UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

PROYECTO FIN DE CARRERA

ALGORITMOS DE DETECCIÓN DE IMÁGENES RUIDOSAS Y DUPLICADAS

Irene Blasco Hernanz

DICIEMBRE 2008

ALGORITMOS DE DETECCIÓN DE IMÁGENES RUIDOSAS Y DUPLICADAS

AUTORA: Irene Blasco Hernanz
TUTOR: José María Martínez Sánchez

Video Processing and Understanding Lab
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Diciembre de 2008

Palabras clave

Algoritmos, calidad, borrosidad, exposición, homogeneidad, duplicados.

Keywords

Algorithms, quality, blur, exposition, homogeneity, duplicates.

Resumen

El objetivo de este proyecto ha consistido en el estudio de una serie de algoritmos que, aplicados sobre imágenes, midieran la calidad de éstas y, comparándolas, fuesen capaces de detectar conjuntos de duplicados.

Entre las medidas que afectan a la calidad, se han tenido en cuenta aspectos tan primordiales como la exposición de la fotografía, su borrosidad o su homogeneidad. Dentro de la clasificación por duplicados, se ha intentado encontrar la representación que mejor incorporase las características de cada imagen, sin penalizar por ello los tiempos de ejecución. Persiguiendo esta finalidad, se ha procurado adaptar algunos de estos algoritmos que funcionaban de nivel de píxel, a nivel de bloque DCT, además de desarrollar nuevos métodos que no necesitasen la decodificación completa de las imágenes para trabajar.

Para verificar la utilidad de los algoritmos propuestos se ha desarrollado una aplicación consistente en un repositorio de imágenes, que clasifica las fotografías según fecha de toma, calidad y duplicados. También se ha implementado la versión gráfica de esta aplicación, que permite una visualización completa de la disposición de las fotografías, además de los valores que las caracterizan y su representación de duplicados.

Finalmente, se ha destinado parte del trabajo realizado a un programa que, trabajando con secuencias de vídeo, informa al usuario de los *frames* que considera de baja calidad y de los que contienen elementos concretos (claquetas, cortinillas, cartas de ajuste…), y advierte de la posibilidad de que la cámara se haya quedado grabando sola. Esta aplicación tiene como fin ser integrada en el sistema IVOnLA.

Abstract

The objective of this project has lain in the study of a series of algorithms that measure the quality of images and decide about its inclusion or not in a group of duplicates.

Among the quality measurements, we point out features such as the exposition of the photograph, its blur or its homogeneity. Involving duplicates, the objective has focused towards a representation which fits the best the characteristics of each image, without penalizing the execution times. Pursuing this aim, we have adapted some of the algorithms from the pixel level to the DCT block level. Besides we have developed new methods, irrespective the whole decoding of the image.

The proposed algorithms are verified within an image repository that classifies the photographs according to the date they were taken, its quality and duplicates. The interface of this application allows a complete visualization of the image disposition, its values and representation.

Finally, an important part of the study has been assigned to a video program, to report on the user which frames have low-quality and which of them contain clapperboards, break bumpers, test patterns… It also warns about the presence of sequences where the camera seems to be filming on its own. This application is destined to be integrated in the IVOnLA system.

Agradecimientos

Primeramente, quiero agradecer a mis padres el cariño y la comprensión que han tenido conmigo durante la realización de este proyecto, y durante la carrera en general. A ellos les debo mis motivaciones y mis ganas de mejorar.

También quiero dar las gracias a mi tutor por tener paciencia, por enseñarme tanto y hasta por comportarse a veces como ese "consejero" tan famoso.

Se merecen una mención especial todos esos compañeros de los que he disfrutado durante mis días en el laboratorio. Gracias a Víctor F., por todas las horas que se ha tirado delante de mi ordenador resolviendo errores que eran imposibles de solucionar para mí y por sus tutoriales, que me salvaron la vida; a Javi, especialmente por su clase magistral sobre librerías, sus explicaciones y su ironía; a Víctor V., por su ayuda y, sobre todo, por meterse tanto con Javi; a Fabri, por toda su colaboración y por enseñarme a bailar chachachá; a Luis, por tener siempre una solución; a Ana, por preguntar y por animarme tanto; a Juan Carlos, por su Dominant Color y por hacernos ver que se puede llegar ahí; a Álvaro, por su búsqueda de la verdad sobre los coeficientes DCT; a Marcos y a Álvaro, por todos esos buenos ratos de desconexión (y lo buenos amigos que habéis sido a lo largo de la carrera). ¡Cómo os voy a echar de menos!

Además, gracias a todos los que me habéis apoyado y animado cuando me han faltado fuerzas, sois los mejores.

INDICE DE CONTENIDOS

INDICE DE FIGURAS

INDICE DE TABLAS

1 Introducción

1.1 Motivación

Se considera imagen ruidosa toda aquella cuya información no sea útil para el receptor, en este caso por deficiente en calidad. De esta forma se puede definir una imagen defectuosa como aquella con un exceso de homogeneidad (fotografías orientadas al suelo, al techo...), sobre/sub exposición, ausencia de nitidez, etc. Pero existen a su vez otro tipo de imágenes que, a pesar de no ser imperfectas, tampoco ofrecen información extra al destinatario: las imágenes duplicadas. Los mínimos cambios que se dan entre figuras de este tipo las hacen idénticas o casi idénticas, con lo que deshacerse de las de menor calidad no supone pérdida de información al sistema.

Imagen borrosa

Imagen subexpuesta

FIGURA 1-1: Ejemplo de imágenes ruidosas

FIGURA 1-2: Ejemplo de imágenes duplicadas

Los algoritmos de detección de imágenes ruidosas y duplicadas se vienen utilizando en proyectos en los que es necesario hacer un filtrado previo de imágenes con información insuficiente o redundante, con el fin de no cargar innecesariamente los algoritmos de procesado posteriores.

El programa *Photo2Video* [1] y [2] hace uso de estos algoritmos para eliminar previamente las imágenes borrosas, homogéneas, sobre/sub expuestas o duplicadas que no son útiles para la secuencia de vídeo final. De la misma forma, en [3], las fotografías que se consideran ruidosas no se procesan y sólo las que pasan cierto umbral de calidad se incluyen en la presentación con música. Si bien estos artículos utilizan estas técnicas para un prefiltrado previo a su aplicación específica, es inmediato ver su utilidad en otras

aplicaciones de acceso a bases de datos de imágenes, como por ejemplo, álbumes de fotos, creación de repositorios para su posterior consulta, etc.

Aplicados a vídeo, estos algoritmos permiten eliminar *frames* borrosos, homogéneos, claquetas, secuencias en las que la cámara se quedó encendida sola, etc., lo cual resultaría a su vez ventajoso en proyectos de sumarización de vídeo o selección de escenas.

1.2 Objetivos

El objetivo principal de este proyecto es el estudio y desarrollo de algoritmos que sean capaces de determinar la calidad en términos de homogeneidad (imágenes de superficies), exposición (exceso de brillo, falta de luz…) y nitidez (definición de bordes y formas), para posteriormente obtener, mediante una adecuada ponderación de cada uno de ellos (función de calidad), el coeficiente que representará la calidad de la imagen.

A su vez, el desarrollo de métodos que construyan una representación de las imágenes involucradas en el proceso, para su consecuente comparación entre ellas, con la intención de detectar cuáles son duplicadas, así como las distintas medidas entre dichas representaciones, acompañan al objetivo principal de investigar los algoritmos de calidad y la función que los pesa.

Para empezar, se trabajará a nivel de píxel, estudiando algunos de los métodos propuestos en artículos, para seguidamente continuar a nivel de bloque DCT, con el objetivo de conseguir que los algoritmos funcionen más rápido.

La utilidad de los algoritmos seleccionados se pondrá a prueba en un programa repositorio de imágenes que, partiendo de una carpeta de fotografías, las clasificará haciendo uso de ellos. Las fotografías serán primeramente ordenadas en carpetas según la fecha en la que fueron tomadas y a continuación separadas en concordancia con su calidad. Las que superen el umbral (las que sean lo suficientemente buenas), pasarán un último análisis de duplicados, que clasificará por nombre (conforme calidad) las que den positivo.

Por último, el fin es aplicar los conocimientos adquiridos en los pasos anteriores a secuencias de vídeo con el propósito de extraer *frames* o escenas que se caractericen por su baja calidad, así como de eliminar claquetas o segmentos en los que la cámara graba sola.

Por tanto, los objetivos del proyecto se resumen en el estudio y desarrollo de algoritmos de análisis de calidad y detección de duplicados, la obtención de la función de calidad y las técnicas de medida de diferencia entre representaciones de fotografías, la implementación de un repositorio de imágenes que pruebe la eficacia de los algoritmos elegidos y, finalmente, la aplicación de las nociones obtenidas sobre los algoritmos para detectar secuencias determinadas en vídeos.

1.3 Organización de la memoria

La memoria consta de los siguientes capítulos:

- **Capítulo 1:** Introducción
- **Capítulo 2:** Estado del arte
- **Capítulo 3:** Diseño
- **Capítulo 4:** Desarrollo
- **Capítulo 5:** Integración y resultados
- **Capítulo 6:** Conclusiones y trabajo futuro

Y de los anexos listados a continuación:

- **Anexo A:** Manual del programador
- **Anexo B:** Wavelets
- **Anexo C:** Espacios de color
- **Anexo D:** JPEG
- **Anexo E:** MPEG
- **Anexo F:** Etiquetas EXIF
- **Anexo G:** Base de datos de imágenes de entrenamiento

2 Estado del arte

2.1 Introducción

A continuación se describen algunos artículos sobre el análisis de duplicados y la detección de imágenes de baja calidad, independientemente de cuál sea su uso posterior, o si se trata de un paso intermedio entre objetivos.

Este capítulo de "Estado del arte" se divide en dos secciones principales, a fin de responder de una forma más fiel a los contenidos mencionados. En la primera parte, se refieren brevemente los artículos en los que, en general, se hace uso de algoritmos que concuerdan con los objetivos del proyecto. En la segunda, se estudian a fondo un conjunto de los algoritmos citados anteriormente, a fin de discernir cuál de ellos se aproxima mejor a las necesidades de la aplicación (repositorio) que se ha de realizar consecutivamente.

Este estudio se corresponde con la fase del proyecto en la que el objetivo son los algoritmos a nivel de píxel, es decir, los que trabajan una vez que la imagen ha sido completamente decodificada.

2.2 Breve relación de artículos relacionados

Los artículos relacionados se clasifican según el contenido de los métodos que proponen, diferenciando previamente entre algoritmos de filtrado de calidad (análisis de imágenes borrosas, sobre/sub expuestas y homogéneas), función de calidad y detección de duplicados.

2.2.1 Filtrado de calidad

Un filtrado de calidad puede ser usado con diferentes objetivos (clasificación de imágenes, desahogo del sistema posterior de análisis…), pero siempre implica tener en cuenta el mayor número de circunstancias posible. Siguiendo la división de [1], una imagen de baja calidad se particulariza por una o más de estas tres características: borrosidad, exceso/defecto de exposición u homogeneidad.

2.2.1.1 Imágenes borrosas

Una imagen borrosa es aquella cuyos bordes y formas no están claramente definidos. Esta ausencia de bordes es lo que inspira a casi todos los artículos consultados para obtener el factor de nitidez de las imágenes.

En [1], por ejemplo, utilizan la transformada wavelet de Haar propuesta en [4]. En este artículo clasifican los bordes en 4 tipos según su forma: estructura delta, estructura tejado, estructura A y estructura G (figura 2-1) y estudian cómo cambian una vez aplicada la transformada. Según el porcentaje y tipo de bordes que quedan tras la operación se determina el valor del factor de *blur*.

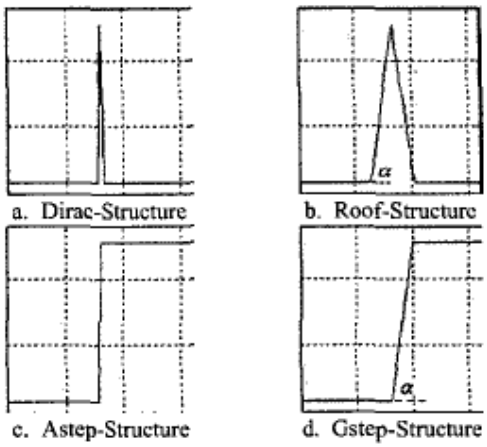


FIGURA 2-1: Tipos de bordes según su forma [4]

De la misma forma, en [11], se detectan los bordes verticales mediante un filtro de Sobel, y la medida de *blur* se relaciona con el ancho de los bordes detectados. El valor final que se adjudica a la imagen es la media del ancho de los bordes detectados.

En [5] la falta de bordes se entiende como la desaparición de las altas frecuencias de la imagen. Una herramienta muy útil para conseguir el factor de *blur* es la DCT, con cuyos coeficientes se elabora un histograma de frecuencias que determina si la imagen es borrosa o no.

Un método diferente a los anteriores es el propuesto en [6], utilizado en [2] y [3], que obtiene el factor de nitidez estimando previamente la función de *blur*. Para conseguirlo supone que el sistema que emborrona la imagen es lineal e invariante y, para los casos de movimiento de cámara o desenfoque, se demuestra que el *cepstrum* de la imagen contiene suficiente información para obtener una medida de *blur*.

2.2.1.2 Imágenes homogéneas

Lo característico de una imagen homogénea es que es prácticamente constante, sin cambio sustancial de colores ni de formas. Tanto en [1] como en [2] y [3] se usa la entropía (medida del desorden) para determinar hasta qué punto es invariante la imagen.

2.2.1.3 Sobre/Baja Exposición

Si el brillo medio de la imagen es demasiado alto o demasiado bajo, la figura se convierte en una imagen homogénea, de manera que los autores de [1], [2] y [3] aprovechan la medida de la entropía para establecer la exposición de la imagen.

2.2.2 Función de calidad

La función de calidad recomendada en los artículos consultados consiste en pesar de igual manera el factor de *blur* que la medida de la entropía.

$$Q(x) = q \cdot CH(x) + (1-q) \cdot BL(x)$$

Siendo *x* una imagen cualquiera, *CH(x)* la entropía de esa imagen y *BL(x)* su factor de calidad de *blur*. El coeficiente *q* se establece en [1] empíricamente como 0.5.

2.2.3 Detección de duplicados

Se considera que dos imágenes son duplicadas si son lo suficientemente parecidas. Normalmente esto quiere decir que los colores y las formas que se encuentran en ellas están distribuidos de forma similar.

Los algoritmos de detección de imágenes duplicadas van a obtener representaciones de las imágenes que permitan una comparación más sencilla y eficiente para tomar la decisión.

En [7], por ejemplo, se establecen primero, para cada imagen, un conjunto de colores dominantes. Luego la imagen se divide en una malla de *mxn* celdas y se le asigna una "firma de color" (describe qué colores dominantes están presentes en la imagen y dónde están situados) que identifica a cada imagen individualmente. Estas "firmas de colores" son las que se comparan con una serie de medidas de distancia y similitud para determinar si las dos imágenes a comparar son duplicadas.

En [8] se desarrolla un algoritmo cuyo objetivo es detectar falsificaciones pero en este caso no se divide la imagen. Primero se definen una serie de puntos de interés (que se corresponden con picos de la imagen) mediante el detector DoG (*Difference of Gaussian*) y se especifica su orientación. Estos puntos son inmunes a rotaciones, degradaciones de la imagen, filtros, cambios de brillo y contraste,… Luego se utilizan funciones Hash para que la comparación entre la base de datos y la imagen sea más rápida, y finalmente se determina si hay acierto, es decir, si hay falsificación.

Un método más sencillo es el que se utiliza en [1], [2] y [3] que consiste en transformar la imagen a otra equivalente en escala de grises de tamaño 8x8, representativa de ésta. La comparación se realiza sólo entre imágenes próximas en el tiempo, mediante la distancia euclídea.

A su vez, en [12] se sugieren dos de los cuatro descriptores de color incluidos en el estándar MPEG-7, Parte 3 [13]. Los descriptores de color crean una imagen representación de la imagen original que es útil para la comparación y la detección de imágenes duplicadas. Los propuestos son *Dominant Color* y *Color Layout*, que se estudiarán en mayor profundidad en la sección a continuación (2.3.2.2 y 2.3.2.3).

2.3 Estudio sobre las técnicas encontradas

En esta sección se describen con más detalle algunos de los algoritmos sugeridos en el apartado anterior, implementados independientemente de su inclusión o exclusión en las aplicaciones objetivos sobre las cuales se harán las pruebas, y que no son las únicas para las cuales estos algoritmos pueden ser de utilidad.

Este estudio se corresponde con la primera fase del proyecto, que se centró en los algoritmos que trabajan a nivel de píxel. Para su funcionamiento se necesita decodificar la imagen hasta la obtención de los valores RGB de cada uno de los píxeles que conforman la misma.

2.3.1 Filtrado de calidad

El filtrado de calidad es necesario para deshacerse de las imágenes que no son lo suficientemente buenas para ser almacenadas para su posterior uso. Con ello también se consigue descargar al sistema posterior de análisis.

2.3.1.1 Imágenes borrosas

En la literatura existen dos propuestas distintas a la hora de evaluar la nitidez de una imagen: métodos directos e indirectos. Los métodos indirectos se concentran en la estimación de la función de *blur* (la responsable de ensuciar la imagen), que convolucionada con la imagen nítida y sumada con la imagen de ruido daría como resultado la imagen borrosa [9].

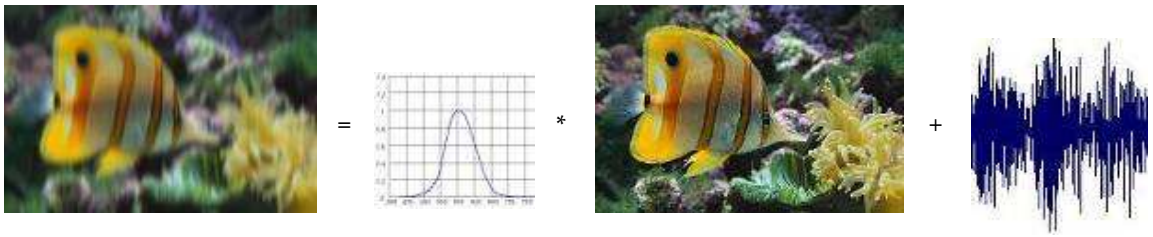


Imagen borrosa **Función de *Blur*** **Imagen nítida** **Ruido**

FIGURA 2-2: Ecuación de *blur*

$$I\_borrosa = F\_blur * I\_nítida + N$$

Aunque la obtención de la función de *blur* permite cuantificar la extensión de la imprecisión de una imagen, es muy costoso computacionalmente, como en [16][17][18].

Los métodos indirectos, por otro lado, intentan determinar el factor de nitidez basándose en características de la imagen, una de las cuales es el "borde". Las imágenes nítidas se identifican por tener sus bordes muy bien definidos, tanto en el primer plano (*foreground*) como en el fondo (*background*) de la imagen. A medida que una imagen pierde definición o cuando se ve afectada por movimientos de cámara o desenfoques, los bordes que la forman se difuminan y se mezclan con el resto de la imagen. Este hecho también provoca una desaparición de las altas frecuencias de la imagen.

Una vez fijada una característica cuantificable presente en imágenes borrosas, se puede establecer un criterio de clasificación y desarrollar unos algoritmos que determinen si una imagen es borrosa o no, y en qué medida.

Así se considera que una imagen está borrosa en el caso de que los objetos que se encuentren en ella tengan sus bordes difusos, poco nítidos. De esta forma, se supondrían subjetivamente imágenes borrosas fotografías del tipo de las expuestas a continuación:

FIGURA 2-3: Ejemplos de imágenes borrosas

De donde se concluye que los desenfoques y los movimientos de cámara se convierten en los principales causantes de borrosidad en una fotografía.

Los algoritmos implementados (explicados a continuación) hacen uso de un método indirecto basándose en la característica de ausencia de bordes de las fotografías borrosas para obtener su factor de nitidez.

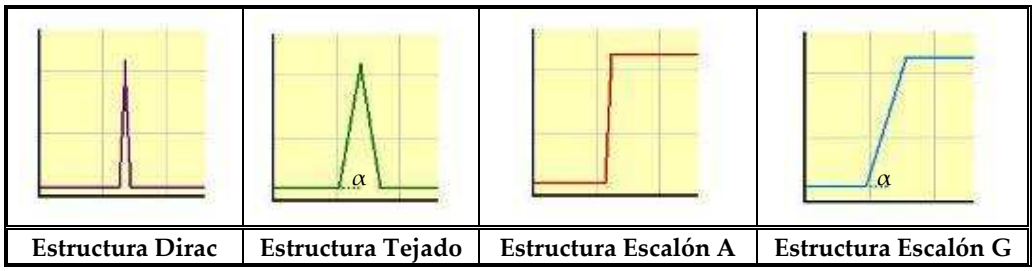
2.3.1.1.1 Transformada de Haar

En 1909 Alfred Haar descubrió un conjunto de funciones ortonormales a partir de las que se podía representar otra función. El hallazgo de Haar no es más que la más simple de las *wavelets* ortogonales: la transformada de Haar. La aplicación de esta transformada permite esencialmente obtener un trazado de los bordes horizontales, verticales y diagonales que se encuentran en una imagen. (Para más información sobre las *wavelets*, véase el anexo B.)

Basándose en varios estudios de bordes y con la ayuda de la transformada de Haar, los autores del *paper* [4], son capaces de determinar si una imagen es borrosa o no y qué extensión de esa imagen no es nítida (*blur extent*).

En estos estudios de bordes se definen los tipos principales que se distinguen y se clasifica su comportamiento frente al efecto borroso y la transformada Haar.

Los tipos de bordes considerados vienen definidos por la estructuras siguientes: la estructura Dirac, la estructura Tejado, la estructura Escalón A y la estructura Escalón G. Cada una de ellas cumple con unas características concretas, a saber: la estructura Dirac representa líneas finas en la imagen; la estructura Tejado se comporta de la misma forma que la Dirac pero en líneas más gruesas, en las que se note la variación progresiva del color o intensidad; por último, las estructuras Escalón representan los cambios de una zona de color o intensidad a otra, siendo la variación más abrupta en el caso de la estructura A y más progresiva en la G. Tanto en la estructura Tejado como en la estructura Escalón G se puede hablar de un ángulo α que representa la pendiente de la curva de variación de color o intensidad.



Estructura Dirac	Estructura Tejado	Estructura Escalón A	Estructura Escalón G

FIGURA 2-4: Tipos de bordes considerados en el algoritmo

Según el estudio de bordes mencionado, los bordes tipo Dirac y Escalón A no existen en imágenes borrosas, (esta afirmación es la que les permite decidir si una imagen es borrosa o no) y los tipo Tejado y Escalón G pasan a tener un ángulo α que tiende a 0, es decir, estos bordes tienden a perder su nitidez.

Una vez conocido el comportamiento de los bordes frente al *blur*, es preciso ver cómo caracteriza la transformada Haar los distintos tipos de bordes para poder extraer conclusiones. A continuación se presentan algunos ejemplos de cómo trabaja la transformada Haar de 3 niveles con imágenes nítidas y borrosas:

FIGURA 2-5: Imagen nítida

FIGURA 2-6: Imagen borrosa

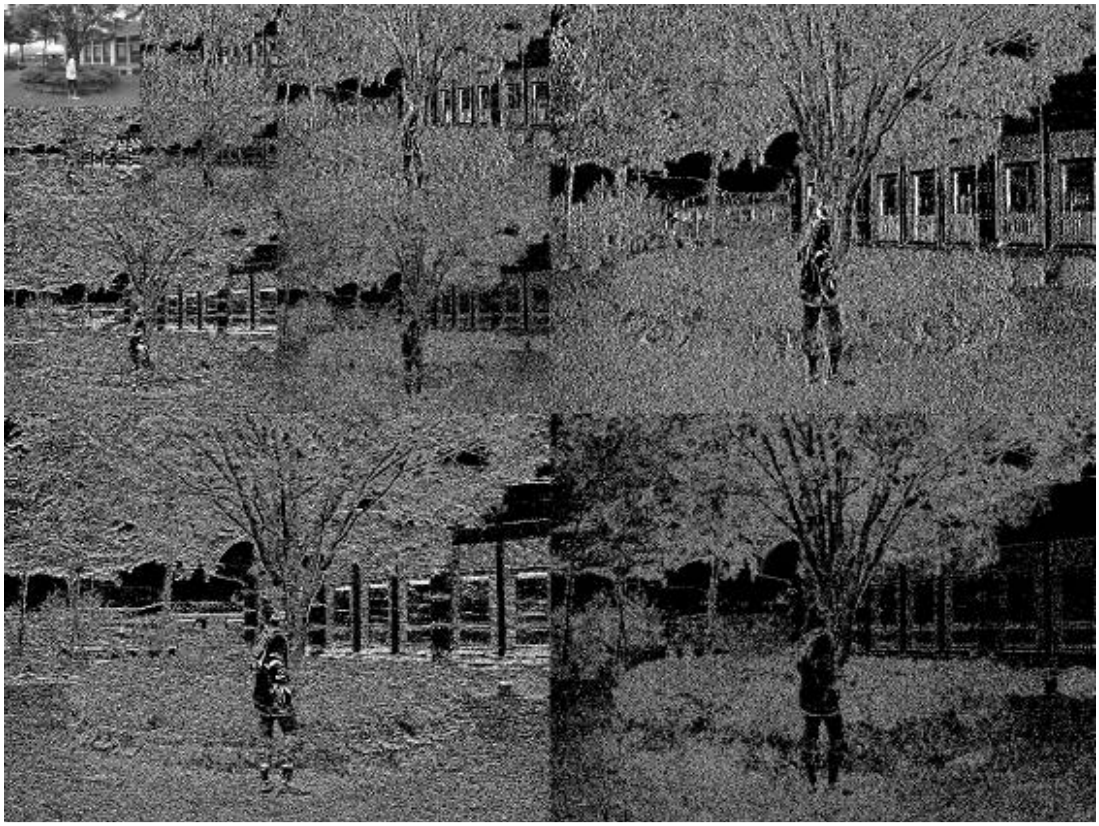


FIGURA 2-7: Transformada Haar de 3 niveles de una imagen nítida

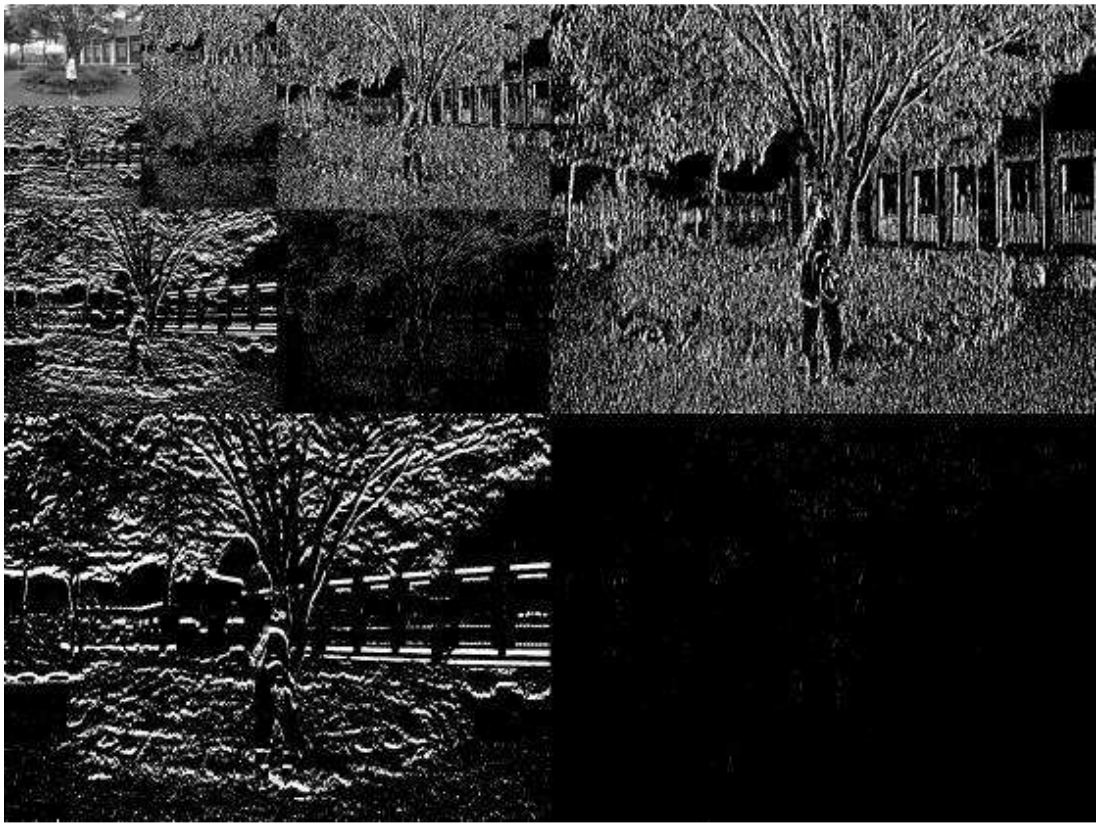


FIGURA 2-8: Transformada Haar de 3 niveles de una imagen borrosa

En la figura 2-5 se pueden considerar bordes tipo Delta, por ejemplo, algunas briznas de hierba, y bordes tipo Escalón A los troncos de los árboles o las columnas del edificio. El detalle de la imagen se mantiene en la imagen de la transformada de Haar (figura 2-7) hasta el tercer nivel, es decir, las estructuras características de las imágenes nítidas son definidas por la transformada en cada uno de los niveles estudiados, si bien son más claros en el primero.

