trabajar con las imágenes de caras en baja resolución, ya que así solo se preserva la información más relevante.

Se propone el siguiente heurístico:

*En una cara con iluminación uniforme, la intensidad promedio de los ojos es menor que la intensidad de la parte de la nariz situada entre los ojos.*

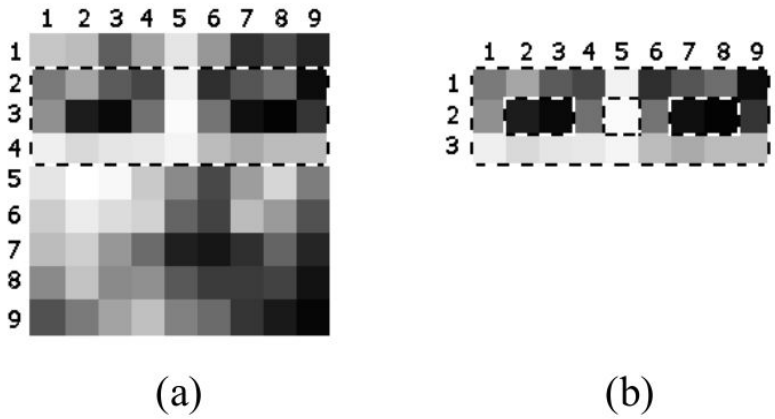
Para aplicar este heurístico, hacemos un reescalado de la window a un tamaño de 9x9 píxeles y a continuación aplicamos un método llamado *histogram equalization* para aumentar el contraste de la window, incrementando así las diferencias de intensidad entre píxeles.

En un tamaño de 9x9 es difícil determinar muchas de las características de la cara, pero este tamaño es suficiente para ubicar los ojos. En este tamaño, es fácil ver que los ojos son de aproximadamente 2x1 píxeles y son más oscuros que la parte de la nariz que queda entre los ojos, correspondiente al píxel central.

Así, la región de los ojos en una window de 9x9 corresponde a las filas 2, 3 y 4. Entonces, la región de los ojos tiene un tamaño de 9x3 (véase la imagen de más abajo). La ecuación para calcular el valor del heurístico de la window, *ENdif*, se muestra a continuación, donde *I* es la intensidad del píxel:

$$ENdif = I_{5,2} - \left(I_{2,2} + I_{3,2} + I_{7,2} + I_{8,2}\right)/4$$

En nuestro caso hemos añadido también la región correspondiente a la boca, ya que comprobamos experimentalmente que esto daba mejores resultados.




(a) Región de la cara de tamaño 9x9 píxeles; y la región de los ojos (encuadrada con una línea discontinua) corresponde a las filas 2, 3 y 4.
(b) Región de los ojos tomada de (a).

Finalmente, si el resultado del heurístico está por debajo de un cierto umbral (determinado experimentalmente), se considera que en aquella región podría haber una cara. Entonces esta window pasa a la segunda etapa.