Ahora, si volvemos borrosa la imagen de la figura 2-5 mediante un filtro gaussiano 3x3 y una desviación de 5.0, simulando un desenfoque (figura 2-6), los resultados son los que aparecen en la figura 2-8. Los bordes que antes representaban el tipo Delta y Escalón A han desaparecido por el efecto del desenfoque, y en la imagen de la transformada de Haar de tres niveles sólo se muestran las estructuras tipo Escalón G y Tejado, que se distinguen en el nivel 1 por sus líneas mucho más gruesas. La representación obtenida pierde detalle, que es justamente lo que sucede cuando una imagen se vuelve borrosa.

Para completar el ejemplo se exponen seguidamente las imágenes de las transformadas de Haar de tres niveles de las fotografías que se muestran al inicio de este apartado como modelo de imágenes borrosas.

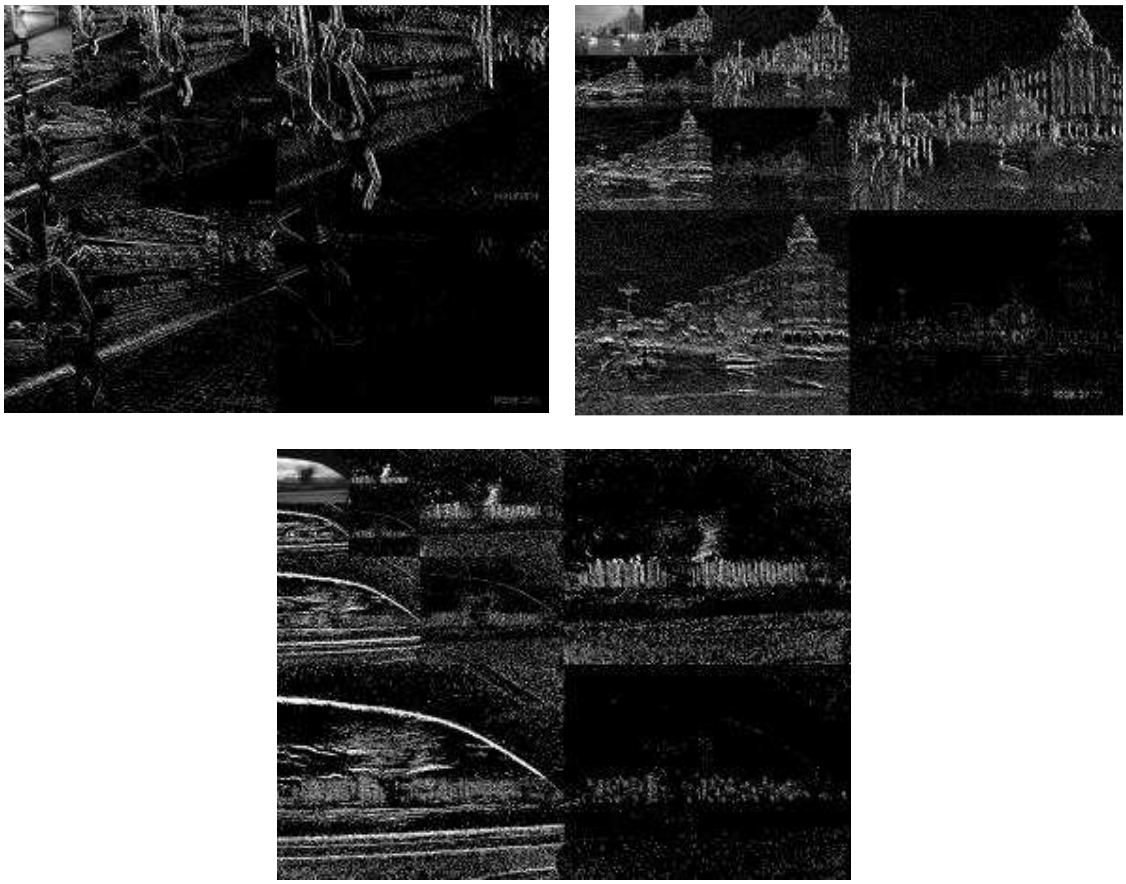


FIGURA 2-9: Ejemplos de transformadas de Haar de 3 niveles de imágenes borrosas

Una vez observado el efecto de la transformada de Haar sobre imágenes nítidas y borrosas, los autores de [4] proponen el trazado de un mapa de bordes para la obtención del valor de la extensión de *blur*.

La imagen resultado derivada de la aplicación de la transformada es una matriz de valores en el rango de 0 (negros) a 255 (blancos). Cada nivel tiene un tamaño diferente y una representación de los bordes verticales, horizontales y diagonales de la imagen. Para construir un mapa de los bordes de cada nivel basta con recoger el valor de la matriz correspondiente al mismo punto de cada nivel y aplicar la siguiente fórmula:

$$Emap_i(k,l) = \sqrt{V_i(k,l)^2 + H_i(k,l)^2 + D_i(k,l)^2}$$

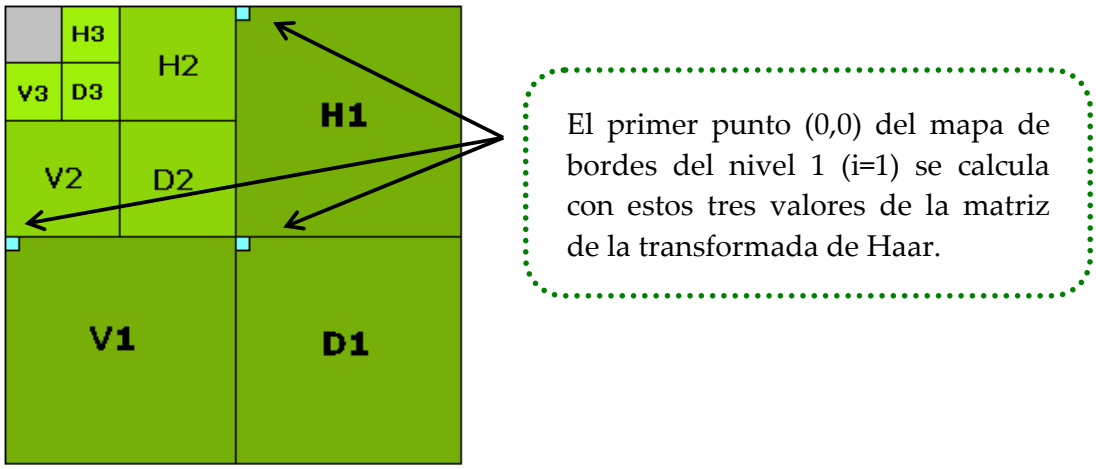


FIGURA 2-10: Estructura piramidal de la transformada de Haar

Pero como los Emaps tienen tamaños diferentes, los valores obtenidos en los Emaps han de ser reducidos mediante el uso de ventanas antes de obtener resultados. El objetivo de la ventana es determinar el valor del máximo local de los puntos de Emap que engloba la ventana, sin que ésta se solape en ningún caso. Precisamente, la ventana de $Emap_1$ tendrá un tamaño de 8x8 celdas, la de $Emap_2$ una de 4x4 celdas y la de $Emap_3$ una de 2x2 celdas, de forma que los Emax que surjan del cálculo tengan todos el mismo tamaño.

Si $Emax_i$ (k,l) supera un cierto umbral, entonces el píxel (k,l) de la imagen se puede considerar un borde.

Según [4], el efecto de este algoritmo sobre los diferentes tipos de bordes es el siguiente:

	Emax1	Emax2	Emax3
Estructura Dirac	Mayor	Medio	Menor
Estructura Escalón A	Mayor	Medio	Menor
Estructura Escalón G	Menor	Medio	Mayor
Estructura Tejado	Menor	Medio	Mayor
	Menor	Mayor	Medio

TABLA 2-1: Valores de $Emax_i$ según el tipo de borde

Como se puede observar, la intensidad de los bordes que definen una imagen nítida disminuye a medida que aumentan los niveles de la transformada, al contrario que lo que sucede en los bordes que se encuentran en las imágenes borrosas, que la transformada tiende a hacer más finos (con mayor intensidad) según aumentan los niveles. Esta es una de las propiedades de la transformada Haar: la capacidad de recuperar la nitidez de los bordes borrosos. El comportamiento de los bordes con respecto al algoritmo propuesto permite al programador emparejar una región de la imagen con un tipo de borde diferente.

La clave para determinar la extensión de *blur* es clasificar qué bordes de las estructuras Tejado y Escalón G parecen pertenecer a una imagen borrosa. Para ello se hace uso de la propiedad de la transformada de Haar, primero buscando los puntos (k,l) en los que $Emax_1 < Emax_2 < Emax_3$ (Tejado o Escalón G), o $Emax_2 > Emax_1$ y $Emax_2 > Emax_3$ (Tejado), que corresponderían con estructuras Tejado y Escalón G en la imagen. Como la transformada convierte este tipo de bordes borrosos en bordes más nítidos según se aumenta de nivel, si dichos puntos (k,l) tienen un $Emax_1$ correspondiente menor que cierto umbral se considera que se encuentran en una imagen borrosa.

La fórmula para obtener la extensión de *blur* de la imagen será, por tanto:

$$BlurExtent = \frac{Nb}{N}$$

Siendo N el número de bordes Tejado y Escalón G encontrados en la imagen y Nb el número de bordes contabilizados en N que se han considerado borrosos.

2.3.1.1.2 DCT

Al igual que las transformadas de Fourier, la transformada discreta del coseno (DCT) es capaz de representar una función como una suma de sinusoides de diferentes frecuencias y amplitudes. Al ser discreta, esta transformada opera de la misma forma que la transformada discreta de Fourier (DFT), con la diferencia de que la DCT utiliza sólo cosenos en su representación, mientras que la primera usa tanto senos como cosenos.

El uso de la DCT para detectar si una imagen es borrosa o no parte del hecho de que una imagen sin bordes carece de altas frecuencias, es decir, sus principales coeficientes AC son en su mayoría cercanos a 0. Como ejemplo se muestran a continuación un par de imágenes (una nítida y una borrosa) y algunos de sus bloques DCT de 8x8 píxeles.

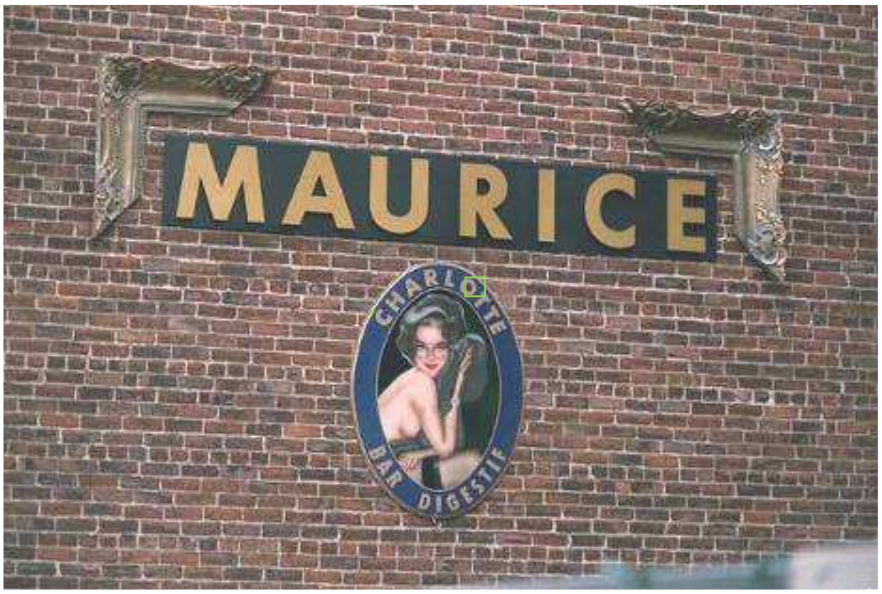


Imagen nítida

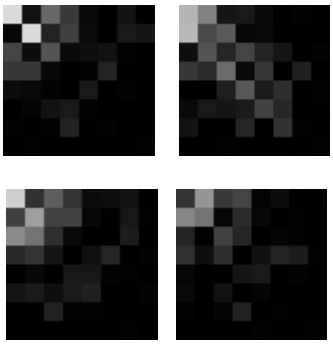
987.87	16.15	-103.98	59.82	-10.12	0.25	-20.11	0.18
10.21	220.13	35.78	-64.24	10.44	0.10	24.02	-22.08
-66.27	-15.11	84.03	-9.72	16.45	-22.8	0.33	-0.31
-53.97	55.93	8.87	0.39	-0.38	34.66	-0.45	0.30
-14.12	-9.28	0.16	-0.14	-27.12	0.04	0.01	0.07
0.54	-0.05	22.22	20.23	0.03	-0.1	0.02	-0.43
-0.28	-0.19	-0.16	34.87	-0.45	-0.46	-0.28	-0.10
-0.45	0.49	-0.21	-0.09	-0.20	-0.01	0.16	0.07

951.75	-127.7	-24.00	-24.13	-10.00	0.07	0.20	-0.14
180.08	75.00	-90.18	-7.9	-9.95	0.26	0.13	-0.26
17.97	84.98	35.78	-70.15	32.07	-23.22	-0.49	-0.38
54.14	-41.90	108.32	11.81	59.80	8.24	-31.9	-0.04
0	-0.38	29.90	88.20	-26.75	-44.08	0.33	-0.17
9.62	-27.38	21.59	26.08	64.04	-41.89	0.06	-0.30
19.88	-0.17	0.50	35.37	-0.29	47.86	-0.28	0.08
-0.03	0.07	0.90	0.38	-0.68	0.02	0.34	-.43

976.00	48.15	-92.30	53.94	-10.25	15.58	-19.67	0.16
64.73	-160.20	-66.22	64.09	-10.29	0.41	-23.80	-0.27
-137.98	-114.86	47.86	10.31	0.04	-0.18	-28.02	0.05
41.48	55.72	18.58	-0.63	20.22	-35.24	-0.02	-0.07
-0.25	-8.89	0.09	22.15	26.50	-0.04	0.23	-0.05
20.26	27.78	21.55	-25.90	-31.80	-0.22	0.52	0.04
0.06	0.15	-31.02	0.42	0.56	0.37	0.13	-0.11
0.07	-0.43	0.11	-0.23	-0.08	-0.12	-0.35	-0.45

820.25	-147.57	-56.04	71.95	-0.50	-15.83	-0.56	0.10
-134.78	115.05	5.95	-40.26	-9.83	-0.15	-0.03	0.30
-35.55	-5.22	65.70	-39.87	0.42	0.29	0.18	0.20
48.04	-14.47	-44.97	-0.34	19.92	35.09	-31.95	0.03
-7.00	-0.11	0.56	22.66	-27.25	0.51	0.04	-0.03
-9.64	0.13	21.80	-0.09	0.18	0.21	0.36	-0.42
0.39	0.36	0.18	-34.84	0.36	-0.32	-0.29	-0.42
0.16	0.12	0.39	-0.53	0.25	0.14	-0.14	0.08

Bloques DCT de la zona marcada en la imagen. Los coeficientes cuyo valor absoluto es menor que 8 se marcan en rojo.



Imágenes correspondientes a los bloques DCT anteriores

FIGURA 2-11: DCT de una imagen nítida

Imagen borrosa

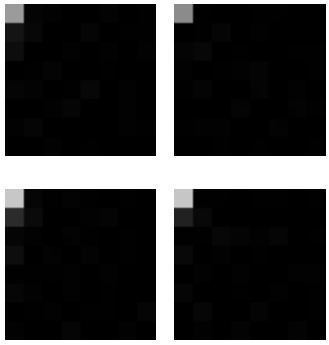
1180.1	3.15	0.33	-0.01	-1.00	2.81	0.14	-0.23
-20.65	-8.28	-0.90	-0.78	-4.37	0.49	-0.04	-0.01
-10.80	0.88	1.21	-0.57	0	0.01	0.35	0.31
-2.78	-0.55	-1.09	-0.05	-0.09	-0.01	0.41	0.29
4.00	-1.05	-0.14	0.07	-4.50	-0.16	0.33	-0.03
-0.77	0.10	0.22	4.00	-0.40	0.04	0.11	-0.30
-2.56	-0.21	0.35	0.50	0	-0.06	0.21	0.10
0.06	-0.50	-0.51	-0.24	-0.07	0.32	-0.07	-0.21

1161.13	2.01	-0.16	0.94	2.13	0.24	-0.07	-0.29
-3.52	0.09	-3.01	-1.19	0.41	-0.35	0.33	0.51
-3.05	-4.18	1.24	-1.37	-2.85	-0.13	-0.34	-0.36
-2.28	0	-0.88	2.03	0.67	0.23	0.48	0.26
0.12	2.77	0.30	-0.22	4.62	0.32	-0.26	-0.25
-1.86	0.57	3.99	0.13	-0.37	0.03	-0.37	-0.18
-0.50	0.25	0.16	-0.21	-0.03	-0.37	0.01	0.33
0.10	-0.09	0.82	0.53	-0.25	-0.02	0.11	-0.16

1223.50	6.81	-2.69	2.00	0	0.14	0.03	0.33
-43.35	8.32	2.20	-1.96	2.18	3.57	0.05	0.36
9.51	-0.99	-1.88	1.02	-3.21	0.12	-0.28	0.27
8.13	0.92	-2.06	2.30	-0.26	0.32	0.08	-0.28
-4.50	3.51	0.46	-4.25	-1.05	0.06	-0.02	0.27
2.15	-1.83	0.07	0.07	-0.26	0.11	0.01	-0.55
0.11	0.33	-0.78	-0.21	-0.56	-0.37	-0.11	0.26
-0.03	0.61	-0.13	-0.09	-0.01	0.16	0.32	-0.24

1221.38	-1.22	-2.86	0.88	1.88	0.16	-0.03	-0.12
-35.10	-6.06	0.02	-0.98	-0.16	0.35	0.13	0.35
-3.78	2.35	2.09	-3.07	-2.59	4.10	-0.34	-0.20
3.76	1.32	-2.78	-0.23	2.97	0.41	0.13	0.14
-2.63	-0.63	0.03	0.47	-0.13	-0.11	0.39	0.21
0.09	0.17	0.09	0.08	0.12	-0.28	-0.14	0.28
0.35	-3.94	0.16	0.48	-0.08	0	0.15	-0.18
-0.13	0.85	-0.09	-0.32	-0.15	-0.01	0.15	0.48

Bloques DCT de la zona marcada en la imagen. Los coeficientes cuyo valor absoluto es menor que 8 se marcan en rojo.



Imágenes correspondientes a los bloques DCT anteriores

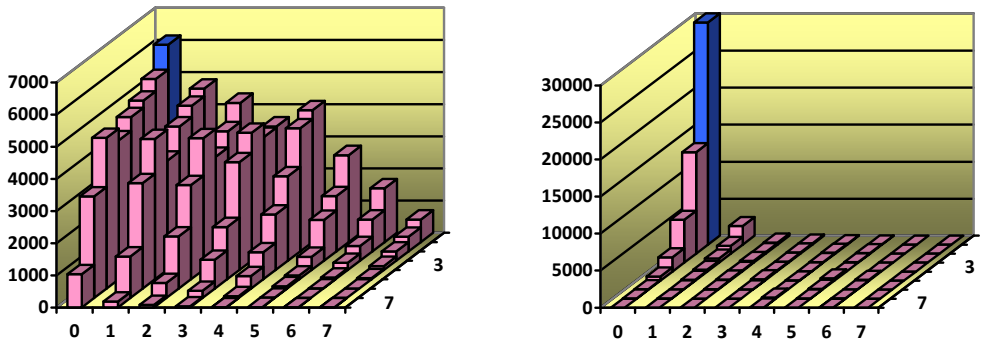
FIGURA 2-12: DCT de una imagen borrosa

Los autores de [5] sacan partido de este comportamiento de la DCT creando un sencillo algoritmo para obtener la calidad en cuanto a *blur* se refiere. Para empezar, se divide la imagen en escala de grises en bloques de 8x8 píxeles a los que se les aplica la DCT. Luego se construye un histograma con todos ellos que da una idea de cómo se distribuyen las frecuencias de la imagen contando cuántos coeficientes superan un umbral. Una vez construido el histograma, se estima el factor de *blur* a partir de éste y de una matriz de pesos, considerando que un coeficiente es borroso si su aparición en el histograma es menor que otro umbral.

La elección de celdas de 8x8 píxeles para la división de la imagen se debe a que el estándar JPEG (anexo D) utiliza la DCT con bloques de esta dimensión, siendo el tamaño más generalizado. Por supuesto, si la imagen no es divisible en bloques 8x8, la última fila y columna se replica hasta que las dimensiones de la figura sean múltiplo de 8.

El histograma tiene el mismo tamaño que un bloque (8x8 valores) y se construye examinando uno por uno todos los bloques de la imagen. Para que un valor del histograma aumente, el coeficiente en la posición correspondiente tiene que tener un valor absoluto superior a 8, que es el valor que tendría el coeficiente DC de un bloque uniforme de luminancia 1. De esta forma se eliminan los coeficientes pequeños que pueden deber su peso al ruido.

Los histogramas de las imágenes de las figuras 2-11 y 2-12 tienen el aspecto siguiente:



Histograma de la figura 2-11 **Histograma de la figura 2-12**

FIGURA 2-13: Histogramas

El número de coeficientes DC es proporcional al tamaño de la imagen, ya que es exactamente el número de píxeles de la imagen dividido entre 64 (tamaño del bloque DCT). Así, el número de coeficientes DC ascenderá en la figura 2-11 (que tiene un tamaño de 768 x 512 píxeles) hasta los 6144, mientras que la figura 2-12 (de tamaño 1600 x 1200 píxeles) alcanzará los 30000. Como es lógico, el resultado debe ser independiente de esta diferencia.

De hecho, el factor de *blur* se obtiene repasando uno a uno los valores del histograma, y analizando si éste es mayor que un 10% de las veces que aparece el coeficiente DC. Si el resultado es positivo, el factor de *blur* queda intacto. Si el valor del coeficiente es menor, por el contrario, se suma el valor de la matriz de pesos que corresponde a esa posición.

8	7	6	5	4	3	2	1
7	8	7	6	5	4	3	2
6	7	8	7	6	5	4	3
5	6	7	8	7	6	5	4
4	5	6	7	8	7	6	5
3	4	5	6	7	8	7	6
2	3	4	5	6	7	8	7
1	2	3	4	5	6	7	8

TABLA 2-2: Matriz de pesos

En el ejemplo anterior, el número de veces que aparecía el coeficiente DC en la figura 2-11 era 6144. El umbral por debajo del cual se suma será, por tanto, 614'4. En la figura 2-12, el umbral se establece a 3000. En el histograma de la figura 2-12, la posición (1,2) se ha contado sólo 204 veces, por lo que al factor de *blur* se le sumará 7, número que corresponde a la posición (1,2) en la matriz de pesos.

Para terminar, el valor obtenido en la variable *blur* se divide entre 344 (suma de todos los números de la matriz de pesos) a fin de obtener un valor entre 0 y 1 que represente el porcentaje de extensión del *blur* en la imagen. Un porcentaje del 0% indicaría que la imagen es completamente nítida y uno del 100% que la imagen es completamente borrosa.

2.3.1.2 Sobre/Baja Exposición y Homogeneidad

Aparte de estudiar si una imagen es borrosa o no, la calidad también se mide en términos de homogeneidad. Las imágenes sobre/sub expuestas se caracterizan por tener un color (o luminosidad) medio muy claro o muy oscuro, cualidad que determina su baja calidad. Y las imágenes homogéneas se distinguen por ser prácticamente constantes (no se diferencian formas y no existen variaciones considerables de color).

Estas definiciones permiten que estos dos conceptos se puedan llegar a mezclar, es decir, que la medida de ambos vaya relacionada con un mismo factor: la entropía.

2.3.1.2.1 Entropía

El papel de un diseñador de sistemas de comunicación es conseguir que la información que se transmite llegue correctamente al receptor. Como esta información varía impredeciblemente con el tiempo (si fuera predecible, no haría falta transmitirla), puede ser modelada como un proceso aleatorio.

En 1948, Claude E. Shannon consiguió medir la aleatoriedad de la señal transmitida, lo que se traduce en determinar la incertidumbre que lleva, su cantidad de información. Un ejemplo en [10] aclara que, de un conjunto de informaciones (como puede ser la

temperatura en el mes de Julio en una ciudad española), la menos probable (frío) es la que aporta más información al receptor.

La medida de la información que descubrió Shannon es la entropía. Medida en bits, define la cantidad de información contenida en la fuente de transmisión a partir de la información particular de cada salida. Si todas las salidas son equiprobables, la entropía será máxima ya que el desorden de la información de la fuente (su aleatoriedad) no puede ser mayor.

La fórmula que permite obtener la entropía de X es:

$$H(X) = -\sum_{i=0}^{N} p_i \log p_i = \sum_{i=0}^{N} p_i \log\left(\frac{1}{p_i}\right)$$

Donde log está en base 2 para obtener la entropía en bits, $0 \cdot \log 0 = 0$ y p_i representa la función densidad de probabilidad de la variable aleatoria discreta X.

Si en una fuente binaria sin memoria (la salida sólo puede ser 0 o 1, y la salida actual no depende de la anterior), las probabilidades de que salga uno u otro símbolo son *p* y (1-*p*) respectivamente, la entropía tendrá el aspecto siguiente:

$$H(X) = -p \log p - (1-p)\log(1-p)$$

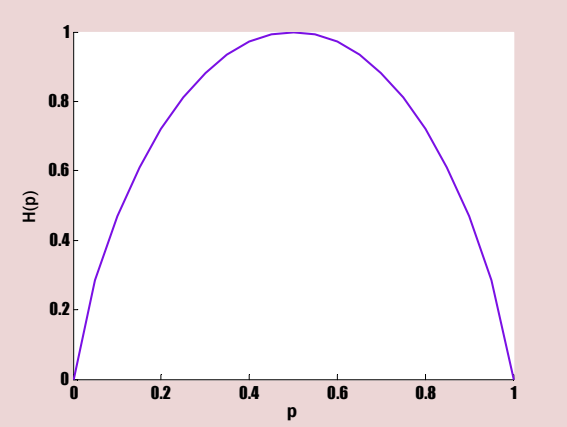


FIGURA 2-14: Entropía de una fuente binaria
La entropía es máxima cuando los símbolos son equiprobables.

En el caso de una imagen, la entropía de su distribución de colores será muy alta en el caso de que la imagen tenga mucha variación. Como las imágenes sobre/sub expuestas y homogéneas suelen imágenes de color casi constante, la medida de la entropía de una imagen puede dar una idea de si la figura es de buena calidad o no.

Para obtener la entropía de una imagen, primero se ha de obtener la función densidad de probabilidad de los colores que se encuentran en ella. Para ello, se construye un histograma HSV (para más información sobre espacios de color, anexo C) que represente,

para cada color estudiado, el número de veces que éste se repite en la figura. Para equipararlo a la función densidad de probabilidad de colores de la imagen, se normalizan los valores del histograma de manera que sumando todos ellos se obtenga 1. A continuación, basta con aplicar la fórmula haciendo uso de cada uno de los valores del histograma normalizado para conseguir la medida de la entropía.

Como ejemplo se exponen seguidamente tres imágenes con diferente exposición y su valor de entropía:

Imagen sobreexpuesta

Su entropía es de 5.27 bits

Imagen normal

Su entropía es de 8.43 bits

Imagen subexpuesta

Su entropía es de 6.76 bits

FIGURA 2-15: Entropía de 3 imágenes con exposiciones diferentes

En este caso la exposición de las imágenes está modificada artificialmente, pero se puede comprobar que el resultado con imágenes naturales también es satisfactorio:

Su entropía es de 11.68 bits

Su entropía es de 5.66 bits

FIGURA 2-16: Imágenes de ejemplo y su entropía correspondiente

2.3.1.3 Función de calidad

La función de calidad es la fórmula que mezcla los dos conceptos (*blur* y entropía) de forma que para cada imagen se obtenga un único valor que la califique.

En [1], la fórmula propuesta sugiere ponderar por igual las dos medidas dando la misma importancia a la sobre/sub exposición y homogeneidad como a la borrosidad.

Para el estudio de los algoritmos sugeridos en los artículos mencionados en 2.2, se mantienen los pesos de la fórmula indicados en [1].

2.3.2 Detección de duplicados

Imágenes duplicadas son aquellas que, por su similitud, conservar ambas no aporta prácticamente ninguna información nueva al usuario. La detección de duplicados, pues, es otro método que permite liberar de carga al sistema posterior de análisis, eliminando las imágenes que no contribuyan a nada nuevo.

Todos los algoritmos que realizan esta detección se ayudan construyendo una "representación" que caracterice las imágenes para luego compararlas entre sí. Esta representación, junto con los métodos que establecen una medida de la diferencia entre ellas, constituyen los temas de estudio de los distintos algoritmos.

Se consideran métodos interesantes los citados en [7] y en [12]: firma de color y descriptores de MPEG-7. La Organización Mundial para la Estandarización (ISO/IEC) lanza en 2001 el estándar MPEG-7, el cual define formalmente cómo ha de ser la descripción de los contenidos audiovisuales. Dentro de la Parte 3 [13] (la que se dedica a contenidos visuales) se incluyen cuatro descriptores de color, cada uno de los cuales representa diferentes aspectos de las características de los colores. Dos de ellos, el *Dominant Color* y el *Color Layout*, son los objetivos de estudio en los apartados 2.3.2.2 y 2.3.2.3.

2.3.2.1 Firma de Color

La representación que crea este algoritmo, propuesto en [7], identificará a cada imagen calculando un conjunto de firmas de color y firmas de imagen. Esta información es almacenada en una base de datos para luego compararla con la correspondiente a las imágenes nuevas que van llegando, y de esta forma equiparar a los pares de imágenes duplicadas.

Lo novedoso del método es que no hace uso de histogramas (que no caracterizan adecuadamente a una imagen, ya que dos imágenes completamente distintas pero con la misma cantidad de colores tendrían el mismo histograma), sino que construye un modelo que incluye tanto el color de la figura cómo su posición en la imagen.

Primero se identifican los "colores dominantes" presentes en la imagen, para posteriormente, con la imagen dividida en celdas, asignar a cada una de ellas el color o colores dominantes que se considere que se encuentren en ellas, dando lugar a las "firmas

de color". Con el conjunto de las firmas de color se construye la "firma de imagen". Para una persona dos imágenes son parecidas si tienen los mismos colores y la misma distribución de colores, lo que se traduce en el sistema de firmas en que las imágenes tengan los mismos colores dominantes y que sus firmas sean parecidas.

El primer paso consiste en identificar cuáles van a ser los colores dominantes que se van a tener en cuenta para las firmas. En general, el objeto de interés se suele encontrar en el centro de las fotografías, mientras que el borde suele representar al fondo. Realizando un histograma de cada una de estas dos regiones (H_c y H_f) se pueden determinar los colores que se encuentran mayoritariamente en cada una de ellas (C_c y C_f). El conjunto de colores seleccionados es lo que se denomina "colores dominantes" (C_T) de la imagen.

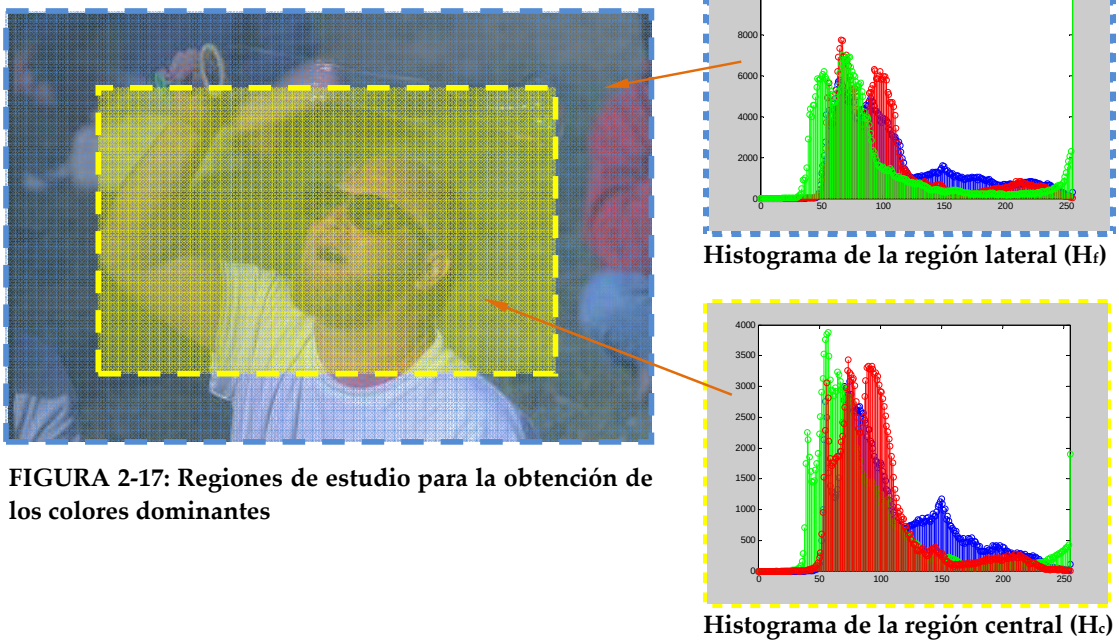


FIGURA 2-17: Regiones de estudio para la obtención de los colores dominantes

Histograma de la región lateral (H_f)

Histograma de la región central (H_c)

Para incorporar la información espacial del color, la imagen se divide en celdas (*m*x*n*). Cada una de estas celdas se examina en busca cada uno de los C_T colores dominantes, dando lugar a C_T firmas de color de tamaño *m*x*n*. Si da positiva una medida de similitud entre el color Ci y el color de la celda, el bit correspondiente a esa celda en la firma del color C_i se activa (1), y si no se deja a 0. La obtención de la firma de la imagen se hace sobreponiendo (con una operación OR) las C_T firmas de color (ver figura 2-18).

Aunque el algoritmo promete por la clase de información que se asocia a cada imagen, resulta demasiado lento para los objetivos del proyecto (en su parte de aplicación a secuencias de vídeo). Su implementación se abandona antes de desarrollar la medida de similitud.

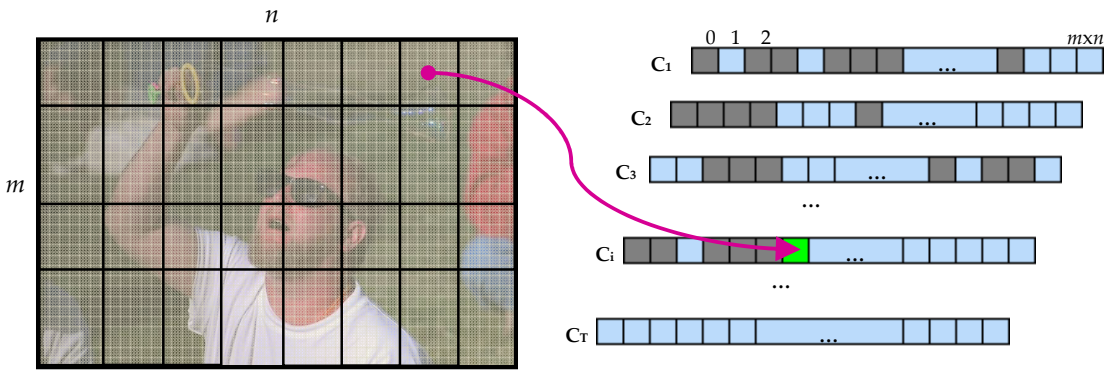


Imagen dividida en celdas

Firmas de color de la imagen. Se estudia si el color C_i se encuentra en la celda 6.

FIGURA 2-18: Firmas de color de la imagen

2.3.2.2 Dominant Color

Dominant Color es un descriptor de MPEG-7 [13] que determina cuáles son los valores de los colores característicos de una región, su porcentaje de aparición, cómo de diseminados se encuentran y la varianza de cada color.

El algoritmo implementado comienza dividiendo la imagen original en celdas de 8x8 píxeles, que son enviadas una a una al algoritmo que utiliza *Dominant Color* para extraer sus colores representativos. Estas representaciones son almacenadas para su posterior comparación.

FIGURA 2-19: Imagen original (768x512 píxeles) con una malla de celdas de 8x8 píxeles

La Parte 8 (sección 4.2.3) del estándar MPEG-7 [14] y [15] describen detalladamente los pasos que se han de seguir para extraer los colores dominantes de una región. *Dominant Color* recurre al algoritmo GLA (Generalized Lloyd Algorithm) para encontrar los colores representativos de una celda y consecutivamente fusionarlos hasta que se localicen uno o varios que cumplan una distancia mínima.

El algoritmo se divide en dos secciones principales: la obtención de los centroides y la fusión de éstos para encontrar los colores dominantes. El criterio que decide cuando se pasa de una parte a otra es que, una vez alcanzado el número máximo de centroides calculados (8), la diferencia entre la distancia anterior (entre ellos y los colores de la imagen) y la distancia actual, dividida entre la distancia anterior sea menor que 0.01, es decir, que haya variado menos del 1%.

$$\frac{Dist_{i-1} - Dist_i}{Dist_{i-1}} \leq 0.01$$

El apartado de obtención de centroides se resume a la repetición de cuatro pasos: el cálculo de los nuevos centroides a partir de una matriz que indica a qué píxel le corresponde qué color; la adquisición de la distancia actual total a partir de las distancias entre cada píxel de la celda y el nuevo centroide que le corresponde por cercanía; la creación de un nuevo centroide dividiendo el menos parecido a los píxeles en dos; y la elaboración de una nueva matriz de parecidos incluyendo los dos centroides recién establecidos.

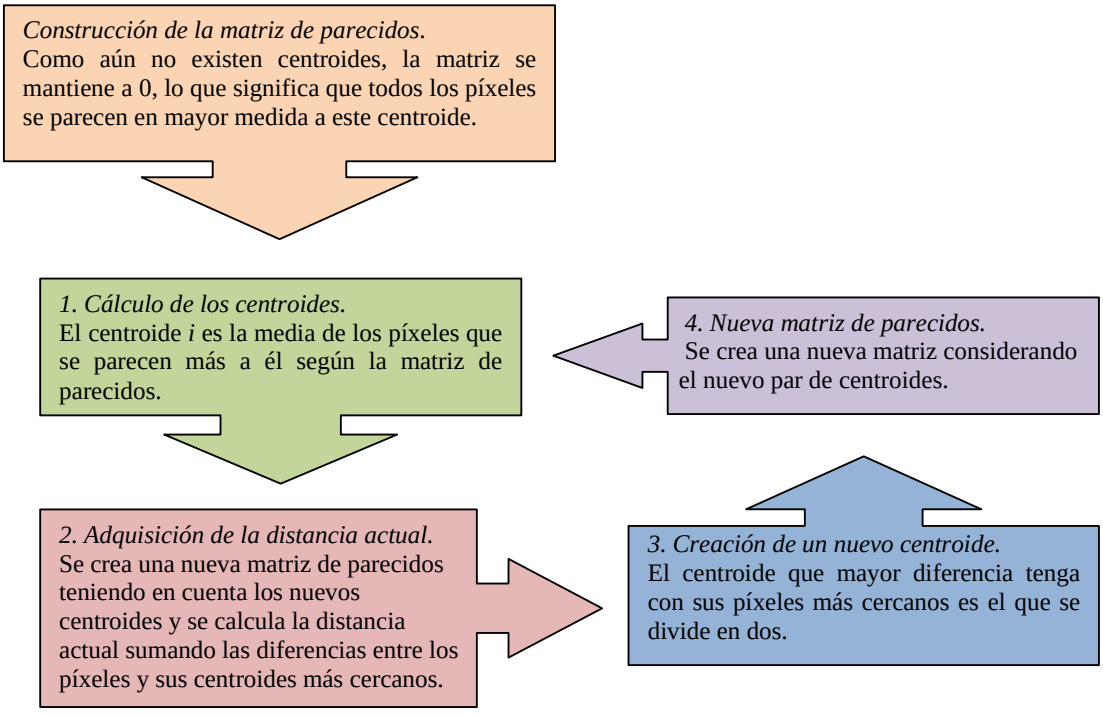


FIGURA 2-20: Diagrama que ilustra los pasos que sigue el proceso de obtención de centroides

Una vez encontrados los 8 centroides (o colores representativos) que corresponden a la celda, el algoritmo continúa con el apartado de fusión que, basándose en las distancias entre centroides, reduce el número de estos fundiendo las parejas que no sean lo suficientemente diferentes.

Para ilustrar el funcionamiento de este algoritmo, se realiza a continuación un ejemplo con una celda de 8x8 píxeles, donde 32 son rojos, 24 son verdes y 8 son azules. En este caso los colores dominantes serán el rojo, el verde y el azul, ya que las distancias entre ellos son demasiado grandes como para unirse en un mismo color.

El primer paso del algoritmo es trasponer la matriz en la que se encuentran los píxeles, para posteriormente convertir los valores RGB al espacio CIE Luv (anexo C: espacios de color).

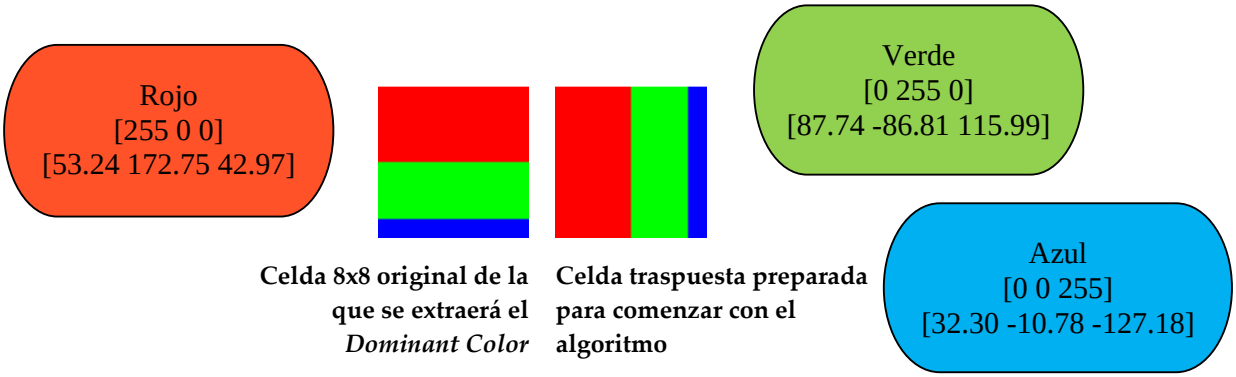


FIGURA 2-21: Celdas original (izda.) y de trabajo (dcha.) acompañadas de los valores de los colores (RGB y Luv) que contienen

Con la matriz de parecidos a 0, el primer centroide no es más que la media de todos los píxeles de la celda. Asociada a la tabla de centroides existe una variable (Pesos) que guarda el número de píxeles que se emparejan con un centroide, lo que permite calcular esta media.

	Centroides (Luv)		
0	63.56	52.47	49.08
1	0	0	0
2	0	0	0
	…		
7	0	0	0

	Pesos
0	64
1	0
2	0
	…
7	0

En el siguiente paso se calculan todas las distancias entre los píxeles y el centroide. Como sólo existe uno, la distancia menor se dará entre cada píxel y el centroide 0, con lo que la matriz de parecidos continúa a 0.

0	1	2	3	4	5	6	7	…	63
0	0	0	0	0	0	0	0	…	0

Matriz de parecidos

Para dividir el centroide en dos, se vuelven a calcular las distancias entre éste y los píxeles con los que se le relaciona y, multiplicándolas por un factor, se establecen los "valores diferencia". Estos números son los que se suman y restan al centroide para obtener un par nuevo.

$Diff_1 = 1.99$
$Diff_2 = 12.25$
$Diff_3 = 7.47$

Centroides (Luv)

0	61.57	40.22	41.61
1	65.54	64.72	56.55
2	0	0	0
3	0	0	0
	…		
7	0	0	0

Con los nuevos centroides calculados se llega al cuarto paso, en el que se calcula una nueva matriz de parecidos. El resultado es que los píxeles rojos se parecen más al centroide 1, mientras que los verdes y azules se acercan al 0.

0	1	2	3	4	5	6	7	8	9		63
1	1	1	1	0	0	0	0	1	1	…	0

Matriz de parecidos

Comenzando de nuevo por el paso 1, la matriz de centroides se calcula de nuevo a partir de la matriz de parecidos. El centroide 1, calculado como la media de los píxeles rojos, pasa directamente a convertirse en este color. El centroide 0 será la media de los píxeles verdes y azules. Tras la aplicación del paso 2 se obtiene la nueva matriz de parecidos, que coincide con la anterior.

Centroides (Luv)

0	73.88	-67.81	55.20
1	53.24	172.75	42.97
2	0	0	0
3	0	0	0
	…		
7	0	0	0

Pesos

0	32
1	32
2	0
3	0
	…
7	0

Llegado el momento de la separación de un centroide en dos, el elegido es, lógicamente, el centroide 0, pues es el que menos se parece a los píxeles que se le asocian.

$Diff_1 = 2.40$
$Diff_2 = 3.29$
$Diff_3 = 10.53$

Centroides (Luv)

0	71.47	-71.10	44.66
1	53.24	172.75	42.97
2	76.28	-65.51	65.73
3	0	0	0
	…		
7	0	0	0

Con estos centroides, la nueva matriz de parecidos se convierte en:

0	1	2	3	4	5	6	7	8	9		63
1	1	1	1	2	2	2	0	1	1	…	0

Matriz de parecidos

Volviendo de nuevo al primer paso, los nuevos centroides son, cada uno, la media de cada uno de los 3 colores.

	Centroides (Luv)		
0	32.30	-10.78	-127.18
1	53.24	172.75	42.97
2	87.74	-86.81	115.99
3	0	0	0
	…		
7	0	0	0

	Pesos
0	8
1	32
2	24
3	0
	…
7	0

El algoritmo continúa dividiendo el centroide 1 en dos pero, como la distancia entre los píxeles rojos y el centroide 1 es tan pequeña, los centroides que se crean a continuación nunca son elegidos por la matriz de parecidos. Los valores de los nuevos centroides se mantienen a 0.

Cuando termina la parte de obtención de centroides, comienza la fusión. El primer paso que se realiza es la construcción de una matriz 8x8 en la que se recopilan las distancias entre unos centroides y otros, pero con la condición de que sus pesos sean superiores a 0. En el ejemplo, sólo se calculan las distancias entre los centroides 1 y 0, 2 y 0 y 0 y 1.

	0	1	2	3	4	5	6	7
0								
1	63069.84							
2	67983.16	73894.01						
3	0	0	0					
4	0	0	0	0				
5	0	0	0	0	0			
6	0	0	0	0	0	0		
7	0	0	0	0	0	0	0	

TABLA 2-3: Matriz de distancias

La fusión de dos centroides se realiza si entre ellos se da la distancia mínima de las que se encuentran en la tabla y si esta distancia es menor que cierto umbral. En el ejemplo la primera distancia mínima se da entre los centroides 3 y 0. El nuevo centroide se calcula aplicando la fórmula siguiente a cada una de las tres componentes L, u y v:

$$Centroides_L[0] = \frac{peso_3 \cdot Centroides_L[3] + peso_0 \cdot Centroides_L[0]}{peso_3 + peso_0}$$

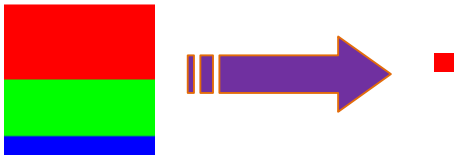
Al ser 0 el peso del centroide 3, éste desaparece y el centroide 0 se queda como estaba.

Avanzando con el algoritmo, las nuevas distancias mínimas se irán dando entre 4 y 0, 5 y 0, etc. de forma que los centroides que no tenían peso van desapareciendo para finalmente dejar como colores dominantes a los centroides 0, 1 y 2. Llegado este punto, la distancia mínima se da (como se puede apreciar en la tabla 2-3) entre los centroides 1 y 0. Casualmente, el valor de 63069.84 es muy superior al umbral, lo que causa que se finalice el proceso y que los colores dominantes de la imagen sean:

	Centroides (Luv)		
0	32.30	-10.78	-127.18
1	53.24	172.75	42.97
2	87.74	-86.81	115.99

	Pesos
0	8
1	32
2	24

Con esos pesos, el color seleccionado para representar la celda será el rojo, puesto que se encuentra en mayor porcentaje. De esta forma la celda tricolor de 8x8 píxeles de tamaño pasa a ser representada por un píxel de color rojo.



Celda de ejemplo **Color representativo de la celda**

FIGURA 2-22: Color representativo de la celda

Una vez llega el momento del balance, las representaciones, de dimensiones 1/64 menores que la imagen original, se convierten del espacio RGB al espacio CIE Luv (véase anexo C), ya que en este último se puede medir la similitud entre dos colores mediante la distancia euclídea [7].

$$d(i,j) = \sqrt{(L_{1_i} - L_{2_j})^2 + (u_{1_i} - u_{2_j})^2 + (v_{1_i} - v_{2_j})^2} \qquad \textbf{Fórmula 2-1}$$

La fórmula que mide la desigualdad entre dos representaciones, y por tanto entre dos imágenes es la siguiente:

$$D_{im1-im2} = \frac{1}{m \cdot n} \sum_{i=0}^{m} \sum_{j=0}^{n} d(i,j) \qquad \textbf{Fórmula 2-2}$$

Donde m y n son las longitudes de los lados de las representaciones, $d(i,j)$ la distancia euclídea entre los colores en la posición (i,j) de las representaciones 1 y 2, y $D_{im1\text{-}im2}$ la distancia total entre las dos imágenes.

La imagen expuesta a continuación es la representación de la figura 2-19, obtenida mediante el algoritmo anteriormente descrito.

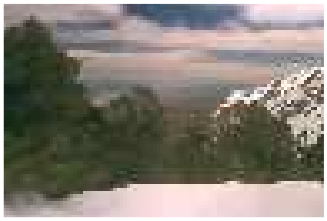


FIGURA 2-23: Imagen representación calculada con *Dominant Color* (96x64 píxeles)

2.3.2.3 Color Layout

Según el estándar MPEG-7 (Parte 3) [13], *Color Layout* es el descriptor visual que permite obtener la distribución espacial de colores de una imagen.

El algoritmo parte de una representación de 8x8 píxeles creada a partir de la imagen original (cuya obtención no especifica el estándar), que se convierte al espacio YCrCb (véase anexo C) para después aplicarle la DCT. El resultado es una matriz 8x8 que representa el *Color Layout* de la imagen original, que es almacenada para su posterior comparación.

La construcción de la imagen 8x8 se realiza mediante submuestreo, promediando de 4 en 4 píxeles adyacentes hasta obtener el tamaño deseado.