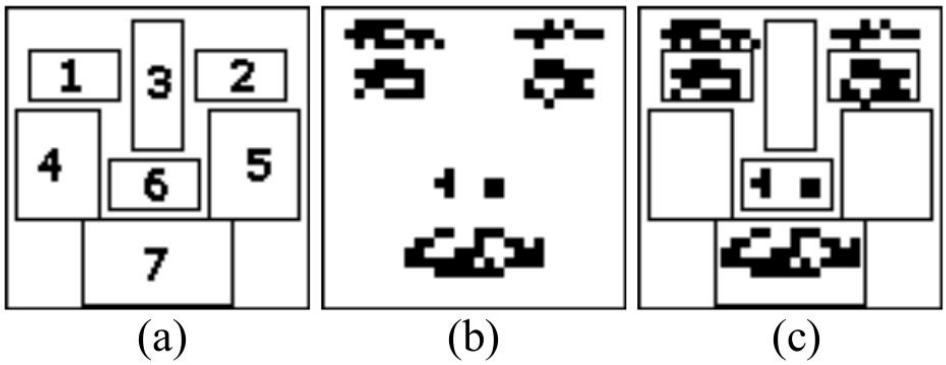
**La segunda etapa**

Sabemos que una cara tiene ojos, nariz, boca, cejas y que está cubierta por piel, y que algunos elementos de la cara son más oscuros que otros. La idea es aprovechar este invariante para comparar la intensidad en las regiones donde debería haber estos elementos y así determinar si la window realmente contiene una cara. Nótese que esta segunda etapa es parecida a la anterior, la diferencia es que en esta consultamos más píxeles y comparamos más regiones, con lo cual el coste es mayor pero también es más precisa. La buena noticia es que aplicaremos esta segunda etapa a muchas menos regiones que la primera etapa, pues el primer heurístico ya habrá descartado algunas windows donde podemos estar seguros de que no hay una cara.

El primer paso de esta segunda etapa es segmentar la imagen usando el conocido método de Sobel para encontrar aristas en dirección horizontal. La imagen resultante contendrá solo los ojos, cejas, boca y nariz. Usamos una máscara simple para evaluar la presencia o ausencia de estos elementos (véase la imagen de más abajo).

La imagen se reescala a 30x30 píxeles, que es el tamaño de la máscara. La máscara se divide en 7 regiones correspondientes a los elementos de la cara. Las regiones 1 y 2 corresponden a los ojos, 3 y 6 corresponden a la nariz, 4 y 5 corresponden a las mejillas y la 7 corresponde a la boca. Para cada región, se calcula la intensidad promedio de los píxeles contenidos en la región. En el caso de 1, 2, 6 y 7 el valor esperado debería estar cerca del 0 (intensidad baja, zona oscura), y en el caso de 3, 4 y 5 el valor esperado debería estar cerca del 255 (intensidad alta, zona brillante). También se evalúa la

simetría entre las regiones 1 y 2, y las regiones 4 y 5, comparando sus valores representativos.




(a) La máscara usada en la segunda etapa.
(b) Resultado de la región de 30x30 píxeles después de aplicar detección de aristas.
(c) Usando la máscara para evaluar si la posible cara es realmente una cara.

Así pues, para cada región se calcula la diferencia con el valor esperado y se acumulan las diferencias de cada región. Si la diferencia total está por debajo de un cierto umbral (determinado experimentalmente), se considera que la región contiene una cara.

Cuando todo el proceso termina, normalmente se detecta una cara con windows de diferentes tamaños, pero solo se necesita una window por cara. Una ampliación del algoritmo consistiría en hacer clustering de las detecciones solapadas, y dejar aquella window con mejores valores en los heurísticos. Esto se aplicaría a cada cara.

## Versiones de código CUDA

## Rendimiento CUDA

## Referencias

El **plazo máximo de entrega del proyecto es el 20 de junio de 2016**.
La entrega hay que hacerla por mail a agustin@ac.upc.edu o botero@ac.upc.edu . Además del informe, en la entrega del proyecto deberéis incluir las versiones implementadas (descritas en 2), los makefiles y los job.sh. Todo ello comprimido en un solo zip.
El informe debe contener los siguientes apartados:

0) Portada **(1 pág.)**. Nombre y apellidos de los integrantes del grupo y título del proyecto.

1) Definición del problema a resolver **(1-4 pág.)**. Documentación (referencias bibliográficas) utilizada para entender el problema y realizar el código secuencial. Si el código secuencial no lo habéis desarrollado vosotros indicad la fuente.

2) Cantidad de implementaciones CUDA realizadas (versiones del código) y descripción de los aspectos más relevantes de cada una (principales diferencias respecto al resto de versiones implementadas). Indicad para cada versión el nombre del fichero que tiene el fichero fuente **(1-4 págs.)**

3) Analizad el rendimiento obtenido para cada implementación CUDA realizada **(1-2 págs.)**. Incluid alguna medida de rendimiento para evaluar las mejoras introducidas (GFLOPS, ancho de banda, tiempo, …). Si se observa alguna inconsistencia en los resultados indicad posible causas. Se recomienda realizar varias ejecuciones por cada prueba y obtener una media. Recordad que el código secuencial debe ejecutarse en el mismo servidor que el código CUDA.