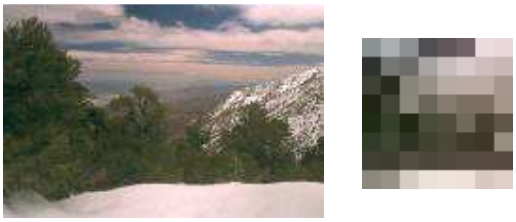


Imagen original **Imagen 8x8 tras el submuestreo**

FIGURA 2-24: Submuestreo

Una vez obtenida esta imagen, su transformación al espacio YCrCb permite obtener las tres capas expuestas a continuación:

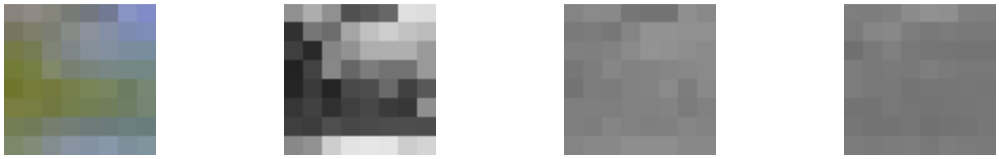


Imagen 8x8 YCrCb **Imagen de la componente Y** **Imagen de la componente Cr** **Imagen de la componente Cb**

FIGURA 2-25: Componentes Y, Cr, Cb

El algoritmo sigue con el empleo de la DCT (implementada según el estándar (Partes 3 (6.6) [13] y 8 (4.2.5) [14])) sobre cada una de las capas, de manera que la representación que finalmente se logra da una idea de cómo se distribuyen las frecuencias en la imagen de tamaño 8x8 YCrCb.

Imagen DCT de la imagen de la componente Y **Imagen DCT de la imagen de la componente Cr** **Imagen DCT de la imagen de la componente Cb**

FIGURA 2-26: DCT de las componentes Y, Cr, Cb

La diferencia entre imágenes se calcula como la resta entre dos coeficientes del mismo tipo al cuadrado, multiplicada por un peso dado (matriz de pesos especificada en [14] que cambia su valor según la posición de los coeficientes). Estos valores se elevan a ½ dando más peso a los coeficientes Y:

$$Diferencia = 2\sqrt{\sum zizag\_weight \cdot (CoefY_A - CoefY_B)^2} + 0.5\sqrt{\sum zizag\_weight \cdot (CoefCr_A - CoefCr_B)^2} + 0.5\sqrt{\sum zizag\_weight \cdot (coefCb_A - CoefCb_B)^2}$$

3 Diseño

3.1 Introducción

El objetivo de los algoritmos desarrollados es formar parte de un programa que implemente un repositorio de imágenes o un sumarizador de secuencias de vídeo en tiempo real. Este objetivo es importante en el diseño ya que cada algoritmo será planteado según las cualidades que se precisen para el programa en concreto.

Este capítulo se divide en tres secciones: algoritmos a nivel de píxel, algoritmos a nivel de bloque y aplicación a secuencias de vídeo. En la primera parte se describen los cambios realizados a algunos de los algoritmos mencionados en el Estado del Arte (apartado 2.2) además de nuevos métodos. En la segunda se intentan adaptar los algoritmos más útiles que operan a nivel de píxel (apartados 2.3 y 3.2) de forma que se consigan reducir los tiempos de ejecución sin pérdida de resultados. Para ello se trabajará con la imagen a nivel de bloque DCT, o lo que es lo mismo, sin estar completamente decodificada. Por último, en la tercera sección, se especifican las adaptaciones de los algoritmos anteriores a las necesidades de una aplicación de vídeo.

3.2 Algoritmos a nivel de píxel

En este apartado de algoritmos de nivel de píxel del capítulo de diseño se intenta innovar partiendo de algunas de las técnicas citadas en el apartado 2.2 (Sobel [11], DCT [5], Color Medio [1][2][3]) y, teniendo en cuenta estos avances, mejorar los resultados obtenidos procediendo a realizar cambios en la función de calidad [1].

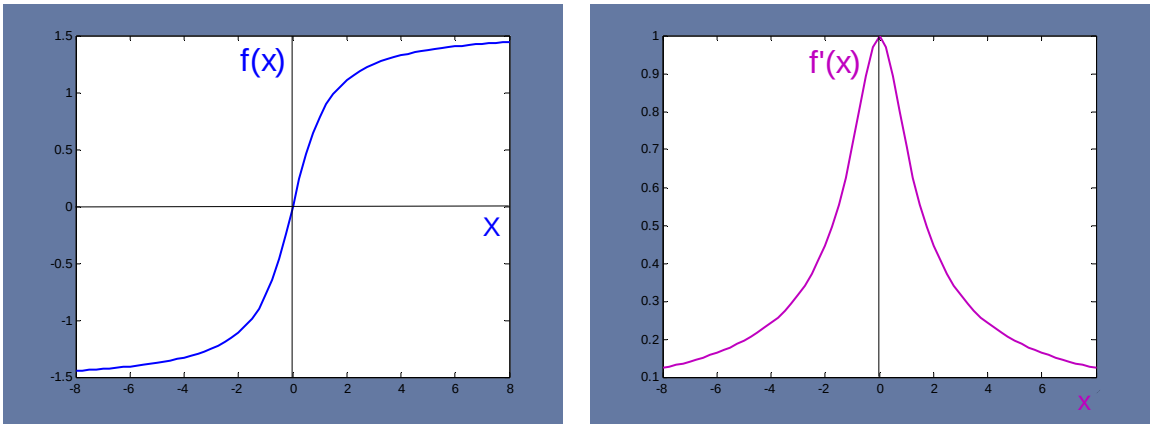
3.2.1 Filtrado de calidad

Las modificaciones aplicadas a nivel de filtrado de calidad se han realizado sobre el algoritmo de Sobel [11], el algoritmo que utiliza la DCT para establecer una medida de *blur* y sobre la función de calidad [1].

3.2.1.1 Operador de Sobel

El operador de Sobel [11] se trata de una técnica basada en el gradiente para realizar la detección de bordes, de forma que las variaciones de intensidad prevalecen frente a las zonas de intensidad constante.

El trabajo que realiza Sobel puede ilustrarse sencillamente mediante funciones de una dimensión, como la que se muestra en la figura 3-1. La función f(x) podría representar la variación de intensidad que se produce en un borde no muy abrupto. Si se aplica la primera derivada a esta función (lo cual es lo mismo que aplicar el gradiente, puesto que sólo existe la dirección x) el resultado es otra función f'(x) cuyos valores máximos se dan en la región en la que se produce el incremento de intensidad. La primera derivada y el operador de Sobel por extensión tienen como propiedad destacar las zonas que no son constantes, y esto se traduce en la capacidad de detectar bordes en funciones de dos dimensiones como son las imágenes digitales.



Función f(t) **Derivada de la función f(t)**

FIGURA 3-1: Función y su derivada

Aunque la derivada está asociada a la continuidad y las imágenes digitales son discretas, se puede llegar a una aproximación suponiendo que la imagen no es más que una función continua muestreada en los píxeles que la definen.

Partiendo de esta aproximación, el operador de Sobel permite obtener para cada punto (píxel) un vector que define la dirección en la que se produce el incremento máximo posible de intensidad (de negro a blanco) entre este punto y los de su alrededor, así como la magnitud de este incremento. Esto implica que el resultado de aplicar el operador de Sobel sobre una región de intensidad constante es un vector 0, mientras que aplicarlo sobre un borde produce un vector resultado perpendicular a la dirección de éste cuyo sentido va desde los puntos más oscuros hacia los más claros.

Las máscaras que realizan esta operación de detección de bordes, en función de cuál sea la dirección en la que se aplica el gradiente, son las siguientes: G_X se aplica en la dirección x, que corresponde a las columnas de la imagen, mientras que G_Y se aplica en y, dirección en la que avanzan las filas.

$$G_X = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad G_Y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

El resultado de aplicar estas máscaras a la imagen nítida del capítulo anterior (figura 2-5) depende de si se emplea G_X, G_Y o una combinación de las dos. En todos los casos la imagen resultante destaca en blanco los bordes y deja en negro las zonas de luminancia uniforme. En el caso de G_X, los bordes resaltados son los que marcan discontinuidades verticales, mientras que G_Y acentúa los que indican cortes horizontales. La combinación de ambas máscaras permite la obtención de los bordes diagonales.

Resultado de aplicar G_X a una imagen nítida

Resultado de aplicar G_Y a una imagen nítida

Resultado de aplicar una combinación de G_X y G_Y a una imagen nítida

FIGURA 3-2: Resultados de aplicar Sobel a una imagen nítida

Resultado de aplicar G_X a una imagen borrosa

Resultado de aplicar G_Y a una imagen borrosa

Resultado de aplicar una combinación de G_X y G_Y a una imagen borrosa

FIGURA 3-3: Resultados de aplicar Sobel a una imagen borrosa

Sin embargo, si se aplican las mismas máscaras sobre la imagen borrosa (figura 2-6), los bordes recalcados son menos y se marcan con líneas más difusas, todo lo cual representa la falta de detalle de este tipo de imágenes (figura 3-3).

El operador de Sobel (como lo hacía la transformada de Haar) diferencia claramente entre imágenes nítidas y borrosas, pero, además, cumple con la propiedad que también tenía la transformada de Haar de recuperar los bordes Tejado y Escalón G (los que cambian gradualmente) a medida que aumentan los "niveles", considerando "nivel 1" a los resultados obtenidos a partir de la imagen original en escala de grises, "nivel 2" a los que surgen de la imagen original muestreada a la mitad y "nivel 3" a los que se consiguen tras muestrear de nuevo la imagen de la que se partía en el nivel 2.

Las figuras mostradas a continuación son el resultado de aplicar Sobel sobre la imagen borrosa en los niveles 2 y 3, donde se aprecia que se cumple la propiedad señalada.

Imagen de nivel 2 de una imagen borrosa

Resultado de aplicar G_X

Resultado de aplicar G_Y

Resultado de aplicar una combinación de G_X y G_Y

FIGURA 3-4: Resultado de aplicar Sobel sobre la imagen de nivel 2 de una imagen borrosa

Imagen de nivel 3 de una imagen borrosa

Resultado de aplicar G_X

Resultado de aplicar G_Y

Resultado de aplicar una combinación de G_X y G_Y

FIGURA 3-5: Resultado de aplicar Sobel sobre la imagen de nivel 3 de una imagen borrosa

En cualquier caso, para poder utilizar el mismo estudio de bordes del que se hacía uso en el apartado 2.3.1.1.1, hace falta demostrar que cuando se aplica el mismo algoritmo que se utilizaba con la transformada de Haar el comportamiento según el tipo de borde es el mismo.

Este algoritmo consistía en combinar las imágenes resultado de un mismo nivel en una matriz de valores (Emap) (ver figura 2-10) y en procesar estas matrices con un enventanado para obtener sus valores máximos, de manera que finalmente tuvieran el mismo tamaño, dando lugar a las matrices $Emax_i$. Posteriormente, la tabla 2-1 reflejaba el valor de $Emax_i$ dependiendo del tipo de borde, de forma que mediante las matrices Emax de los tres niveles se podía caracterizar el tipo de borde concreto, y si éste estaba borroso.

Aspecto de $Emax_1$ de una imagen nítida

Aspecto de $Emax_1$ de una imagen borrosa

Aspecto de $Emax_2$ de una imagen nítida

Aspecto de $Emax_2$ de una imagen borrosa

Aspecto de $Emax_3$ de una imagen nítida

Aspecto de $Emax_3$ de una imagen borrosa

FIGURA 3-6: Las imágenes $Emax_i$ obtenidas tras la aplicación del algoritmo, esta vez tras el uso del operador de Sobel, sobre una imagen nítida y una borrosa

La figura 3-6 demuestra que la utilización del algoritmo es posible: los bordes tipo Delta y Escalón A (como las briznas de hierba o los troncos de los árboles) aparecen cada vez más débiles a medida que se aumentan los niveles de estudio; los bordes tipo Tejado y Escalón G (columnas del edificio), sin embargo, son más gruesos y más luminosos en el nivel 3 que en el 1.

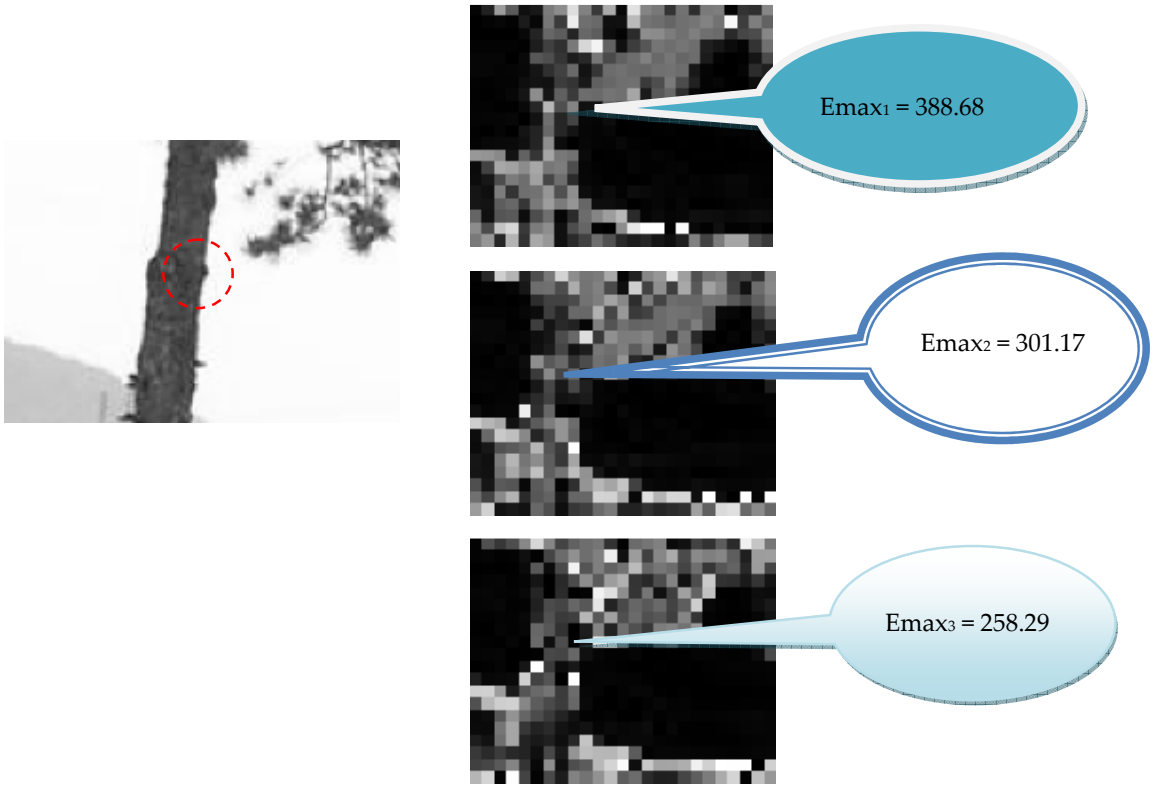


FIGURA 3-7: Detalle de un borde Escalón A de la imagen nítida en escala de grises y detalle de los Emax$_i$ correspondientes

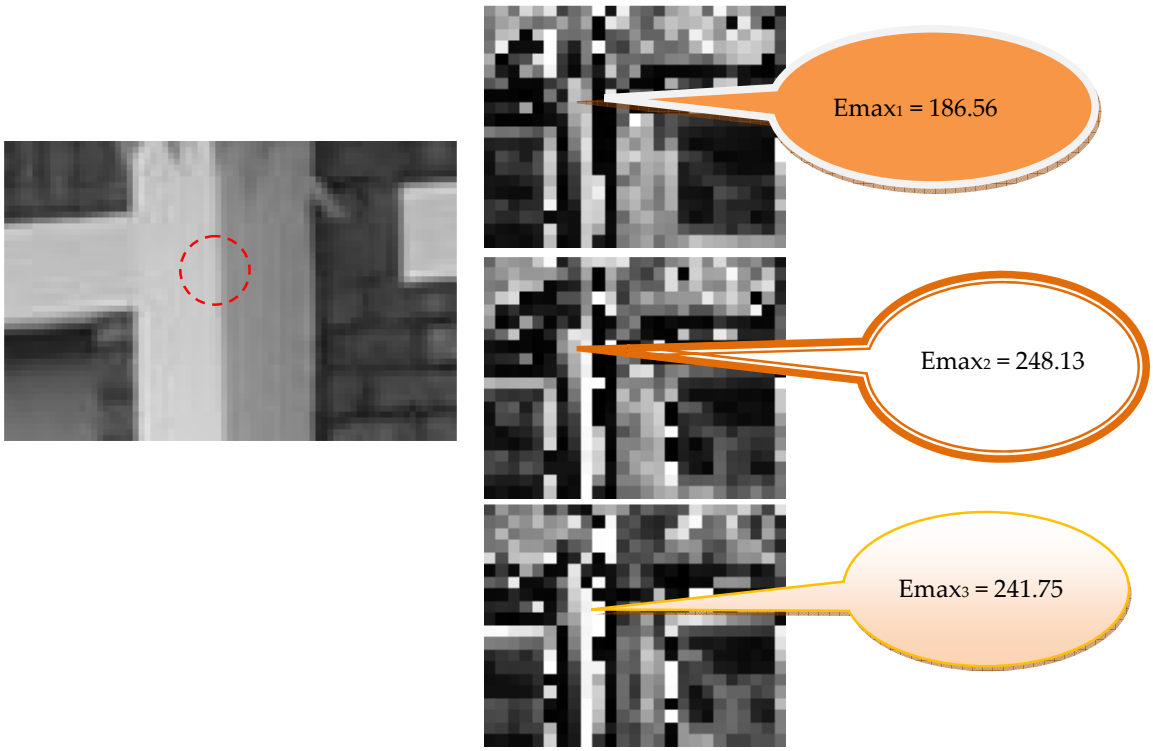


FIGURA 3-8: Detalle de un borde Tejado de la imagen nítida en escala de grises y detalle de los Emax$_i$ correspondientes

En la figura 3-7 se subraya el comportamiento de un borde Escalón A frente al algoritmo, y se comprueba que efectivamente cumple con las variaciones que se especifican en la tabla 2-1. Mediante un ejemplo similar se demuestra que el borde tipo Tejado también tiene el mismo comportamiento que tenía cuando se aplicaba la transformada de Haar (figura 3-8).

El algoritmo propuesto es, por tanto, el mismo utilizado para trabajar con la transformada de Haar, sustituyendo ésta por el operador de Sobel, pero añadiendo también un filtro Gaussiano (máscara 3x3 con desviación igual a 0.95) que elimine el ruido que provoca el que a fotos que no son de buena calidad se les adjudique un valor de calidad de *blur* altísimo, puesto que el operador de Sobel es muy sensible al ruido.

3.2.1.1 DCT

La técnica planteada en [5] (estudiada en profundidad en 2.3.1.1.2) cuenta con una flaqueza en cuanto a ruido se refiere. El algoritmo propuesto en es muy susceptible al ruido porque éste genera altas frecuencias. Siendo esto lo que precisamente caracteriza a un borde, el ruido puede causar que se falsee el resultado.

Siguiendo con la misma actuación sugerida en el algoritmo del operador de Sobel, las imágenes que tienen un porcentaje de calidad por encima de cierto límite se consideran ruidosas y se filtran, a fin de que reduzcan su medida de *blur*.

3.2.1.2 Función de calidad

En [1], la función de calidad es una forma de pesar el resultado de la medida de *blur* (calculada con el algoritmo de la transformada de Haar [4]) y el de la entropía, estableciendo los pesos a 0.5 respectivamente. Esta fórmula, aunque no desencaminada, sí está desactualizada con respecto al resto de los algoritmos que se estudian.

El objetivo es, por tanto, encontrar los valores adecuados para pesar las medidas calidad de *blur* y sobre/sub exposición y homogeneidad que mejores resultados ofrezcan para cada tipo de algoritmos. El establecimiento de estos valores es algo subjetivo, de la misma forma que lo son los resultados. El procedimiento más adecuado para encontrar estos pesos será la heurística, haciendo uso de una base de datos de imágenes de entrenamiento (anexo G).

Es importante tener en cuenta que, así como la variación del umbral determina una clara diferencia entre imágenes de alta y baja calidad, la oscilación de los pesos afecta a la ordenación que sufren entre ellas las imágenes de una calidad parecida. Para la institución de los pesos se utilizan 3 conjuntos de imágenes de entrenamiento (presentes en el anexo G): del grupo de imágenes de la base de datos de la Universidad de Corea, las 20 que corresponden a "*Globally Sharp*" y las 20 de "*Globally Blurry*" y, de "Internet", las once imágenes numeradas del 10 a 20.

Partiendo de las clasificaciones que surgían en estos grupos de imágenes con el algoritmo de la transformada de Haar (ponderando la función de calidad tal y como se sugería en el

artículo [4]) y de un criterio subjetivo, los pesos más adecuados para la función de calidad del algoritmo de Sobel son 0.5, al igual que para el algoritmo de la DCT.

Finalmente, los umbrales más apropiados para determinar si la calidad de una imagen es suficiente o insuficiente son 6.88 para el algoritmo de la transformada de Haar, 8.40 para el algoritmo del operador de Sobel y 6.90 para el que hace uso de la DCT, resultados a los que se llega examinando la totalidad de las imágenes de entrenamiento (anexo G). Los resultados que se obtienen con estos pesos y umbrales son los siguientes:

Algoritmo	Aciertos (%)	Falsos positivos (%)	Falsos negativos (%)
Haar	79.33	3.33	17.33
Sobel	90	2	8
DCT	89.33	0.67	10

TABLA 3-1: Resultados de entrenamiento de los algoritmos de calidad (píxel)

3.2.2 Detección de duplicados

El objetivo de los algoritmos de detección de duplicados es encontrar una imagen representación que sea lo más parecida posible a la original, de forma que se ahorre tiempo de ejecución sin perder en calidad de resultados.

3.2.2.1 Color Medio

Al igual que los demás algoritmos de detección de duplicados, éste crea una representación que simbolice la imagen original y establece una operación de medida de la diferencia entre representaciones.

En este caso, la representación se construye a partir de la imagen dividida en celdas de 8x8 píxeles de tamaño. Cada una de estas celdas se reduce a un píxel obteniendo su color medio. Este método es similar al propuesto en [1], [2] y [3], con la diferencia de que en aquéllos cada celda representaba la luminosidad media de cada región, y en este caso se tienen en cuenta las tres componentes (R G B) de color.

$$ColorMedio_R = \frac{1}{64}\sum_{i=0}^{7}\sum_{j=0}^{7} celda_R(i,j) \qquad ColorMedio_G = \frac{1}{64}\sum_{i=0}^{7}\sum_{j=0}^{7} celda_G(i,j)$$

$$ColorMedio_B = \frac{1}{64}\sum_{i=0}^{7}\sum_{j=0}^{7} celda_B(i,j)$$

Esta representación, que tiene 1/64 de las dimensiones de la imagen, es almacenada para su posterior comparación mediante la distancia euclídea, como se realizaba con *Dominant Color* (apartado 2.3.2.2). Las fórmulas 2-1 y 2-2 son las que se aplican tras la transformación de la imagen representación al espacio CIE Luv (véase anexo C para más información sobre espacios de color).

La figura a continuación refleja el resultado de utilizar el algoritmo del Color Medio sobre la imagen de la figura 2-19:

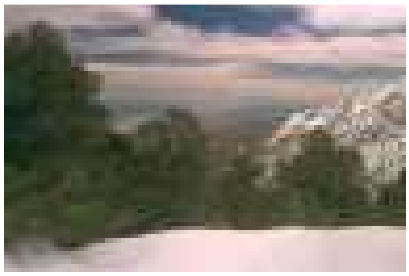


FIGURA 3-9: Imagen representación calculada con el color medio (96x64 píxeles)

La implementación de este algoritmo con la modificación de trabajar en color en vez de en escala de grises supone una mejora, puesto que la imagen representación se acerca más a la imagen original sin penalizar en demasía los tiempos de ejecución.

Ensayando con las imágenes de entrenamiento (anexo G), se decide que el umbral bajo el cual han de considerarse dos imágenes duplicadas con este método es de 1.5, mientras que para el algoritmo de *Dominant Color* el umbral más adecuado sería 1.63 y para el de *Color Layout* uno de 3.9, obteniéndose los siguientes resultados:

Algoritmo	Aciertos (%)	Falsos positivos (%)	Falsos negativos (%)
DC	60	5.56	40
CL	75	5.56	25
CM	55	5.56	45

TABLA 3-2: Resultados de entrenamiento de los algoritmos de detección de duplicados (píxel)

3.3 Algoritmos a nivel de bloque

Un bloque es el resultado de decodificar cualquier imagen comprimida en JPEG hasta el momento anterior a la aplicación de la IDCT (para más información sobre el estándar JPEG, véase el anexo D). Gestionar los algoritmos a nivel de bloque permite que el programa final ahorre mucho tiempo en decodificación (debido a que no es necesario esperar a obtener los píxeles que conforman la imagen), lo cual es idóneo para el trabajo en tiempo real.

El objetivo de esta sección es adaptar, en la medida de lo posible, algunos de los algoritmos seleccionados tanto en el apartado anterior como en el capítulo anterior al hecho de que en este caso ya no se cuenta con píxeles, sino con bloques DCT de tamaño 8x8.

3.3.1 Filtrado de calidad

Al igual que en otros apartados similares, los algoritmos de filtrado de calidad se encargan de deshacerse de las imágenes que, por su cantidad de imperfecciones, no aportan información útil al sistema.

3.3.1.1 Imágenes borrosas

Debido a que ya se había implementado un método que trabajaba con bloques DCT para detectar imágenes borrosas y calificarlas, se decide adoptar éste para este apartado a nivel de bloque.

3.3.1.1.1 DCT

En el apartado 2.3.1.1.2 se describe la utilidad de la DCT cómo método de análisis de altas frecuencias, y se relaciona a éstas con la presencia de bordes en una imagen (lo que la clasificaría como imagen no borrosa).

En el algoritmo correspondiente a nivel de píxel, era necesario dividir la imagen en bloques de 8x8 píxeles para empezar a trabajar, y aplicarles a éstos la DCT. La ventaja de operar a nivel de bloque es que ya no es preciso realizar estos pasos, puesto que se pueden obtener los bloques DCT directamente de la imagen JPEG. Aplicados estos cambios, se puede continuar con el resto del algoritmo tal y como se explicó en el capítulo de Estado del Arte.

3.3.1.2 Sobre/Baja Exposición y homogeneidad

En 2.3.1.2 se explica que estas dos ideas se pueden llegar a fundir en un mismo concepto si consideramos que la sobre/sub exposición hace que las imágenes se vuelvan planas, eliminando los detalles de color y forma, y que la homogeneidad se define precisamente como eso.

En esta sección, sin embargo, la medida de la entropía ya no es suficiente para determinar la calidad en cuanto a exposición y homogeneidad. Al recibir tan solo los bloques DCT, se pierde información de la imagen que era primordial para el cálculo de la variación de colores. Ahora se necesita otra medida adicional que aclare cómo varían las regiones.

A continuación se explican qué dos medidas se han tenido en cuenta para el cálculo de la calidad partiendo de la información disponible: coeficientes DC y coeficientes AC.

3.3.1.2.1 Coeficientes DC

El coeficiente DC representa el color medio de la celda de 8x8 píxeles donde se aplicó la DCT. A partir de los coeficientes DC, por tanto, es posible obtener una imagen reducida 1/64 de la imagen original en píxeles (1/8 de cada lado) con sus colores característicos.

A pesar de perder resolución, la paleta de colores de la imagen original se mantiene en su mayoría, lo que permite repetir la medida de la entropía utilizada en algoritmos a nivel de píxel.

Imagen creada a partir de los coeficientes DC

Imagen original

FIGURA 3-10: Obtención de la imagen de coeficientes DC

Para ello se calcula el histograma HSV de la imagen DC, que representa su función densidad de probabilidad de colores, y con ella la entropía que mide el desorden de colores en la imagen. A menor desorden, más posibilidades de que la imagen estuviera sobre/sub expuesta, o que fuera homogénea.

De esta forma, se muestran los resultados de la medida de la entropía en los tres casos (imagen sobreexpuesta, imagen normal e imagen subexpuesta) y en las imágenes que se daban de ejemplo en la figura 2-16. Las figuras son las que se obtienen mediante los coeficientes DC y a partir de las cuales se calcula la entropía.

Imagen sobreexpuesta

Su entropía es de 5.05 bits

Imagen normal

Su entropía es de 8.31 bits

Imagen subexpuesta

Su entropía es de 6.90 bits

Su entropía es de 10.83 bits

Su entropía es de 4.54 bits

FIGURA 3-11: Imágenes DC y sus correspondientes entropías

Sin embargo, aunque los resultados de estos ejemplos son coherentes con la calidad de la imagen, se observó que existían casos en que esto no se cumplía. Las imágenes expuestas a continuación se diferencian en cuanto a la homogeneidad de la imagen, e intuitivamente se entiende que la de la izquierda (la menos homogénea) tiene una mayor entropía. En cambio, los resultados obtenidos no se corresponden con esta teoría.

Su entropía es de 7.74 bits (cuando a nivel de píxel era de 9.17 bits)

Su entropía es de 7.85 bits (cuando a nivel de píxel era de 8.72)

FIGURA 3-12: Entropía de imágenes problemáticas

FIGURA 3-13: Imágenes de coeficientes DC de las imágenes problemáticas

Observando las imágenes a partir de las que hacemos el cálculo, (las que se forman con los coeficientes DC) tampoco se aprecia una gran diferencia, pero es posible que la reducción produzca un suavizado indeseable. El resultado es que la imagen borrosa tiene una mayor cantidad de colores, contradiciendo los resultados obtenidos en el estudio a nivel de píxel.

Aunque el valor del factor de *blur* es muy diferente para cada una de las imágenes (4.51 y 0.96) y la función de calidad terminaría otorgando una mayor calidad a la imagen nítida (con pesos de 0.5 para ambas medidas, la calidad de las imágenes sería 6.125 y 4.84), la aparición de este problema pone en peligro el análisis de la exposición y homogeneidad, en caso de que se desease estudiar por separado. Este hecho exige el uso de otro factor que compense la pérdida de resolución de la imagen creada a partir de los DC.

Volviendo a las definiciones de homogeneidad y exposición, otro aspecto que caracteriza a este tipo de imágenes es su constancia, es decir, su ausencia de altas frecuencias. Ya que los coeficientes AC miden precisamente estas altas frecuencias, son muy útiles para determinar una nueva magnitud.

3.3.1.2.2 Coeficientes AC

Un bloque DCT de una imagen de buena calidad tiene este aspecto:

$$\begin{bmatrix} 1054 & -99 & -146 & -98 & -41 & 15 & 39 & 26 \\ 2 & 23 & 9 & -5 & -30 & -24 & -22 & -1 \\ -4 & -4 & -6 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 6 & 8 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

FIGURA 3-14: Bloque DCT de una imagen con una buena calidad

No obstante, si la imagen estuviera subexpuesta se obtendría un bloque de este tipo:

$$\begin{bmatrix} 460 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 6 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

FIGURA 3-15: Bloque DCT de una imagen subexpuesta

A la vista de los ejemplos se comprueba que existe una distinción entre los bloques DCT de una imagen de alta calidad y una de mala calidad. Este detalle, por tanto, se puede aprovechar para determinar si una imagen tiene las características que se están buscando haciendo uso de los coeficientes AC del bloque DCT.

Para distinguir entre imágenes de buena calidad y sobre/sub expuestas u homogéneas no es necesario utilizar todos los coeficientes AC. La energía de la imagen se concentra en la esquina superior izquierda, con lo que sólo se tendrán en cuenta los coeficientes (0,1), (1,0) y (2,0) para el cálculo (siguiendo el orden de importancia de la secuencia en zigzag).

El algoritmo consiste en sumar estos coeficientes AC en valor absoluto y finalmente dividir esta cantidad entre el número de coeficientes tenidos en cuenta. De esta forma se

descartan los valores que han dado 0 y que indican que la imagen es de baja calidad y de la misma forma se tienen en cuenta los valores de los coeficientes, que también informan sobre la calidad de la imagen.

Al final se obtiene un valor que ayuda a decidir si la imagen tenía las suficientes altas frecuencias como para no considerarse homogénea o sobre/sub expuesta.

Esta nueva medida soluciona el problema surgido con la entropía en el apartado anterior:

La entropía de su imagen DC es de 7.74 bits y su medida a partir de coeficientes AC es 5.55.

La entropía de su imagen DC es de 7.85 bits y su medida a partir de coeficientes AC es 2.90.

FIGURA 3-16: Entropía y medida a partir de coeficientes AC de las imágenes problemáticas

A pesar de que la medida de la entropía no refleja la verdadera calidad de homogeneidad de la imagen, los primeros 3 coeficientes AC resuelven el problema.

3.3.1.3 Función de calidad

Para calcular la función de calidad relacionada con los algoritmos a nivel de bloque se necesitan cuatro valores: los correspondientes a las medidas de *blur* y exposición, y los que determinan el valor de la exposición (entropía y coeficientes AC).

Para establecer los pesos involucrados en el cálculo de la exposición, se trabaja con imágenes problemáticas como las número 21 a 26 del grupo "Internet" del anexo G, e imágenes con distinta exposición como las clasificadas de 1 a 3 y 27 a 36. Tras hacer una serie de pruebas empíricas se estipula que los pesos adecuados para el valor de entropía y coeficientes AC son 0.7 y 0.3 respectivamente.

Para obtener los valores más adecuados para los pesos de *blur* y exposición, se trabaja con las imágenes con las que se estudió para determinar la función de calidad a nivel de píxel (apartado 3.2.1.2). Como el algoritmo que se usaba para determinar la calidad en cuanto a borrosidad es el mismo (DCT), se utilizan los resultados obtenidos al aplicar los métodos a nivel de píxel para orientarse. Según estos criterios, los valores más apropiados son 0.6 y 0.4.

Con estos pesos, la calidad final de las imágenes problemáticas del apartado anterior (21 y 22 del conjunto "Internet" del anexo G) es de 6.05 y 4.20.

Para terminar, el trabajo con las diferentes imágenes de entrenamiento determina que el umbral más conveniente para el umbral de calidad cuando se usan estos algoritmos es de 5.8, con el que se obtienen los siguientes resultados:

Algoritmo	Aciertos (%)	Falsos positivos (%)	Falsos negativos (%)
AC + DCT	89.33	4.67	6

TABLA 3-3: Resultados de entrenamiento del algoritmo de calidad (bloque)

3.3.2 Detección de duplicados

Conseguir algoritmos que funcionaran como los que trabajan a nivel de píxel para la sección de "Filtrado de calidad" no se percibe tan complicado como intentar extraer características representativas de cada imagen para encontrar duplicados.

En este apartado se explican tres métodos que, basándose en coeficientes DC, pruebas anteriores y un poco de imaginación, se aproximan más o menos a representar la imagen a la que se aplican. Así, los Métodos 1 y 2 trabajan en una dirección que es opuesta a la que sigue el Método 3.

A continuación se describen los métodos desarrollados a partir de bloques DCT para probar si dos imágenes son duplicadas.

3.3.2.1 Método 1

El Método 1 surge a partir de las representaciones que se obtenían con el algoritmo del Color Medio y el de *Dominant Color* (apartados 2.3.2.2 y 3.2.2.1 respectivamente). Estas representaciones pretendían incorporar la información de color sin olvidar la información espacial. Las imágenes duplicadas eran aquellas cuya paleta de colores y la distribución de éstos eran, a la vez, parecidas.

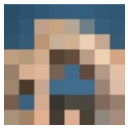
Pues partiendo de esta premisa, el propósito de este algoritmo es utilizar los coeficientes DC (al igual que se hace para calcular la entropía) para crear una imagen más pequeña que sea representativa. Como los coeficientes DC son el valor medio un bloque DCT, construir una imagen con ellos equivale a obtener una representación 1/8 menor en ambos lados con los colores medios de la imagen.

Para deshacerse de la sobrecarga que supone almacenar imágenes tan grandes para su posterior comparación, el Método 1 reduce la representación a un bloque de 8x8 o 16x16 píxeles mediante submuestreo.

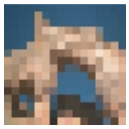
Seguidamente se muestra una imagen original, su correspondiente representación a partir de coeficientes DC y las representaciones finales de tamaños 8x8 y 16x16.

Imagen original

Imagen DC



Representación de tamaño 8x8



Representación de tamaño 16x16

FIGURA 3-17: Procedimiento del Método 1

Es preciso destacar que estos bloques 8x8 o 16x16 son comparados mediante la distancia euclídea para comprobar si sus imágenes respectivas son duplicadas.

Los umbrales más apropiados para considerar que dos imágenes tratadas con este método son duplicadas son 1.4 para el caso de representaciones de 8x8 píxeles, y 2.9 para las de 16x16. Estos resultados se deducen del trabajo empírico con las imágenes de entrenamiento expuestas en el anexo G, que proporcionan los siguientes porcentajes:

Algoritmo	Aciertos (%)	Falsos positivos (%)	Falsos negativos (%)
Mét 1 (8x8)	60	6.48	40
Mét 1 (16x16)	60	6.48	40

TABLA 3-4: Resultados de entrenamiento del Método 1

3.3.2.2 Método 2

El Método 2 es una extensión del Método 1, pero partiendo de la base que se estableció con *Color Layout* (apartado 2.3.2.3). Si con este algoritmo se aplicaba la DCT a la imagen primeramente reducida mediante submuestreo a un tamaño de 8x8 píxeles, se puede partir de la imagen 8x8 construida por el Método 1 para calcular su Color Layout.

Continuando con el ejemplo del apartado anterior, se muestran a continuación las imágenes de los tres bloques DCT obtenidos de cada una de las capas (Y, Cr, Cb) de la imagen DC construida en el método 1.

Imagen DCT de la imagen de la componente Y

Imagen DCT de la imagen de la componente Cr

Imagen DCT de la imagen de la componente Cb

FIGURA 3-18: Imágenes de bloques DCT Y, Cr y Cb

La diferencia entre dos representaciones de este tipo se hace como en 2.3.2.3, es decir, pesando con la matriz de zigzag las distancias entre coeficientes de las imágenes DCT Y, Cr y Cb.

Para terminar, comentar que el umbral elegido para determinar que dos imágenes son duplicadas utilizando este método es 3.3, valor que se infiere tras los ensayos realizados con las imágenes del anexo G, que devuelven estos porcentajes:

Algoritmo	Aciertos (%)	Falsos positivos (%)	Falsos negativos (%)
Mét 2	70	2.78	30

TABLA 3-5: Resultados de entrenamiento del Método 2

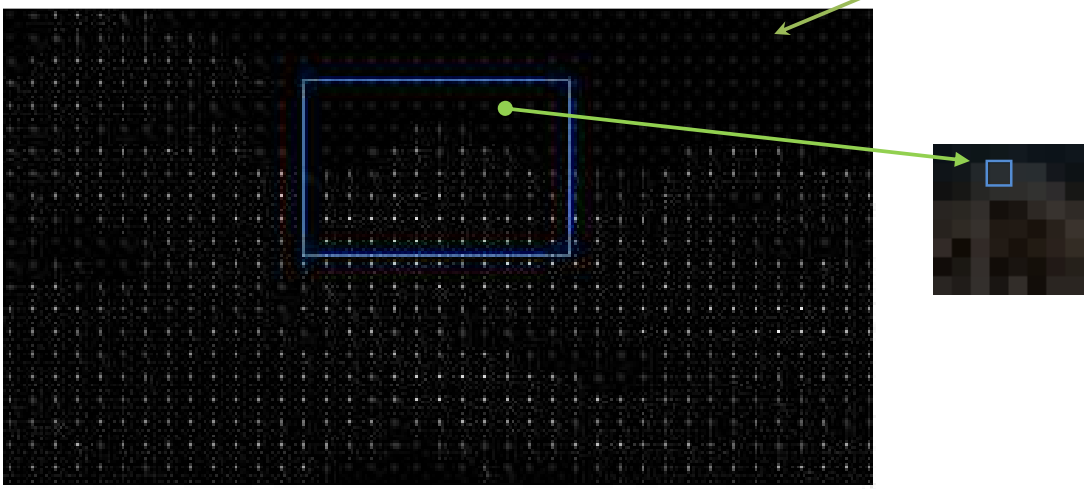
3.3.2.3 Método 3

Si los Métodos 1 y 2 trabajaban a partir de la imagen creada por medio de los coeficientes DC de los bloques DCT y luego realizaban el submuestreo, el Método 3 opera precisamente a la inversa.

Consiste en la realización de un submuestreo promediando los píxeles más cercanos sobre la imagen formada por bloques DCT hasta obtener otra imagen, supuestamente representativa, de tamaño 8x8 píxeles. Esta mezcla de coeficientes DC y AC crea una imagen que en principio sería una aberración y que no serviría para la detección de duplicados, sin embargo, una vez implementado el algoritmo, el descubrimiento de unos resultados favorables indica que la representación obtenida con este método no es tan indeseada.

El ejemplo a continuación muestra el trabajo que realiza este algoritmo desde que recibe la imagen dividida en bloques hasta que obtiene su imagen representativa. La imagen original se corresponde con la de la figura 3-17, por medio de la cual se explicó el Método 1.

Imagen de bloques DCT



Detalle de los bloques que crean el píxel (1,3) e imagen representación obtenida mediante el Método 3

FIGURA 3-19: Procedimiento del Método 3

La distancia entre dos representaciones se mide mediante la distancia euclídea.

Se admite que un par de imágenes son duplicadas si la distancia entre ellas es menor que 0.45, resultado razonado a partir del trabajo con las imágenes de entrenamiento (anexo G), que ofrece estos porcentajes:

Algoritmo	Aciertos (%)	Falsos positivos (%)	Falsos negativos (%)
Mét 3	25	10.19	75

TABLA 3-6: Resultados de entrenamiento del Método 3

A la vista de unos resultados tan poco acertados, se decide suspender la inclusión del Método 3 como técnica de detección de duplicados.

3.4 Aplicación a secuencias de vídeo

El objetivo de esta parte del proyecto es desarrollar un filtrado de calidad que funcione con secuencias de vídeo sin editar. Ello implica que los vídeos tendrán secuencias repetidas, tomas falsas, planos desenfocados, etc. El filtrado se encargará de reconocer cuáles de los *frames* I analizados están sobreexpuestos, subexpuestos, son demasiado homogéneos o están borrosos, lo que permitirá a sistemas de análisis posteriores descartar segmentos de vídeo (en este caso, restringidos a GoP que contengan los *frames* I de baja calidad o duplicados) que no sean de interés. De la misma forma, se detectarán, mediante el análisis de duplicados, las secuencias en las que la cámara se haya quedado grabando sola o las tomas en las que aparezcan elementos indeseados en el vídeo final (claquetas, cartas de ajuste, etc.).

El estándar de vídeo MPEG trabaja, al igual que JPEG, con bloques DCT, por lo que los algoritmos escogidos para aplicarse sobre los *frames* de vídeo serán los propuestos en el apartado anterior (los que operan a nivel de bloque). (Para saber más sobre MPEG, anexo E.)

La obtención de los bloques DCT se realiza gracias a IVOnLA. IVOnLA (Image & Video OnLine-Analysis module) es el módulo que se encarga de la extracción en tiempo real de los descriptores visuales correspondientes a los datos de entrada dentro del sistema MESH [24]. Estos descriptores se dedican a obtener detalles sobre la entrada tan pronto como se deposite información en el "almacén de contenido" del MESH, lo que les convierte en los iniciadores de tareas como la clasificación en tiempo real y la sumarización en tiempo real.

IVOnLA afronta el trabajo en tiempo real mediante un doble sistema. El primer análisis on-line se aplica para evitar retrasos significativos debidos a las dependencias entre medios. Los resultados del análisis se obtienen continuadamente a través de un esquema causal. Mientras que se obtienen los resultados para un *frame k*, sólo se necesita que esté disponible la información resultante de los *frames 0…k-1*. En segundo lugar, se utilizan las técnicas en el dominio comprimido con el fin de aumentar la eficiencia, ya que la información del análisis ya se encuentra codificada en el flujo de bits.

La figura 3-20 representa la arquitectura de IVOnLA. El módulo de entrada acepta tanto vídeos MPEG-2 como ciertos parámetros que controlen algunas operaciones sobre diferentes elementos. Su salida se resume en una descripción del contenido del vídeo, incluyendo su segmentación en tomas (shots), descriptores sobre el movimiento de la

cámara, máscaras de segmentación basadas en movimiento, imágenes DC, coeficientes DCT…

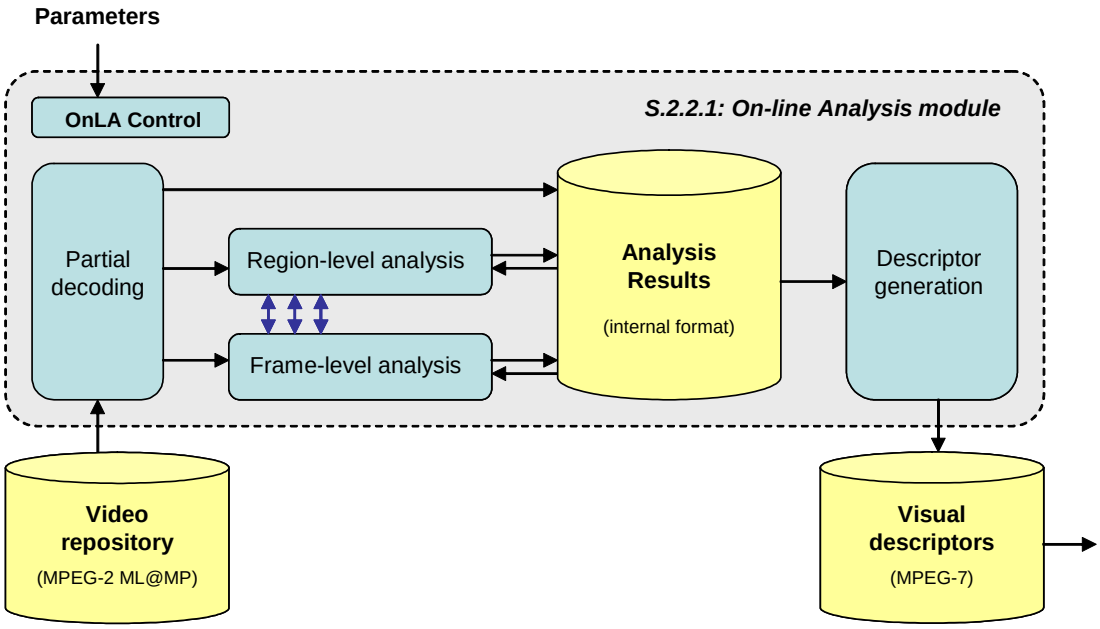


FIGURA 3-20: Arquitectura de IVOnLA

El objetivo final de los algoritmos de vídeo será el de integrarse en IVOnLA como un módulo más, permitiendo la extracción de características como la calidad global de los *frames* I, su coeficiente de borrosidad, su medida en cuanto a la exposición, desde y hasta cuándo se considera que una secuencia es una grabación involuntaria, o si contiene algún elemento característico (por ejemplo, una claqueta).

3.4.1 Análisis de calidad

El análisis de calidad se realiza utilizando algunos de los algoritmos descritos en el apartado anterior, a saber: algoritmo DCT para *frames* borrosos (apartado 3.3.1.1.1) y entropía y coeficientes AC para *frames* sobre/sub expuestos u homogéneos (apartado 3.3.1.2).

Como se señaló anteriormente, entre los resultados que retorna IVOnLA [43] se encuentran el valor de los bloques DCT de cada uno de los *frames* y el valor de los coeficientes DC, a partir de los cuales trabaja el algoritmo.

Las medidas que se devuelven como salida se resumen en: coeficiente de *blur*, valor de exposición (suma pesada de la entropía y la variación de los coeficientes AC) y calidad global (suma ponderada de la exposición y del coeficiente de *blur*). Estos valores se imprimen finalmente en una gráfica para una visualización más cómoda.

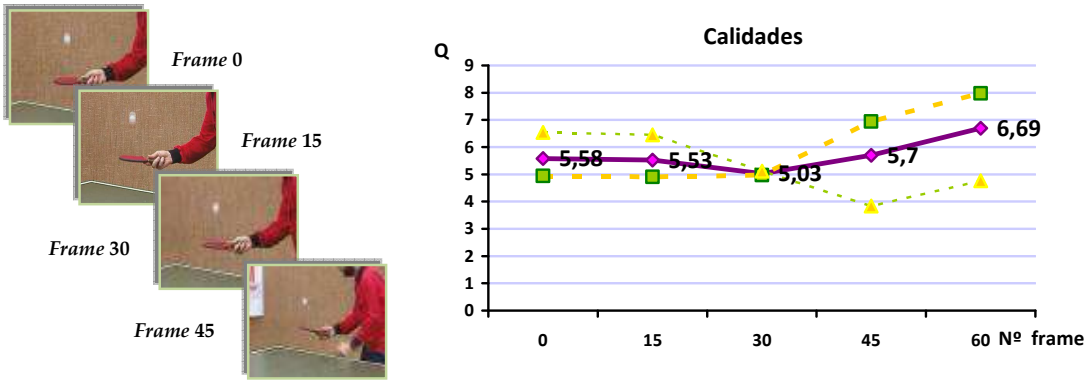


FIGURA 3-21: *Frames* I analizados y su correspondiente gráfica de calidades

El umbral por debajo del cual un *frame* debería considerarse de baja calidad, es decir, sería recomendable su exclusión del vídeo, depende en mucho del contenido de la cinta. Así, si una grabación se realiza en entornos oscuros, o con fondos uniformes, el umbral debe ser más bajo. Como en imágenes se trabajó con un umbral de 5.8, se recomienda este mismo valor para unos resultados generales.

3.4.2 Detección de duplicados

En el estudio de duplicados se ha de tener en cuenta la característica temporal de los vídeos. El sentido de este examen radica en la capacidad de poder distinguir cuándo la cámara se ha quedado grabando sola o si una toma se considera indeseada (este ejemplo de uso está actualmente dentro de las actividades de TRECVID dentro del desafío *TRECVID BBC Rushes Summarization* [25]), para poder eliminarlas en un paso posterior.

3.4.2.1 Grabación involuntaria

Una grabación involuntaria se puede caracterizar por la mala calidad de sus *frames* (escenas desenfocadas, objetivo apuntando al suelo, al techo…), lo cual se detectaría con los algoritmos anteriores de análisis de calidad, pero también es posible que los *frames* no sean defectuosos, es decir, que la cámara se quede grabando una escena sin que en ésta suceda nada.

Para este segundo caso es ideal la detección de duplicados. Cuando una reproducción de vídeo permanece estable durante un tiempo, el público comienza a inquietarse. Este tiempo dependerá de las características del vídeo pero, tras probar con los vídeos de entrenamiento, se sugiere que sea de unos 10 segundos. Estos 10 segundos se traducen aproximadamente en un conjunto de 16.67 *frames* I, si consideramos una velocidad de visionado de 25 *frames*/s y GoPs de 15 *frames*. Durante una grabación involuntaria, los *frames* I prácticamente no cambian, lo que hace innecesaria una comparación exhaustiva.

Para averiguar si una secuencia es una grabación involuntaria, se realiza una detección de duplicados entre el *frame* actual y el anterior mediante cualquiera de los métodos descritos en el apartado 3.3.2, sobre una totalidad de 17 *frames* I (que se traducen en 10.2 segundos de vídeo). Si un gran número de análisis da positivo, se puede decir que la secuencia ha

sido grabada involuntariamente. Este porcentaje de aciertos por encima del cual se puede considerar que la cámara se ha quedado grabando sola es también un parámetro con el que el usuario puede ajustar el programa a las necesidades del vídeo. Las pruebas con los vídeos de entrenamiento sugieren que sea del 100%.

Aunque este método consigue la evaluación de un total de 18 segundos sin tener una gran penalización en el cálculo, es inevitable que exista un retardo con respecto a los resultados que son devueltos desde el programa a IVOnLA [43] ya que, si la secuencia se considera una grabación involuntaria, se retorna información sobre el *frame* I *k* cuando se termina de analizar el *frame* I *k+17*.

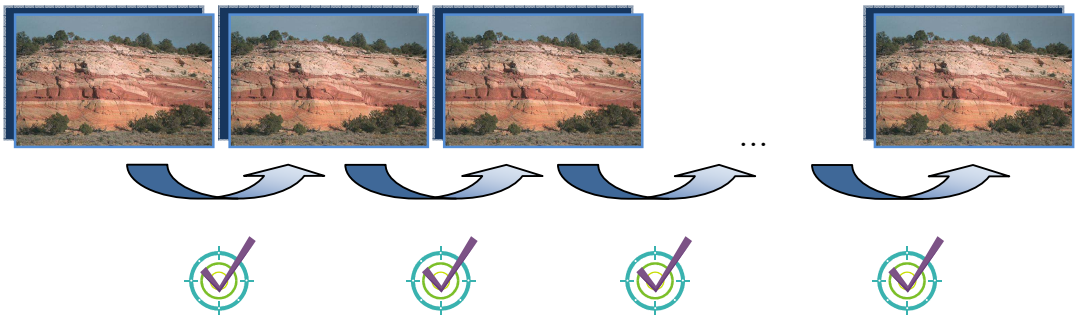


FIGURA 3-22: Secuencia de duplicados. Un gran número de positivos implica que la cámara se quedó grabando sola.

En la comparación hay que asegurarse de que el umbral es lo suficientemente bajo, ya que hay casos en los que la escena contiene pocos cambios (por ejemplo, una conversación entre dos personas) pero no ha de considerarse una grabación involuntaria. Para el Método 1, con tamaño de imagen representativa de 8x8 píxeles se sugiere un umbral de 0.4; para el mismo algoritmo, pero con representación de 16x16 píxeles, uno de 0.75; y, por último, el Método 2 funciona mejor con uno de 0.75.

3.4.2.2 Repetición de tomas

Cuando una toma se repite, el director lo indica grabando, antes de la toma repetida, una claqueta. En ésta se refleja información como el número de toma, la escena que se repite, los nombres del productor y del director, la cámara que se prepara a grabar la secuencia, la fecha, etc.

FIGURA 3-23: Algunos ejemplos de claquetas

En el proceso de sumarización, estas tomas de claquetas se incluyen en el resumen indeseablemente debido a que son muy distintas al resto del vídeo. Es muy útil, por tanto,

que la aplicación de vídeo realice, como parte del preprocesado, una eliminación de este tipo de tomas, al igual que, para otras aplicaciones, posibles cortinillas, créditos, cartas de ajuste, etc.

La idea es crear una base de datos de claquetas lo suficientemente representativa que permita la detección de otras claquetas distintas mediante la detección de duplicados. La representación de cada una de las claquetas de la base de datos se guarda en un archivo de forma que, cuando se ha calculado la representación del *frame* actual, pueda compararse con cada una de ellas.

El umbral, en este caso, ha de ser lo más alto posible, a fin de intentar detectar las escenas con claquetas por muy diferente que sea el tipo de claqueta y el ambiente en el que se encuentran.

El vídeo de entrenamiento con el que se han obtenido los umbrales más adecuados para los tres métodos de detección de duplicados cuenta con 31 elementos de corte: barras de colores al inicio, tramos en blanco y claquetas. La base de datos se compone de una imagen de carta de ajuste, dos imágenes de barra de colores, una imagen en blanco y otra en negro y quince imágenes de distintas claquetas, ninguna perteneciente a las cintas de entrenamiento o prueba.

Los umbrales que se han considerado los más adecuados son: 1.5 para el Método 1 (8x8), 3.2 para el Método 1 (16x16) y 4.2 para el Método 2. Para describir los resultados obtenidos se utilizan 4 títulos, con el fin de reflejar el hecho de que, aunque haya veces que la claqueta no se detecte, sí se descubre el corte en blanco o negro que la precede. Así, en "Detecciones" se incluyen estos sucesos que no pueden aceptarse del todo en "Aciertos".

	Detecciones	Aciertos	Fallos	Falsos positivos
Método 1 (8x8)	27	19	4	3
Método 1 (16x16)	27	21	4	4
Método 2	28	26	3	20

TABLA 3-7: Resultados del entrenamiento de detección de claquetas

4 Desarrollo

4.1 Introducción

En este capítulo de Desarrollo se describe la implementación de los algoritmos descritos en el apartado anterior. Para comenzar, se hace un escueto resumen sobre las herramientas y formatos utilizados para su realización, para posteriormente describir cada implementación en detalle.

4.2 Herramientas y formatos

4.2.1 C++

En 1983 el lenguaje C fue rediseñado, extendido y nuevamente implementado dando lugar a C++. Además del concepto nuevo de "clase", se crearon las funciones virtuales, funciones sobrecargadas y operadores sobrecargados. Posteriormente, C++ ha sido ampliamente revisado y refinado dando lugar a nuevas características como herencia múltiple, funciones miembro, plantillas referidas a tipos y manipulación de excepciones.

Según [19], C++ es un lenguaje híbrido que, por una parte, ha adoptado todas las particularidades de la programación orientada a objetos y, por otra, mejora sustancialmente las capacidades de C. Esto dota a C++ de una potencia, eficacia y flexibilidad que lo convierten en un estándar dentro de los lenguajes de programación orientados a objetos.

Todas las aplicaciones desarrolladas están implementadas en este lenguaje ya que su utilización es muy atractiva: es rápido, es una extensión de C y permite un amplio control de la memoria.

4.2.2 Microsoft Visual Studio 2005

Microsoft Visual Studio es un entorno de desarrollo integrado (IDE) para sistemas Windows. Soporta varios lenguajes de programación tales como Visual C++, Visual C#, Visual J#, ASP.NET y Visual Basic .NET, aunque actualmente se han desarrollado las extensiones necesarias para muchos otros.

La elección de esta plataforma de desarrollo llevó a la obligación de hacer uso de las MFCs para el desarrollo de la interfaz gráfica, de forma que se hace imposible su ejecución en otros sistemas operativos que no sean Windows.

4.2.3 OpenCV

Como se define en su principal página web [20], OpenCV es una librería cuyas funciones están principalmente dirigidas a las técnicas de procesamiento de imágenes y *computer vision* en tiempo real.

Tiene la ventaja de ser una librería abierta (desarrollada por Intel) y ha prestado ayuda en este proyecto proporcionando funciones de alto nivel para el procesado de imágenes como: operaciones básicas de apertura, creación, acceso a píxeles…, filtrado, reducción mediante submuestreo, cambio de espacio de color, aplicación de la DCT, etc.

4.2.4 JPEG y campos EXIF

Las imágenes con las que trabajan las aplicaciones son imágenes JPEG. JPEG es un estándar internacional (desarrollado conjuntamente por los comités CCITT e ISO) de compresión de imagen digital, tanto para imágenes en escala de grises como de color. El conocimiento del funcionamiento de este estándar es importante para la parte 2 del proyecto, donde se necesita realizar la extracción de los bloques DCT sin tener que decodificar la imagen completa, pero también en la parte 1 para la obtención de los campos EXIF. (Para más información sobre JPEG véase el anexo D.)

EXIF es una cabecera donde se insertan metadatos según el estándar JPEG. En ellos se almacena información como el tipo de cámara, la resolución de la imagen, el copyright, la orientación, tiempo de exposición, etc. La utilidad de los campos EXIF en este proyecto radica en el campo de fecha y hora de toma de la imagen, que es necesario para agrupar las imágenes en el tiempo y hacer una detección de duplicados más eficiente.

En el anexo F se describen con más detalle el formato que tienen estas etiquetas, la estructura y localización de los campos y cómo se realiza la extracción de la fecha de creación de la fotografía.

4.3 Algoritmos

En este capítulo de Desarrollo se relata cómo se ha realizado la implementación de los algoritmos descritos hasta este punto en los capítulos de Estado del Arte y Diseño, que se consideran lo suficientemente óptimos para su inclusión en alguna aplicación.

4.3.1 Nivel de píxel

En la aplicación que trabaja a nivel de píxel, las imágenes se decodifican con la función *cvLoad* de *OpenCv*, que devuelve un puntero a *IplImage*. *IplImage* es una estructura que almacena datos como el ancho y el alto de la imagen en píxeles, su número de canales (1 equivale a escala de grises, 3 a espacios de color como RGB, HSV, YCrCb…), el número de bits que se utilizan para guardar cada color, etc. y un puntero a los datos de la imagen. El valor de los píxeles se almacena en esta cadena por filas y, dependiendo del espacio de color, en un orden determinado.

Accediendo a esta estructura de imagen y, más en concreto, a la cadena de datos, se obtienen los valores de los píxeles necesarios para operar con los algoritmos.

4.3.1.1 Cálculo del factor de calidad

En este apartado se engloban los algoritmos incluidos en la aplicación para el cálculo del factor de calidad. Mientras que Haar, Sobel y DCT se reemplazan unos a otros para obtener diferentes resultados para el coeficiente de *blur*, la entropía siempre está presente.

4.3.1.1.1 Haar

En el apartado 2.3.1.1.1 se describen los pasos que permiten calcular el coeficiente de *blur* haciendo uso de la transformada de Haar y en el apéndice B se explica brevemente la teoría alrededor de las transformadas wavelets, pero implementar una mecánica para obtener la transformada de Haar de tres niveles necesaria para el algoritmo es mucho más sencillo.

Para calcular la transformada de Haar se aplica la secuencia representada en el diagrama de módulos siguiente:

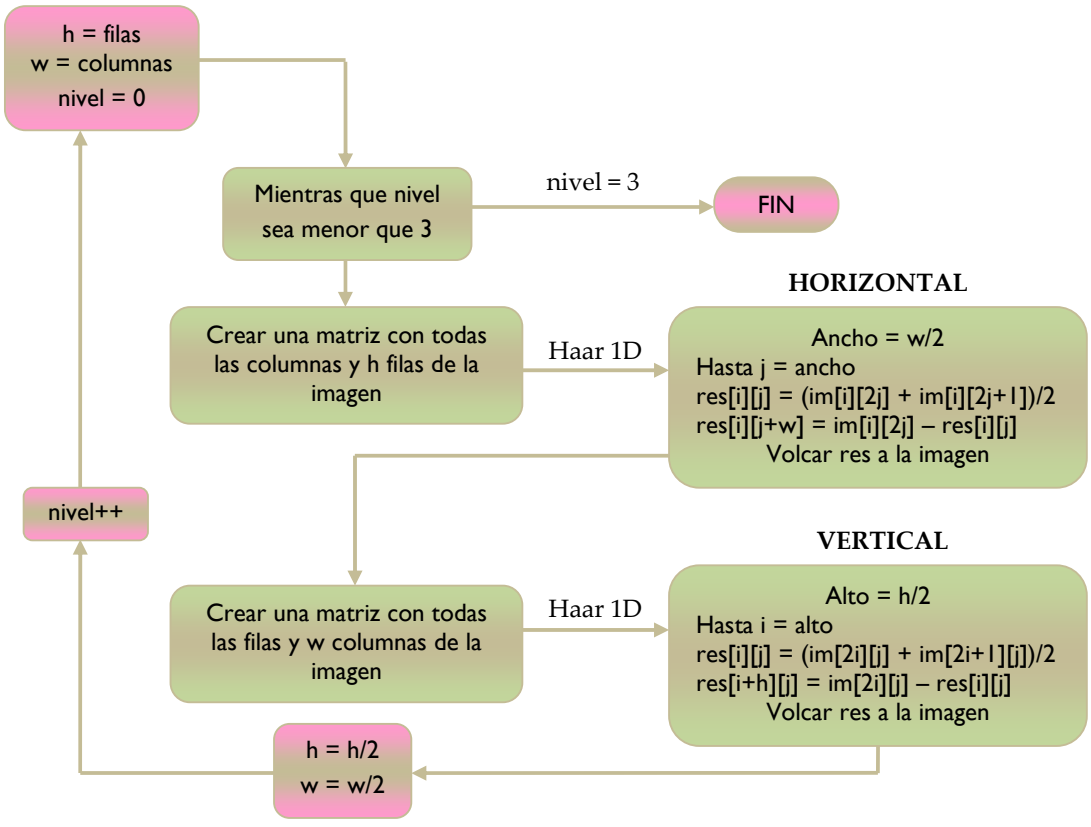


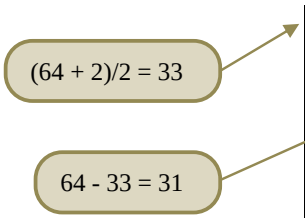
FIGURA 4-1: Diagrama de módulos para calcular la transformada de Haar de tres niveles de una imagen

La transformada de Haar de una imagen en escala de grises de un tamaño de 8x8 píxeles se realizaría de la siguiente manera:

64	2	3	61	60	6	7	57
9	55	54	12	13	51	50	16
17	47	46	20	21	43	42	24
40	26	27	37	36	30	31	33
32	34	35	29	28	38	39	25
41	23	22	44	45	19	18	48
49	15	14	52	53	11	10	56
8	58	59	5	4	62	63	1

FIGURA 4-2: Imagen 8x8 en escala de grises

Con *nivel* = 0, *h* = 8 y *w* = 8, la imagen que se pasa al algoritmo de ejecución de la transformada Haar 1D horizontal es la imagen entera. La primera mitad (hasta *ancho* = *w*/2) de la matriz *res* se rellenaría con la media de los píxeles emparejados por filas, y la segunda mitad con la resta del primer píxel de la pareja menos el resultado anterior. De esta forma, la imagen que se devuelve sería la siguiente:



33	32	33	32	31	-29	27	-25
32	33	32	33	-23	21	-19	17
32	33	32	33	-15	13	-11	9
33	32	33	32	7	-5	3	-1
33	32	33	32	-1	3	-5	7
32	33	32	33	9	-11	13	-15
32	33	32	33	17	-19	21	-23
33	32	33	32	-25	27	-29	31

FIGURA 4-3: Imagen 8x8 tras la primera ejecución en horizontal

Como *w* y *h* siguen teniendo los mismos valores, la imagen anterior entera se transfiere al algoritmo que aplica la transformada Haar en vertical. Los píxeles se emparejan por columnas.

32.5	32.5	32.5	32.5	4	-4	4	-4
32.5	32.5	32.5	32.5	-4	4	-4	4
32.5	32.5	32.5	32.5	4	-4	4	-4
32.5	32.5	32.5	32.5	-4	4	-4	4
0.5	-0.5	0.5	-0.5	27	-25	23	-21
-0.5	0.5	-0.5	0.5	-11	9	-7	5
-0.5	0.5	-0.5	0.5	-5	7	-9	11
0.5	-0.5	0.5	-0.5	21	-23	25	-27

FIGURA 4-4: Imagen 8x8 tras la primera ejecución en vertical

El paso consecutivo es dividir *w* y *h* entre 2, con lo que pasarían a valer 4. El nivel de cálculo de la transformada es el 2. En esta iteración del algoritmo sólo se procesa la mitad de las imágenes.

32.5	32.5	0	0	4	-4	4	-4
32.5	32.5	0	0	-4	4	-4	4
32.5	32.5	0	0	4	-4	4	-4
32.5	32.5	0	0	-4	4	-4	4
0	0	0.5	0.5	27	-25	23	-21
0	0	-0.5	-0.5	-11	9	-7	5
0	0	-0.5	-0.5	-5	7	-9	11
0	0	0.5	0.5	21	-23	25	-27

32.5	32.5	0	0	0	0	0	0
32.5	32.5	0	0	0	0	0	0
0	0	0	0	4	-4	4	-4
0	0	0	0	4	-4	4	-4
0	0	0.5	0.5	27	-25	23	-21
0	0	-0.5	-0.5	-11	9	-7	5
0	0	-0.5	-0.5	-5	7	-9	11
0	0	0.5	0.5	21	-23	25	-27

FIGURA 4-5: Imágenes 8x8 después de la segunda ejecución horizontal y vertical.

De nuevo se dividen *w* y *h*, tomando un valor de 2, y se llega al tercer y último nivel.

32.5	0	0	0	0	0	0	0
32.5	0	0	0	0	0	0	0
0	0	0	0	4	-4	4	-4
0	0	0	0	4	-4	4	-4
0	0	0.5	0.5	27	-25	23	-21
0	0	-0.5	-0.5	-11	9	-7	5
0	0	-0.5	-0.5	-5	7	-9	11
0	0	0.5	0.5	21	-23	25	-27

32.5	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	4	-4	4	-4
0	0	0	0	4	-4	4	-4
0	0	0.5	0.5	27	-25	23	-21
0	0	-0.5	-0.5	-11	9	-7	5
0	0	-0.5	-0.5	-5	7	-9	11
0	0	0.5	0.5	21	-23	25	-27

FIGURA 4-6: Imágenes 8x8 después de la tercera ejecución horizontal y vertical. La de la derecha es la transformada de Haar de tres niveles de la imagen de origen.

Otro detalle importante a tener en cuenta es que tanto el ancho como el alto de la imagen tienen que ser divisibles entre 16, que es el tamaño final que tendrán los Emax. Para ello se repiten la última columna y la última fila el número de veces necesario, aunque este aumento de la imagen altere un poco los resultados.

Imagen original (600x403)

Imagen agrandada (608x416)

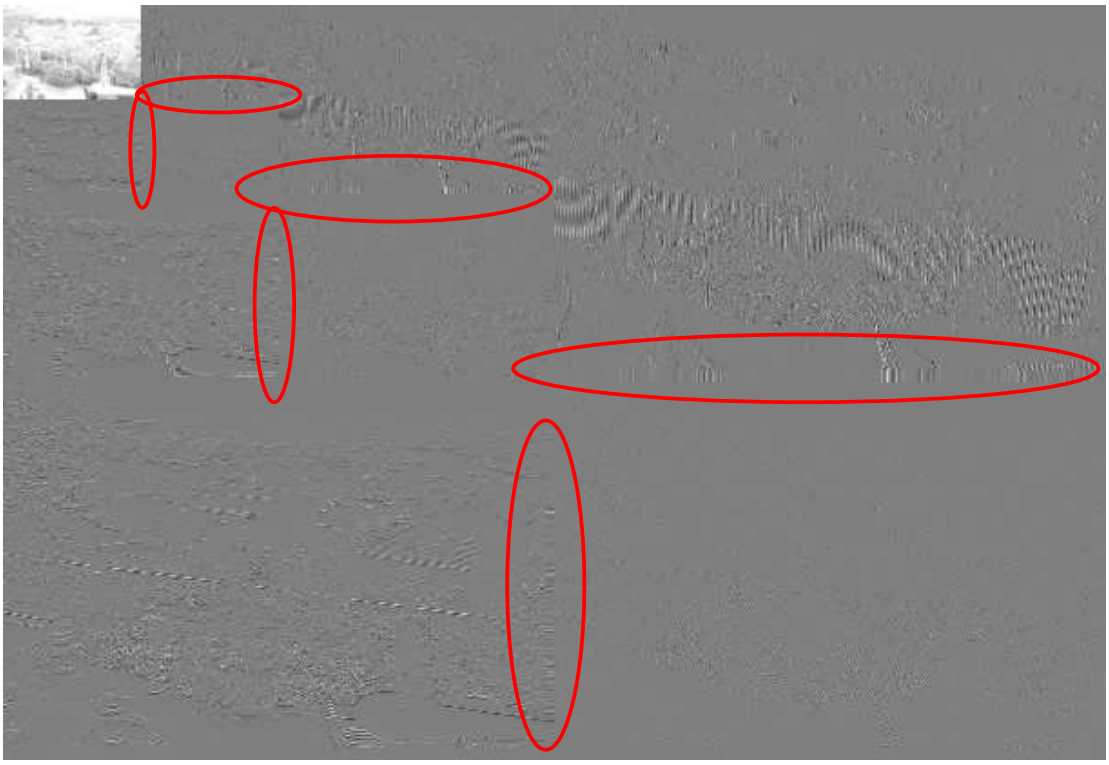


FIGURA 4-7: Transformada de Haar de la imagen agrandada. Debido a la exigencia de agrandar la imagen, se crean bordes inexistentes que adulteran el resultado.

Por último, para obtener el factor de calidad es necesario realizar una operación más. Como el valor final que se obtiene es un porcentaje de la superficie que se considera borrosa en la imagen, hay que restarle al 100% de borrosidad el factor de *blur* calculado, con el fin de informar sobre el porcentaje que no está borroso. Además de esto, hay que tener en cuenta los rangos en los que se mueven las medidas calculadas: si la entropía de una imagen de malísima calidad ronda 1 bit, y la de un de calidad excelente los 10-12 bits, lo normal es que el factor de calidad de *blur* se multiplique por diez para alcanzar el mismo rango de medida que la entropía.

4.3.1.1.2 Sobel

Del algoritmo de Sobel (apartado 3.2.1.1) describir únicamente dos detalles: qué tipo de filtro y bajo qué condiciones se utiliza, y porqué a pesar de seguir el mismo método que se aplicaba con el algoritmo de la transformada de Haar, la imagen no se aumenta con este método.

Como se explicó en el apartado correspondiente al algoritmo de Sobel, las imágenes con ruido del tipo sal y pimienta se ven favorecidas en el cálculo del factor de *blur* debido a que el operador de Sobel magnifica estos pequeños errores. Para impedir que el ruido falsee el resultado, las imágenes con un factor de *blur* superior a 9.75 se suponen ruidosas y se filtran. El filtro utilizado es un filtro gaussiano de tamaño 3x3, implementado en OpenCv con la función *cvSmooth*, con una desviación típica de 0.95.

Al igual que en el algoritmo de la transformada de Haar, la imagen es dividida consecutivamente a medida que se aumentan los niveles. En el caso de Haar, este hecho provocaba un necesario aumento del número de filas y columnas de la imagen original hasta que éstos fueran divisibles entre 16. Pero en el caso de Sobel, el agrandamiento no es preciso.

La diferencia radica en el mismo funcionamiento de los operadores. Con Sobel, la imagen que se filtra no tiene porqué tener un tamaño determinado, mientras que con la transformada, debido al cálculo por mitades descrito en el apartado anterior, es obligatorio garantizar la divisibilidad.

Debido a que el algoritmo de Sobel se basa en el muestreo para obtener las imágenes de los distintos niveles, la pérdida de un par de filas y/o columnas a la hora de construir estas imágenes de niveles se compensa con el tiempo de proceso ahorrado al no tener que expandir la imagen original. En cierto modo, la supresión de algunos píxeles puede ser tan perniciosa como la añadidura.

De esta forma, aunque lo ideal sea que las dimensiones de la imagen original sean divisibles entre 8 (en el algoritmo de Sobel el primer nivel tiene el tamaño de la imagen original, en Haar el primer nivel ya estaba reducido a la mitad), si no se da el caso no se añaden píxeles adicionales.

Así, si la fotografía tiene un ancho de 37 píxeles, el ancho del nivel 2 será de 18 y el de nivel 3, de 9. Cuando se realice el enventanado a los Emaps, el ancho de $Emap_1$ se dividirá entre 8, el de $Emap_2$ entre 4 y el de $Emap_3$ entre 2, consiguiendo que el ancho de los Emax sea en todos los casos 4.

Al tratarse igualmente de un porcentaje, el factor de calidad de *blur* también se multiplica por diez antes de devolverse.

4.3.1.1.3 DCT

El cálculo de la DCT se realiza a partir de los píxeles de la imagen. Para ello, la imagen ha de ser divisible en celdas de 8x8 píxeles. En caso de no ser así, se repite la última fila y/o la última columna el número de veces que sea necesario.

A medida que se van obteniendo bloques 8x8, se hace uso de la función *cvDCT* de OpenCv (que devuelve la DCT del bloque) y se continúa con el resto del algoritmo como se describió en el apartado 2.3.1.1.2 del Estado del Arte, conservando los valores empíricos recomendados en el artículo.

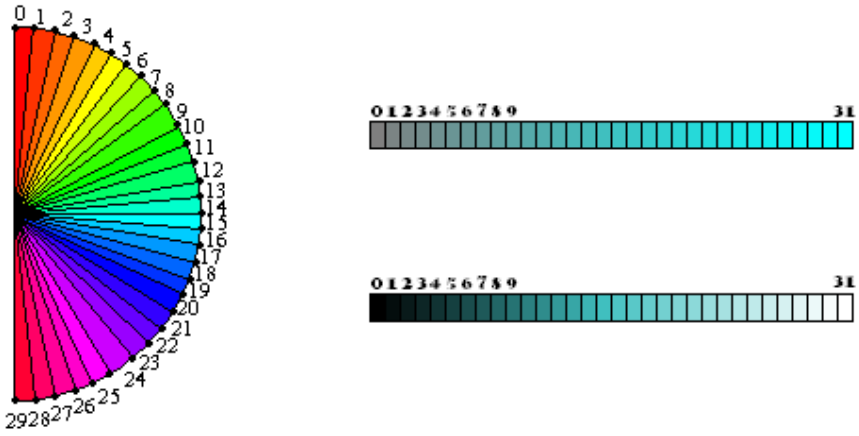
De la misma forma que sucedía en el caso del operador de Sobel, el ruido de sal y pimienta desvirtúa el valor de calidad de *blur* que devuelve el algoritmo. Dado el caso, las imágenes con un factor de calidad de *blur* superior a 7 también son filtradas con el filtro gaussiano.

4.3.1.1.4 Entropía

El algoritmo del cálculo de la entropía (apartado 2.3.1.2.1) basa su funcionamiento en la obtención de un histograma de colores que, normalizado de forma que la suma de sus valores sea igual a uno, simule el comportamiento de la función densidad de probabilidad de colores de la imagen.

El primer paso hacia la construcción del histograma es la transformación de los píxeles de la imagen al espacio de color HSV (en el anexo C se puede localizar más información acerca de los espacios de color). La librería OpenCv cuenta con la función *cvCvtColor* para este tipo de transformaciones, la cual establece el valor de H como números enteros en el rango 0-180 (donde 180 representaría los 360º que esta variable puede llegar a valer) y S y V en el de 0-255 (donde 255 representaría el 100%).

Debido a que el estudio de 11704500 (180*255*255) colores es excesivo para la obtención de resultados óptimos, el número de intervalos tenidos en cuenta en el histograma se reduce a 30720: 30 tonos de H y 32 para S y V.



Divisiones de H **Divisiones de S y V**

FIGURA 4-8: Divisiones de H, S y V

Si un color no pertenece a ninguno de estos valores, se cuenta como igual al color del intervalo inmediatamente inferior (por facilidad de programación). Por ejemplo, el color (45, 203, 185) se igualará al (42, 200, 184) y se contabilizará como presente en el intervalo (7, 25, 23).

FIGURA 4-9: El cambio de color debido a los intervalos es prácticamente inapreciable

De esta forma, el histograma de color de una imagen se puede entender como una tabla de valores en la que 3 cantidades (H, S y V) determinan una posición, es decir, un color diferente. Para su elaboración, se van extrayendo uno a uno los píxeles de la imagen

convertida al espacio HSV y, tras calcular a qué intervalo pertenecen, se suma una unidad a ese espacio.

El diagrama siguiente ilustra sobre la construcción del histograma de color y el cálculo de la entropía.

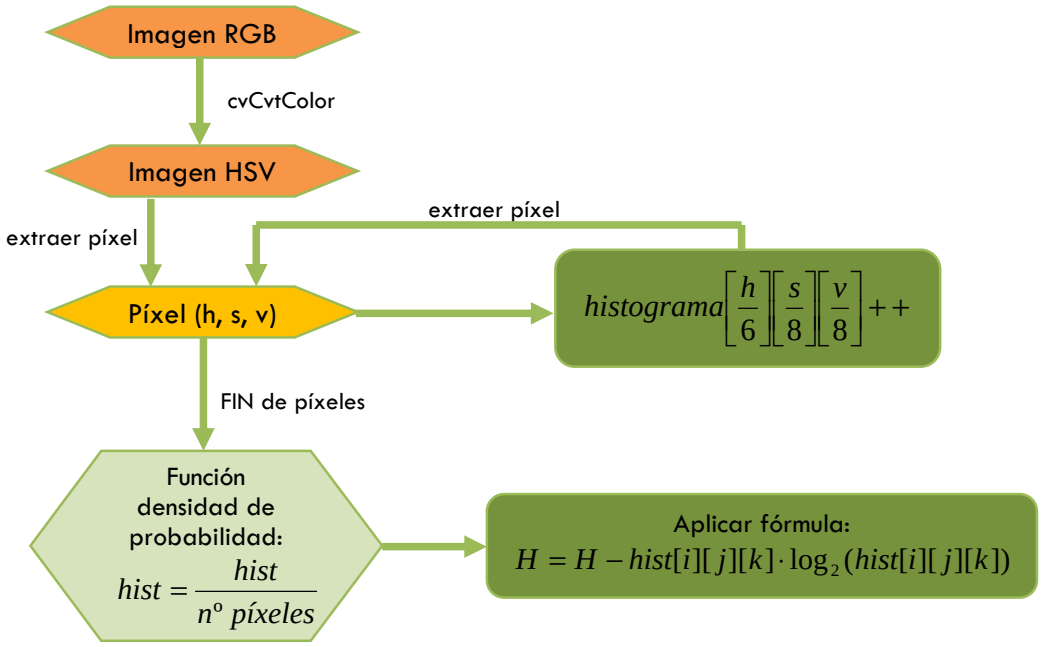
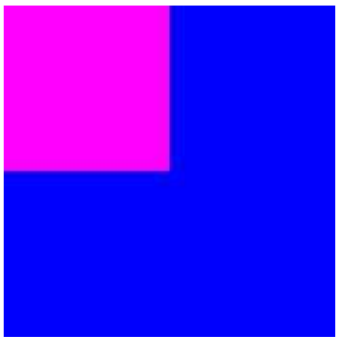


FIGURA 4-10: Diagrama de módulos reflejando el cálculo de la entropía

Para concluir el apartado, se comprueba que el funcionamiento del desarrollo llevado a cabo es el correcto partiendo de una imagen artificial como ejemplo.



$$H = -\sum_{i=0}^{30}\sum_{j=0}^{32}\sum_{k=0}^{32} hist[i][j][k]\log_2(hist[i][j][k]) = -\frac{3}{4}\log_2\left(\frac{3}{4}\right) - \frac{1}{4}\log_2\left(\frac{1}{4}\right) = 0.811$$

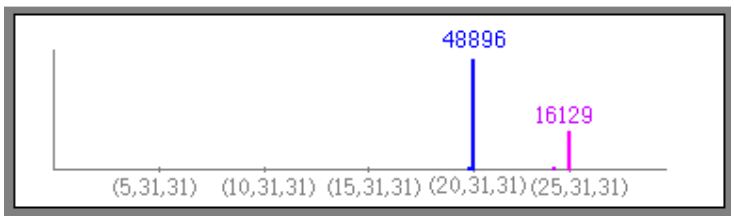


FIGURA 4-11: El valor que devuelve el algoritmo es de 0.88 bits. La diferencia entre el valor teórico y el práctico radica en el histograma, que descubre que los colores de la imagen no son completamente puros: el 74.61% (48896/65536) de los píxeles es azul, el 24.61% (16129/65536) es rosa, y hay menos de un 0.78% (mitad rosados, mitad azulados) que se localizan en el borde entre los dos colores.

4.3.1.2 Detección de duplicados

Los algoritmos implementados para la realización de la detección de duplicados a nivel de píxel en las posteriores aplicaciones son el del Color Medio, el *Dominant Color* y el *Color Layout*.

4.3.1.2.1 Color Medio

Se decide desarrollar el algoritmo de Color Medio (apartado 3.2.2.1) en la aplicación debido a su gran simplicidad, rapidez y a sus buenos resultados.

Este algoritmo calcula el color medio de cada una de las componentes (RGB) de una celda de 8x8 píxeles de tamaño, para lo cual es necesario dividir la imagen en bloques. Al contrario de lo que sucedía con el algoritmo de la DCT, por ejemplo, no es necesario agrandar o reducir la imagen. Si se diera el caso de que la imagen no fuera divisible en bloques de 8x8 píxeles, existirían tres acciones a llevar a cabo: eliminar las filas y/o columnas sobrantes, con lo que se perdería una fila y/o columna de la imagen representación final; repetir la última fila y/o columna hasta obtener un bloque, lo que falsearía el color medio obtenido, debido a que primarían los colores copiados; o, simplemente, obtener los colores medios de los "cuasi bloques", aunque no lleguen a ser la media de 64 colores.

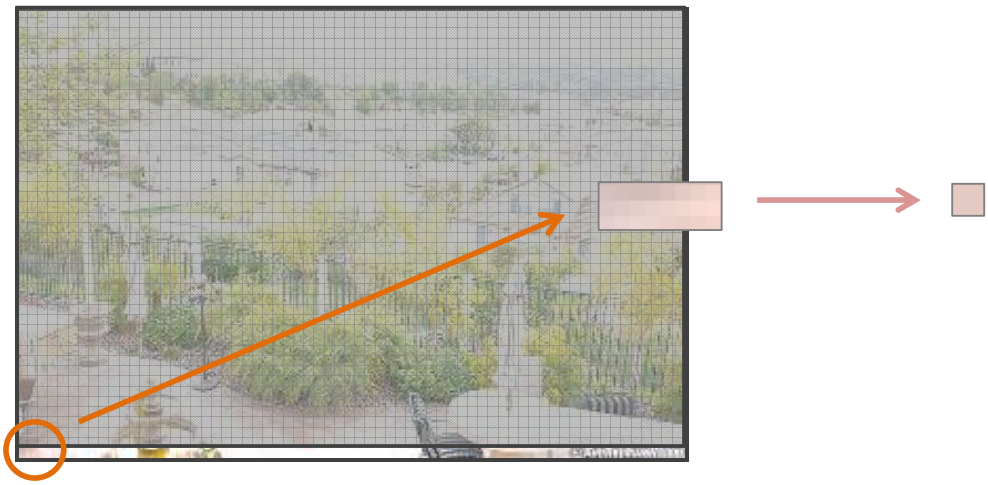


FIGURA 4-12: Las 403 filas de la imagen no son divisibles en bloques de 8x8 píxeles. Las tres últimas filas se agrupan en bloques de 3x8 píxeles, a los que se les aplica el Color Medio

Otra cuestión a resolver por el programa es cómo comparar fotografías que tengan distintos tamaños. Aunque lo lógico es que las fotografías de un mismo día pertenezcan a la misma cámara, y por tanto tengan las mismas dimensiones, es posible que, ante un evento que lo merezca, se hayan almacenado fotos de varios fotógrafos. La imagen representación Color Medio se almacena en la variable *tabla* como una cadena de *Colores*, siendo *Colores* un objeto capaz de almacenar 3 *double*. Si las representaciones tienen distintas longitudes, es decir, si una imagen es más grande que la otra, se disminuye la imagen de mayor tamaño para que alcance las dimensiones de la menor, como si se tratase de un escalado de ésta. Esta operación se realiza mediante la función *cvResize* de OpenCv.

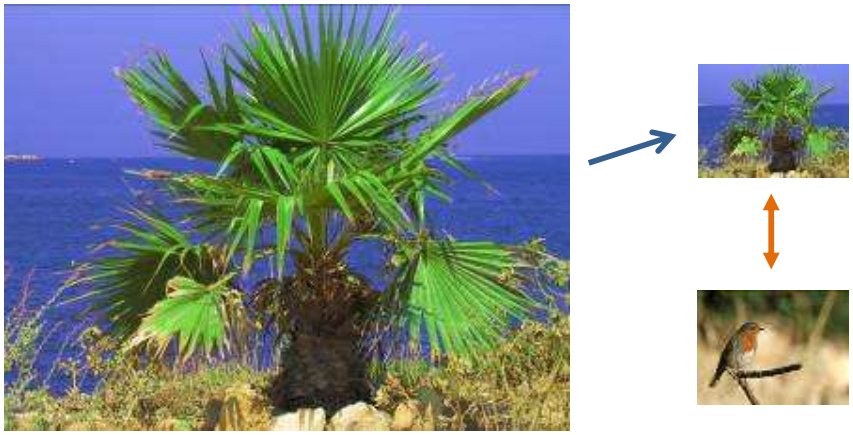


FIGURA 4-13: Las imágenes originales tienen unas dimensiones de 2592x1944 y 640x480 píxeles cada una. Para compararlas, se reduce la de mayor tamaño.

Por último, las fórmulas utilizadas para la conversión de los píxeles del espacio RGB al espacio CIE Luv (anexo C para más información) para calcular la distancia euclídea entre dos representaciones son las que se exponen a continuación:

$$L = \begin{cases} 116\sqrt[3]{y_n} - 16 \; si \; y_n < 0.008856 \\ 7.787 y_n + \dfrac{16}{116} \; si \; y_n \le 0.008856 \end{cases} \qquad u = 13 \cdot L \cdot (u' - u'_n) \qquad v = 13 \cdot L \cdot (v' - v'_n)$$

Donde:

$$y_n = \frac{Y}{Y_n} \qquad u' = \frac{4X}{X + 15Y + 3Z} \qquad v' = \frac{9X}{X + 15Y + 3Z} \qquad u'_n = \frac{4X_n}{X_n + 15Y_n + 3Z_n} \qquad v'_n = \frac{9X_n}{X_n + 15Y_n + 3Z_n}$$

$$X = 0.4121R + 0.3576G + 0.1804B \qquad X_n = 0.4121 \cdot 255 + 0.3576 \cdot 255 + 0.1804 \cdot 255$$
$$Y = 0.2127R + 0.7156G + 0.0722B \qquad Y_n = 0.2127 \cdot 255 + 0.7156 \cdot 255 + 0.0722 \cdot 255$$
$$Z = 0.0193R + 0.1192G + 0.9502B \qquad Z_n = 0.0193 \cdot 255 + 0.1192 \cdot 255 + 0.9502 \cdot 255$$

Siendo X_n, Y_n y Z_n la referencia del color blanco en el espacio de color XYZ (ver anexo C).

4.3.1.2.2 Dominant Color[1]

El algoritmo de *Dominant Color* (apartado 2.3.2.2) se encarga de dividir la imagen original en celdas de 8x8 píxeles a partir de las cuales se extrae el color más significativo. Para ello es obligatorio agrandar la imagen en caso de que sus lados no sean divisibles entre 8, haciendo uso de la misma técnica que con el algoritmo de la DCT (apartado 4.3.1.1.3).

La comparación de dos representaciones calculadas con *Dominant Color* se realiza de la misma forma que en el caso del Color Medio (apartado anterior): se convierten los píxeles al espacio CIE Luv y se contrastan con la distancia euclídea, escalando la imagen de mayor tamaño en el caso de que las representaciones tengan diferentes longitudes.

[1] La implementación de este algoritmo es de JCSM.

4.3.1.2.3 Color Layout

En el caso del algoritmo *Color Layout*, el tamaño de la imagen a examinar no es relevante. La reducción de la imagen original a otra de tamaño 8x8 se realiza aplicando la función *cvResize* de OpenCv con el método de submuestreo CV_INTER_LINEAR. Posteriormente se convierte la imagen reducida al espacio YCrCb (anexo C: Espacios de Color) mediante la función *cvCvtColor* para a continuación aplicarle la DCT.

La diferencia entre dos representaciones *Color Layout* (siempre de longitud igual a 64) se realiza aplicando la fórmula descrita en el apartado 2.3.2.3.

4.3.2 Nivel de bloque

El planteamiento de utilizar los bloques DCT en vez de los píxeles surge de la inquietud de ahorrar tiempo de computación. Para ello debía de encontrarse un programa ajeno al proyecto que realizara la decodificación de una imagen JPEG (para más información sobre JPEG, apéndice D) hasta el punto anterior de la aplicación de la IDCT. Lamentablemente, las opciones encontradas tardaban más en obtener los bloques DCT que la función *cvLoad* de OpenCv en decodificar la imagen en píxeles.

Por todo lo cual se decide simular la obtención de los bloques DCT decodificando la imagen con la función *cvLoad*, agrandándola en caso de que no sea divisible en celdas de 8x8 píxeles y convirtiendo las celdas en bloques DCT con la función *cvDCT* de OpenCv.

4.3.2.1 Cálculo del factor de calidad

Los algoritmos elegidos para calcular el factor de calidad son: DCT, para obtener la borrosidad de la fotografía, y la entropía y los coeficientes AC para determinar su exposición y homogeneidad.

4.3.2.1.1 DCT

El algoritmo de la DCT se mantiene sin cambios con respecto al descrito a nivel de píxel en el apartado 4.3.1.1.3.

4.3.2.1.2 Entropía y coeficientes AC

El cálculo de entropía y del peso de los coeficientes AC se realiza al mismo tiempo.

En primer lugar, se crea una imagen cuyas dimensiones son 1/64 menores que las de la fotografía de partida. Esta miniatura será la encargada de recoger el valor de los coeficientes DC de cada una de los componentes divididos entre 8, es decir, cada uno de sus píxeles representará el color medio de cada bloque. A la vez que se va construyendo esta imagen, se recolectan los coeficientes AC de las posiciones (0,1), (1,0) y (2,0) (los tres primeros AC siguiendo el zigzag de la codificación JPEG) y se van sumando paulatinamente hasta el último bloque.

Terminado el montaje de la imagen DC, se calcula la entropía a partir de ella de la misma forma que se realizaba a nivel de píxel (apartado 4.3.1.1.4). El valor de variación de los ACs se obtiene dividiendo la cantidad anteriormente calculada entre el número total de coeficientes AC tenidos en cuenta.

Ambas medidas pasan a integrar el coeficiente de exposición que se pesará junto con el factor de *blur* para adjudicar un valor de calidad a cada imagen.

4.3.2.2 Detección de duplicados

Para la detección de duplicados los algoritmos implementados son los métodos 1, 2 y 3, descritos en profundidad en la sección 3.3.2.

4.3.2.2.1 Método 1

El Método 1 es muy parecido en su desarrollo al algoritmo que calcula la entropía: en ambos se necesita construir la imagen DC para los cálculos. Tras la obtención de la miniatura, se reduce aún más con la función *cvResize* de OpenCv hasta obtener un tamaño de la representación de 8x8 o 16x16 píxeles.

El diagrama de módulos explica el procedimiento que se sigue en este algoritmo:

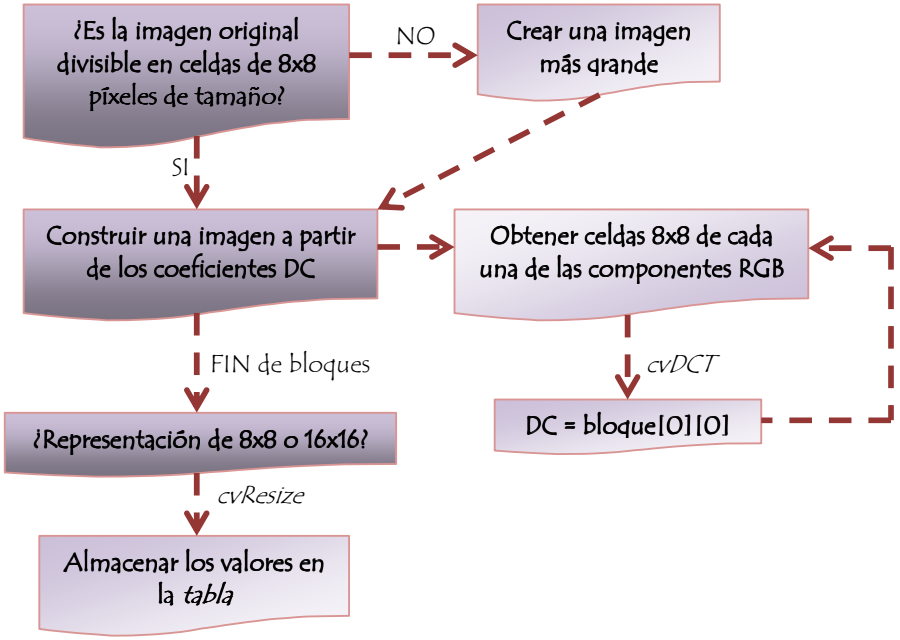


FIGURA 4-14: Diagrama de bloques del algoritmo Método 1

Para calcular la distancia euclídea entre dos representaciones, se convierten los píxeles al espacio CIE Luv, de la misma forma que se operaba a nivel de píxel.

4.3.2.2.2 Método 2

Para desarrollar el Método 2, se aplica primeramente el Método 1 (apartado anterior) forzando que la longitud de la representación obtenida sea de 64 colores. Esta imagen DC se modifica con la técnica del *Color Layout* (4.3.1.2.3), a la que se elimina el paso de la reducción de la imagen a otra de tamaño 8x8.

Al diagrama de bloques anterior bastaría con añadirle un bloque que especificara "Aplicación del algoritmo *Color Layout*", antes del de "Almacenar los valores en la *tabla*".

5 Integración y resultados

5.1 Introducción

En este capítulo se expone la descripción de la aplicación "Repositorio de imágenes", en la que se incluyen los algoritmos implementados en el capítulo de Desarrollo.

5.2 Aplicación Repositorio de imágenes

5.2.1 Integración

Tras el estudio e implementación de los diversos algoritmos de detección de imágenes ruidosas y duplicadas, una de las mejores maneras de demostrar su funcionamiento y utilidad era desarrollar una aplicación en la que fuera necesario su empleo.

La ordenación de un repositorio de imágenes digital (más comúnmente conocido como álbum) engloba todos los análisis estudiados (calidad y duplicados).

En esta sección se describen los algoritmos de funcionamiento de la aplicación mediante diagramas de módulos. Para una información más pormenorizada, consúltese el anexo A.1, donde se especifican los detalles de programación.

5.2.2 Motivación y objetivo

La inmensa capacidad de las memorias de las cámaras digitales hace que a menudo se almacenen cientos de fotos en sus tarjetas y que, cuando por fin se produzca el momento de la descarga al disco duro, haya una inmensa mezcla de imágenes de diferentes eventos. Entre estas imágenes seguro que hay varias repeticiones de un mismo momento (imágenes duplicadas), fotos tomadas sin querer (imágenes homogéneas), algunos retratos a contraluz o con mucho flash (imágenes subexpuestas o sobreexpuestas) o fotografías movidas o desenfocadas (imágenes borrosas).

Parece justificable la necesidad de una aplicación repositorio de imágenes que clasifique y ordene una carpeta de fotografías, siendo éste el objetivo final de todos los algoritmos desarrollados.

En general, los principales puntos que tendrá que cumplir el programa son:

- Leer el *path* de la carpeta que establezca el usuario.
- Organizar las imágenes en subcarpetas según la fecha en que fueron tomadas y crear una carpeta "*Trash*" en cada una de ellas.
- Calcular la calidad de las imágenes de cada carpeta y enviar a las que no superen el umbral a la carpeta "*Trash*".
- Cambiar el nombre de las imágenes que se consideren duplicadas, organizándolas por calidad.

De esta forma conseguimos una distribución de las imágenes que permite al usuario deshacerse rápidamente de las fotografías que no le interesen y mantener ordenadas las que desee.

5.2.3 Módulos

En este apartado se describen, mediante diagramas de bloque, los dos módulos en los que se divide la aplicación.

5.2.3.1 Módulo 1: Gestión de archivos según calidad

El módulo 1 del programa organiza la totalidad de las imágenes de la ruta proporcionada por el usuario en carpetas cuyo nombre es la fecha de toma de las fotografías. En el interior de cada una de estas carpetas se crea otro directorio con el fin de hacer las veces de "Papelera de reciclaje". Las imágenes que no superen el umbral de calidad son enviadas a estas carpetas.

El diagrama de flechas de la figura 5-1 ilustra sobre el funcionamiento de la aplicación durante el módulo 1.

El usuario introduce la ruta donde se encuentran las imágenes que desea clasificar, donde se crea un archivo (*names.txt*) con los nombres de las imágenes JPEG presentes en la carpeta. A continuación, se averigua la fecha de toma de cada una de las fotografías, creándose una carpeta cada vez que se encuentre una fecha distinta. Estos directorios llevarán el nombre de la fecha en cuestión, y en su interior albergarán la carpeta "*Trash*". Cada imagen procesada se trata con los algoritmos de calidad explicados en la sección anterior (4.3.1.1 o 4.3.2.1) y son trasladadas, según si su factor de calidad supera un umbral, a la carpeta de fecha correspondiente o a la carpeta "*Trash*" que se encuentra dentro de ella.

El valor de calidad de cada una de las imágenes, junto con su nombre, se almacenan para su posterior consulta.

5.2.3.1 Módulo 2: Gestión de archivos según duplicados

Una vez las imágenes están distribuidas en sus correspondientes carpetas, queda por analizar si existen duplicados entre ellas. Esta detección se realiza entre imágenes que hayan sido tomadas el mismo día (pues es donde más probabilidades hay de que se haya repetido la misma escena, y porque dos imágenes iguales de dos días diferentes pueden ser de interés del usuario, si se ha tomado la molestia de hacerlas), y sólo si son de buena calidad (se supone que las que se encuentren en "*Trash*" serán eliminadas por el usuario).

Para marcar las imágenes duplicadas, el programa cambia sus nombres por el de la imagen de entre ellas de mayor calidad, y añade un número que indica el orden que siguen según este parámetro.

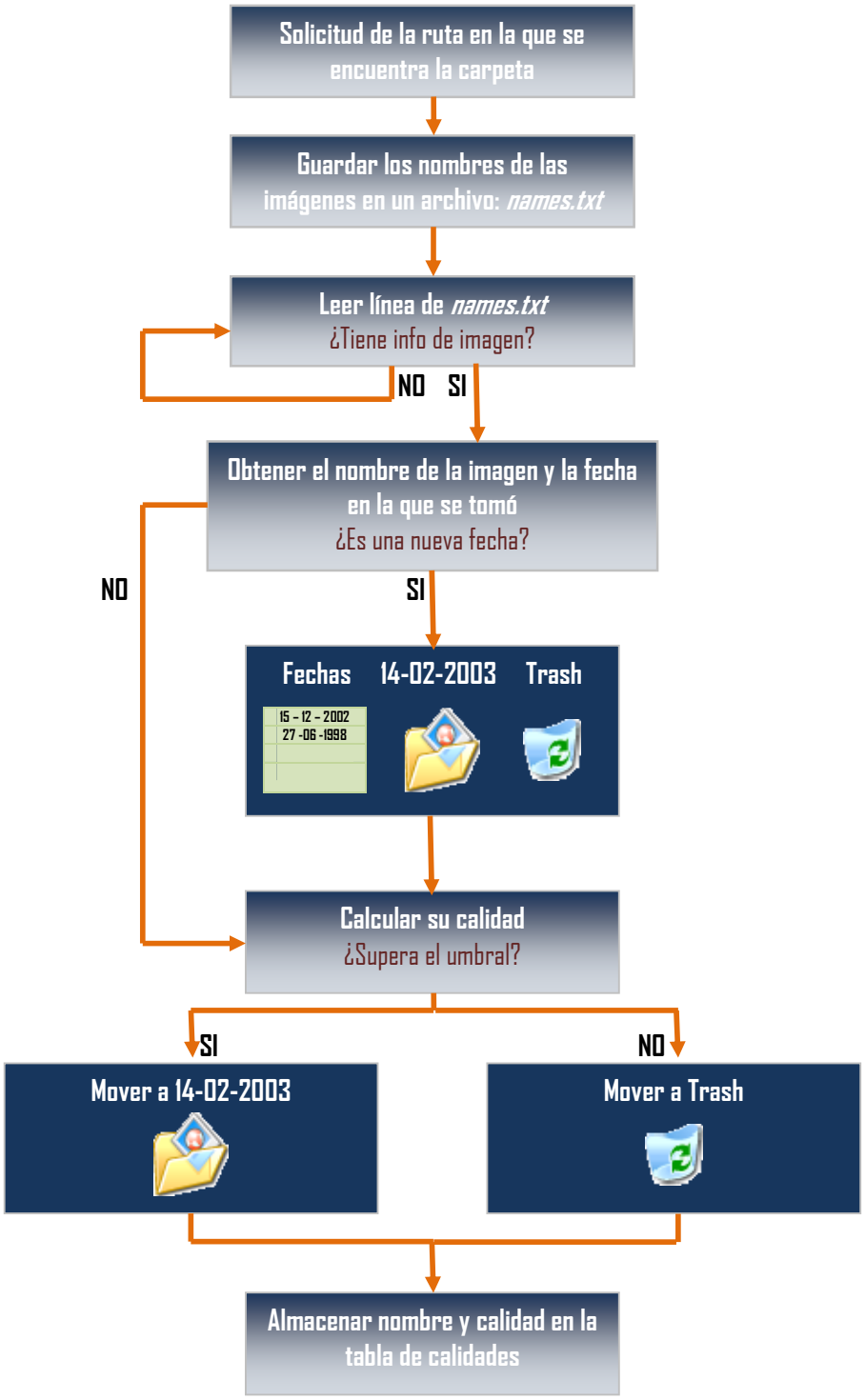


FIGURA 5-1: Diagrama de flechas: módulo 1

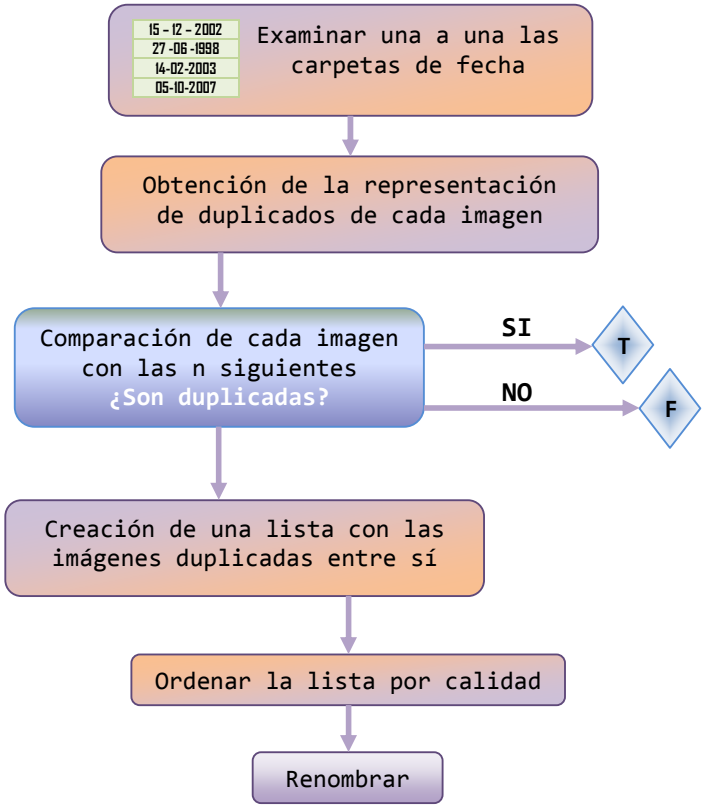


FIGURA 5-2: Diagrama de bloques del módulo 2

Cuando se accede al interior de una carpeta, el primer objetivo es calcular la representación de duplicados de cada imagen según el método propuesto. Estas representaciones se almacenan para, a continuación, decidir mediante la distancia correspondiente si dos imágenes son duplicadas entre sí. El algoritmo compara cada imagen con las *n* siguientes, marcando para cada una cuáles se consideran duplicadas suyas (T o F).

Con el resultado del análisis se construyen listas de imágenes duplicadas, es decir, para una imagen concreta, se construye una lista con los identificadores de aquellas que hayan dado T al realizar la comparación. A estas listas se han de añadir las imágenes duplicadas de las que ya se encontraban en la lista. Este caso se da cuando una imagen A es duplicada de otra imagen B, pero no de C. Sin embargo, cuando se examinan los duplicados de B, C se determina duplicada de ésta. Es aconsejable considerar que A, B y C son duplicadas entre sí, razón por la que se añade C a la lista.

Una vez se han construido todas las listas, los identificadores de cada una de ellas se ordenan según la calidad de las imágenes que las integran. A continuación, se renombran las imágenes de cada lista con el nombre de la mayor calidad (para indicar que son, en realidad, duplicadas suyas), numerándolas siguiendo la organización establecida en el paso anterior.

Al terminar, la carpeta original contiene una serie de carpetas cuyos nombres son fechas, y dentro de cada una de ellas se encuentran las imágenes tomadas ese día nombradas según duplicados y su calidad, y una carpeta *"Trash"* donde se hallan las que se consideran inservibles.

FIGURA 5-3: Aspecto final de la carpeta original

5.2.4 Interfaz gráfica

Las interfaces gráficas surgen de la necesidad de hacer los ordenadores más accesibles para el manejo de los usuarios comunes. Con la idea de simplificar el uso de la aplicación repositorio de imágenes, se desarrolla una interfaz gráfica que haga más amigable la introducción de datos y la vista de resultados.

5.2.4.1 Motivación y objetivo

La idea de la implementación de una interfaz gráfica surge como una ayuda para el establecimiento de umbrales y pesos en la función de calidad a partir del código del repositorio.

La aplicación repositorio de imágenes ordena la carpeta designada por el usuario haciendo uso de los umbrales y pesos que estaban establecidos en el programa. Para analizar los resultados era preciso observar como había quedado el directorio o depurar el programa en busca de valores como la entropía o el factor de *blur*.

La interfaz a desarrollar, por tanto, ha de cumplir con los siguientes objetivos:

- Permitir la introducción de una ruta.
- Consentir la elección de cualquiera de los algoritmos para la clasificación.
- Establecer como propios los umbrales de calidad y duplicados introducidos por el usuario, así como los pesos en la función de calidad.
- Mostrar las imágenes ordenadas por fecha, por calidad y por duplicados, etiquetadas con los valores de calidad, *blur* y entropía que las caracterizan y garantizar la navegación entre ellas.
- Soportar la repetición de los cálculos hasta que sea necesario.
- Tener un adecuado control de errores.

5.2.4.2 Interfaz a nivel funcional

En este apartado se ilustra brevemente el funcionamiento de la interfaz a nivel de usuario. Para más información sobre la implementación de la interfaz, se han desarrollado uno a uno los módulos en los que se divide la aplicación en el Manual del programador (anexo A.2).

Al comienzo del programa aparece una ventana en la que únicamente se puede pulsar el botón *Nuevo Path*, que abre a su vez un diálogo en el que se solicita la ruta en la que se encuentran las imágenes que se desean examinar (figura 5-4).

Inmediatamente después de aceptar, se muestran las distintas opciones que se ofrecen a elegir con respecto a los algoritmos de calidad y duplicados. Estas ventanas son diferentes si la aplicación se centra en algoritmos a nivel de píxel o de bloque (figura 5-5).

En este punto el usuario ha de esperar unos segundos para que se compruebe que existen imágenes en la ruta que ha sugerido y para que se trabaje con ellas. A continuación, el botón *Recalcular* aparece habilitado. Las opciones serán, por tanto, introducir una nueva

ruta por medio de *Nuevo Path* o establecer pesos y umbrales en la ventana "Recalcular" (figura 5-6).

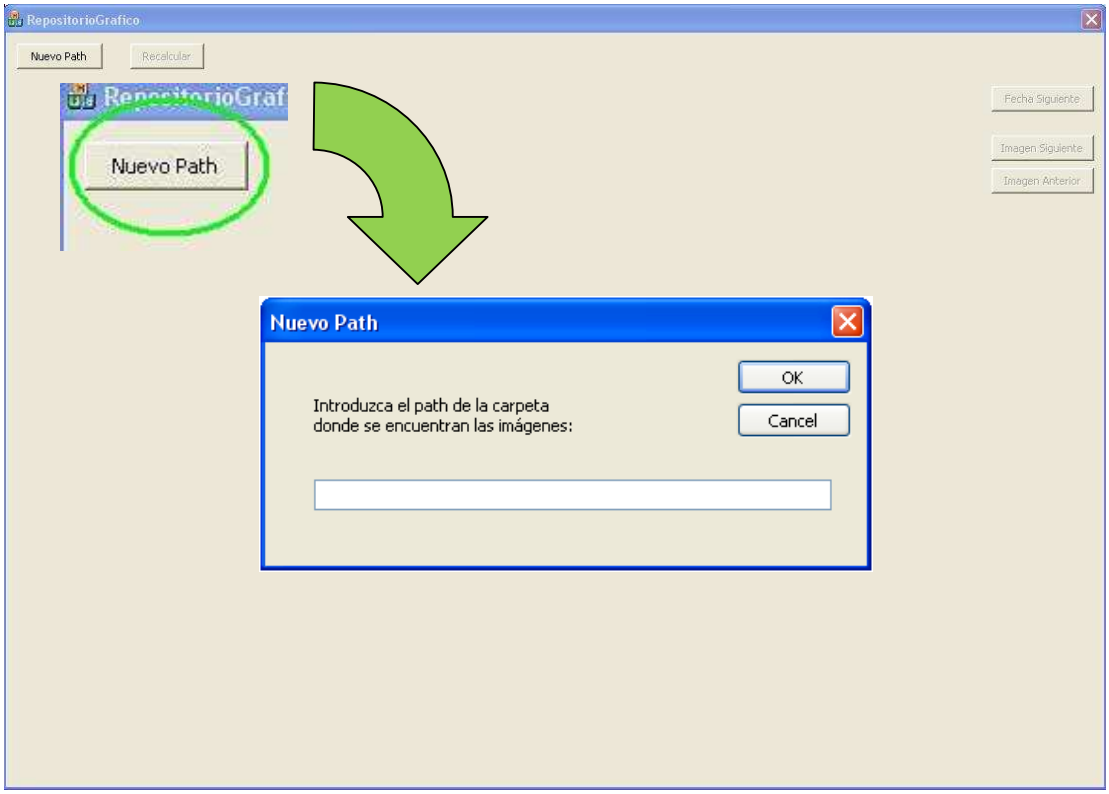


FIGURA 5-4: Ventana principal de la interfaz gráfica

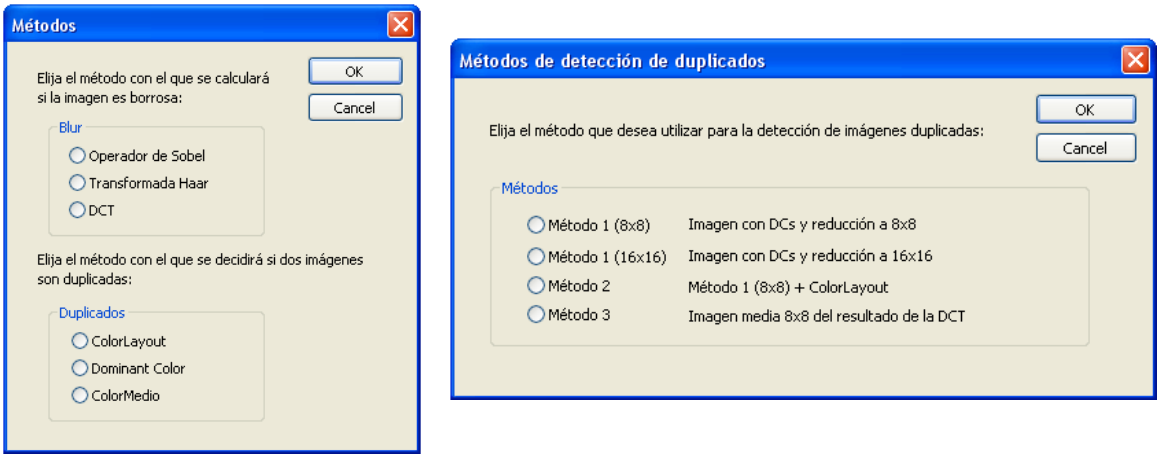
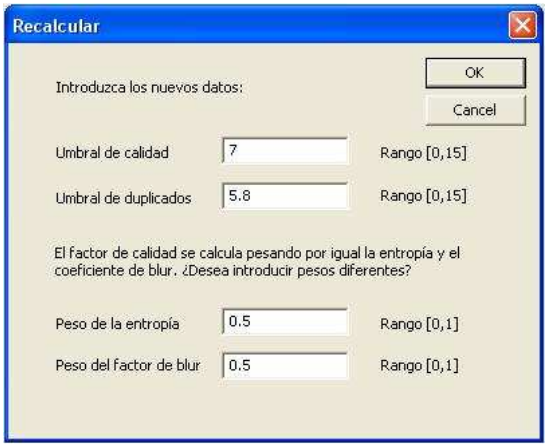
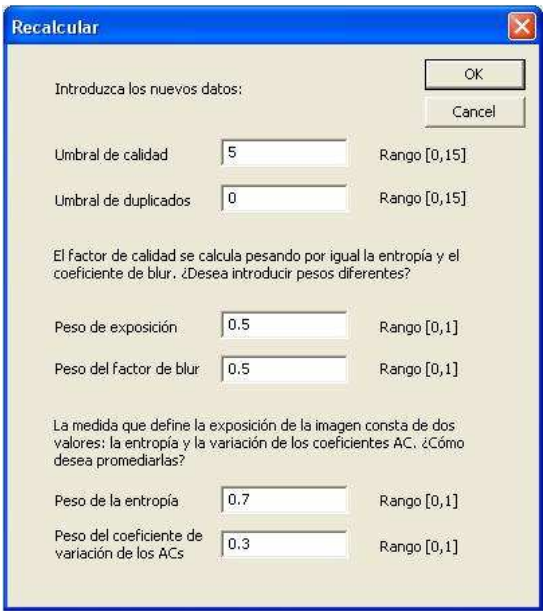


FIGURA 5-5: Ventanas de elección de algoritmos para las interfaces que trabajan a nivel de píxel y a nivel de bloque

Los umbrales elegidos, así como los pesos seleccionados, se imprimen en la parte superior de la ventana principal, junto con la fecha en la que fueron tomadas las imágenes que van a tratarse con esa información. Las miniaturas de estas imágenes se muestran en un mosaico junto a sus representaciones de duplicados y los valores de calidad, coeficiente de *blur*, exposición, etc. que las caracterizan.



Ventana "Recalcular" de la aplicación a nivel de píxel



Ventana "Recalcular" de la aplicación a nivel de bloque

FIGURA 5-6: Ventanas "Recalcular"

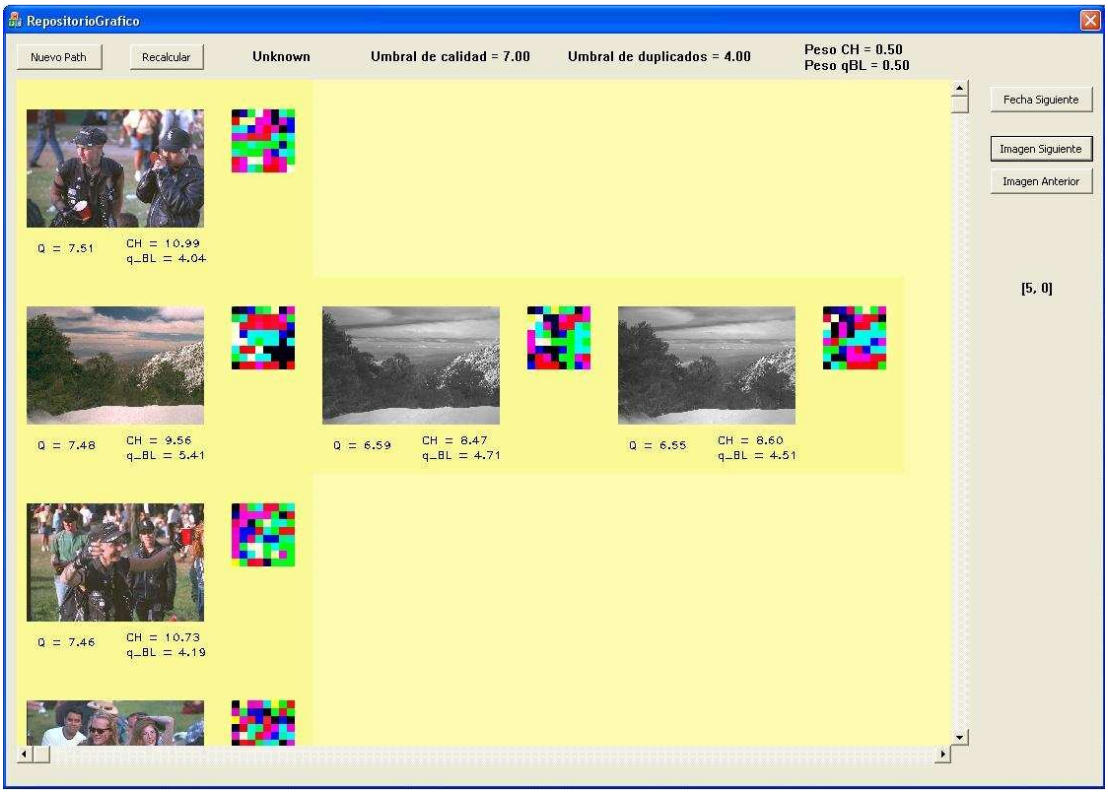


FIGURA 5-7: Ventana principal con los botones *Imagen Anterior* e *Imagen Siguiente* habilitados y con el rótulo de coordenadas que indica la posición de la imagen mostrada dentro de la imagen global

Estas miniaturas se ordenan por filas según duplicados, siendo la primera por la izquierda la de mayor calidad. Si la imagen no supera el umbral de calidad impuesto por el usuario, su miniatura se muestra en blanco y negro.

Si la imagen global (formada por el mosaico de miniaturas) es demasiado grande como para ser manejada por el programa, se divide en porciones y se habilitan los botones *Imagen Siguiente* e *Imagen Anterior* para permitir la navegación entre las piezas. Las coordenadas de la pieza que se está visualizando en ese momento aparecen en el lateral derecho de la ventana principal.

En este punto se pueden comprobar los resultados obtenidos en la ordenación de las imágenes y, si se desea, introducir nuevos datos mediante el botón *Recalcular*. Si existen imágenes tomadas en una fecha distinta, se puede analizar una nueva carpeta por medio del botón *Fecha Siguiente*.

5.2.5 Resultados

Los resultados de calidad se describen con tres cifras diferentes: aciertos (imágenes que concuerdan con la calidad que se les asignó previamente), falsos aciertos (imágenes que son consideradas de buena calidad por el algoritmo, cuando estaban clasificadas como de mala calidad) y falsos negativos (imágenes que se suponían de buena calidad y que son calificadas por el algoritmo como malas).

Los resultados de duplicados se detallan también con tres valores: aciertos (parejas que se consideran duplicadas y que se clasifican de esta forma), falsos aciertos (imágenes que se agrupan con otras cuando no son duplicadas) y falsos negativos (imágenes que se califican duplicadas de otras pero que el algoritmo no clasifica como tal).

5.2.5.1 Detalles sobre el conjunto de imágenes de prueba

El conjunto de prueba se compone de 273 imágenes provenientes de dos fuentes diferentes: el set S1 (CD6 (91 imágenes), CD7 (62 imágenes), CD8 (90 imágenes)) y el set S4 (30 fotografías) de la base de datos de MPEG-7.

El conjunto de fotografías se ordenó por calidad siguiendo un criterio subjetivo, llegando a las cifras señaladas en la tabla que a continuación se expone.

	Nº de imágenes cuya calidad supera el umbral	Nº imágenes cuya calidad no supera el umbral
CD6	83	8
CD7	54	8
CD8	88	2
S4	21	9
TOTAL	**282**	**51**

TABLA 5-1: Clasificación según calidad de las imágenes de prueba

El contenido de las fotografías de prueba difiere según el catálogo al que pertenezcan. Las imágenes del CD6 inmortalizan paisajes de campo, edificios y a varias personas en una celebración nocturna. En el CD7 se muestran las fotografías de una fiesta popular, y en el CD8 paisajes o fotografías de campo donde no se encuentran personas. El set S4 se concentra en imágenes en primer plano de seis personas.

De la misma forma, en la tabla siguiente se detallan el número de parejas consideradas por set.

	Nº de parejas
CD6	14
CD7	9
CD8	16
S4	8
TOTAL	**48**

TABLA 5-2: Número de parejas en las imágenes de prueba

5.2.5.2 Calidad

La tabla siguiente detalla los resultados en cuanto a calidad obtenidos por los cuatro algoritmos desarrollados, a saber: a nivel de píxel, transformada de Haar y entropía, operador de Sobel y entropía, DCT y entropía y, a nivel de bloque, entropía, DCT y coeficientes AC.

	Algoritmo	Set	Aciertos (%)	Falsos positivos (%)	Falsos negativos (%)
Nivel de píxel	Entropía + Haar	CD6	83.52	3.30	13.18
		CD7	74.19	4.84	20.97
		CD8	60	0	40
		S4	90	0	10
	Entropía + Operador de Sobel	CD6	86.81	3.30	9.89
		CD7	69.35	1.61	29.04
		CD8	81.11	0	18.89
		S4	46.67	0	53.33
	Entropía + DCT	CD6	81.32	4.40	14.28
		CD7	72.58	1.61	25.81
		CD8	65.56	0	34.44
		S4	56.67	0	43.33
Nivel de bloque	Entropía + coef AC + DCT	CD6	87.91	3.30	8.79
		CD7	88.71	1.61	9.68
		CD8	84.44	0	15.56
		S4	56.67	3.33	40

TABLA 5-3: Resultados de calidad

De esta tabla se deduce que, a la hora de catalogar primeros planos (generalmente con fondo uniforme, sin problemas de iluminación y con variaciones muy sutiles), el algoritmo de la transformada de Haar trabaja mejor que los demás. Sin embargo, para clasificar paisajes y panorámicas el que mejor funciona es el que opera a nivel de bloque, al igual que con fotografías de contenido general.

Los porcentajes de resultados generales se especifican en la tabla siguiente, donde se justifica que la opción más óptima es la que trabaja a nivel de bloque:

Algoritmo	Aciertos (%)	Falsos positivos (%)	Falsos negativos (%)
Haar	74.36	2.20	23.44
Sobel	76.56	1.47	21.97
DCT	71.43	1.83	26.74
AC + DCT	81.32	1.83	16.85

TABLA 5-4: Resultados generales de calidad

Si se comparan estos resultados con los obtenidos con el conjunto de imágenes de entrenamiento, se comprueba que los algoritmos implementados con los umbrales escogidos generalizan bien.

5.2.5.3 Detección de duplicados

Los resultados sobre detección de duplicados se presentan en la tabla a continuación, detallados según algoritmo y set de imágenes. Así como el valor de aciertos y el de falsos negativos se da en cuanto al total de parejas (son complementarios), el valor de falsos positivos se asigna en cuanto al número total de imágenes que no deberían pertenecer a ninguna pareja.

A la vista de los resultados (tabla 5-5) se puede determinar que el conjunto más sencillo de emparejar correctamente es el CD8, seguido por los retratos en primer plano (S4). El que parece ser más complicado de clasificar es el CD7, pues ningún algoritmo asimila más del 50% de las parejas.

En cuanto a los resultados generales (tabla 5-6), los algoritmos que más aciertos tienen son *Dominant Color* y los dos que utilizan *Color Layout*. Este hecho se explica debido a que *Dominant Color* realiza una búsqueda muy exhaustiva (en comparación con el Color Medio, por ejemplo) de los colores que representan una región, consiguiendo una mejor imagen representativa. *Color Layout*, sin embargo, debe su gran acierto a que, además de obtener una imagen de colores representativos, define cómo son las variaciones entre los colores de esa imagen gracias a la DCT, lo que le permite determinar mejor las igualdades entre imágenes cuyo contenido esté ligeramente desplazado.

	Algoritmo	Set	Aciertos (%)	Falsos positivos (%)	Falsos negativos (%)
Nivel de píxel	Dominant Color	CD6	57.14	37.66	42.86
		CD7	22.22	1.89	77.78
		CD8	100	9.46	0
		S4	75	54.54	25
	Color Layout	CD6	50	12.99	50
		CD7	44.44	3.77	55.56
		CD8	87.50	14.86	12.50
		S4	87.50	13.64	12.50
	Color Medio	CD6	42.86	27.27	57.14
		CD7	22.22	0	77.78
		CD8	93.75	9.46	6.25
		S4	62.50	31.82	37.50
Nivel de bloque	Método 1 (8x8)	CD6	57.14	41.56	42.87
		CD7	22.22	0	77.78
		CD8	81.25	20.27	18.75
		S4	62.50	27.27	37.50
	Método 1 (16x16)	CD6	50	42.86	50
		CD7	22.22	0	77.78
		CD8	87.50	28.38	12.50
		S4	62.50	27.27	37.50
	Método 2	CD6	50	16.88	50
		CD7	33.33	5.66	66.67
		CD8	93.75	10.81	6.25
		S4	75	9.09	25

TABLA 5-5: Resultados de duplicados

Los resultados generales aparecen en la tabla siguiente:

	Algoritmo	Aciertos (%)	Falsos positivos (%)	Falsos negativos (%)
Píxel	DC	68.09	21.68	31.91
	CL	68.09	11.05	31.91
	CM	59.57	15.49	40.43
Bloque	Mét 1 (8x8)	59.57	23.45	40.43
	Mét 1 (16x16)	59.57	26.55	40.43
	Mét 2	65.96	11.50	34.04

TABLA 5-6: Resultados generales de duplicados

Un resultado sorprendente es que el Método 1 con tamaño de imagen representativa 16x16 tiene el mismo porcentaje de aciertos que el Método 1 con tamaño de imagen representación 8x8. El Método 1 (16x16) consigue una imagen mucho más precisa, acertando mejor entre imágenes con pocas variaciones entre ellas. Sin embargo, el Método 1 (8x8), al ser menos preciso con la posición de los colores, acierta más cuando las imágenes contienen el mismo elemento, pero desplazado.

5.2.5.4 Tiempos de ejecución

Medir el tiempo que tarda en ejecutarse la aplicación repositorio de imágenes con los distintos algoritmos ayuda a decidir sobre la supremacía de algunos sobre otros. En la tabla siguiente se especifican los tiempos que tarda la aplicación en ordenar el conjunto de imágenes de prueba.

	Calidad	Duplicados	Tiempo (s)
Nivel de Píxel	Entropía + Haar	DC	1041.030
		CL	99.515
		CM	153.203
	Entropía + Sobel	DC	1403.550
		CL	140.203
		CM	194.610
	Entropía + DCT	DC	1270.170
		CL	88.970
		CM	141.015
Bloque	Entropía + AC + DCT	Mét 1 (8x8)	154.985
		Mét 1 (16x16)	158.625
		Mét 2	152.031

TABLA 5-7: Tiempos de ejecución

Tiempos de ejecución en un ordenador con procesador Intel Core 2 Duo a 2.40 GHz y con 3.25 GB de RAM.

Tras analizar los resultados en cuanto a aciertos y el tiempo de ejecución de los algoritmos, se puede determinar que el método de detección de duplicados que funciona mejor a nivel de píxel es el *Color Layout*, puesto que obtiene el mayor número de aciertos, el menor número de falsos positivos y se ejecuta en el menor tiempo. A nivel de bloque, el Método 2 es el más rápido y el que más aciertos ofrece.

En cuanto a calidad, es difícil decidir si hay un algoritmo que supere a los otros, puesto que parece que el número de aciertos va en consonancia con los tiempos de obtención de los resultados.

Es necesario recordar que, para implementar los algoritmos a nivel de bloque, se simuló la obtención de los bloques DCT decodificando la imagen por completo y aplicando a los píxeles obtenidos la DCT, debido a que las soluciones encontradas al respecto tardaban

más que este sistema. Es de suponer que los tiempos ofrecidos a nivel de bloque en esta tabla disminuirían si se contase de forma inmediata con los bloques DCT, lo que decantaría la balanza a favor de estos algoritmos.

Llama la atención que el Método 2, que consiste en la aplicación del Método 1 y luego el algoritmo *Color Layout*, tarde menos que el Método 1 en sí mismo. Esto es debido a que, cuando se calcula la distancia entre dos representaciones, con la distancia euclídea es necesario convertir los valores RGB al espacio CIE Luv, paso que no es necesario cuando se aplica la distancia entre dos *Color Layouts*.

5.3 Aplicación a secuencias de vídeo

La "Aplicación a secuencias de vídeo" se divide en dos módulos: el que lee los archivos .dct, .ppm y la base de datos de claquetas, y el que trabaja con esta información para extraer datos sobre el *frame*. Con vistas a integrar esta aplicación de vídeo en el sistema IVOnLA [43], el segundo módulo recibe la información de la misma manera que si formara parte de este programa, y devuelve los datos de forma que IVOnLA pueda aprovecharlos.

Con respecto a esta devolución de información, hace falta hacer una aclaración. IVOnLA, como sistema OnLine, no puede esperar a los resultados de aplicar el algoritmo de "cámara grabando sola", ya que se necesita comprobar la duplicidad entre cierto número de *frames*, lo que implica un retraso. Para conseguir integrar este algoritmo es necesario que IVOnLA introduzca como entrada, además de los valores de los coeficientes DCT, las imágenes .ppm, las representaciones de duplicados de la base de datos o el método elegido para calcular los duplicados, un *array* de estructuras de salida. Estas estructuras de salida están llamadas a contener datos sobre calidad global, exposición, coeficiente de *blur*, claqueta con la que se identifica el *frame*, etc., pero también ha de añadirse una variable que avise de si se ha producido el fenómeno de "cámara grabando sola". Hasta que el *array* de estructuras no esté lleno, IVOnLA no puede extraer datos sobre ninguno de los *frames*, es decir, que para que la aplicación de vídeo sea integrable en IVOnLA, es preciso suministrar al programa el retardo que supone rellenar de información el *array* de estructuras de salida.

5.3.1 Integración

El siguiente reto consiste en desarrollar una aplicación que trabaje a partir de secuencias de vídeo que consiga extraer, mediante los algoritmos implementados, distintas características de éste. Un vídeo es una sucesión de imágenes (cuadros) que se muestran normalmente a una frecuencia de 25 por segundo. Ya que un vídeo es divisible en imágenes, es posible aplicar los algoritmos desarrollados con el fin de detectar secuencias ruidosas o de poco interés para el usuario.

5.3.1.1 Motivación y objetivos

Las secuencias de vídeo sin editar suelen contener escenas repetidas, tomas que no interesa incluir, errores de grabación… Parte del trabajo de selección y eliminación de

estas secuencias no deseadas puede realizarse gracias al uso de algunos de los algoritmos implementados para detectar imágenes ruidosas y duplicadas.

Los objetivos que debe cumplir un programa de estas características son los siguientes:

- Procesar la información referente a un vídeo MPEG (anexo E) conseguida mediante IVOnLA [43].
- Detectar los *frames* I.
- Permitir la elección de los algoritmos que se desee aplicar para la detección de duplicados.
- Revelar, mediante la aplicación de algoritmos de detección de duplicados, si la cámara se ha quedado grabando sola.
- Ser capaz de aplicar la detección de duplicados, no sólo sobre el vídeo propiamente, sino frente a una base de datos de patrones a filtrar. La idea es que si existen imágenes tipo que se quieren eliminar (cartas de ajuste, cortinillas de cabecera de programas, créditos, claquetas) la base de datos contendrá imágenes tipo con esos patrones que serán los que se usen para buscar (y posteriormente eliminar si así se decide) como duplicados en el vídeo.
- Mostrar los resultados de cada *frame* en cuanto a calidad (coeficiente de blur, exposición).
- Avisar si se detecta alguna secuencia no deseada, ya sea una toma en la que la cámara esté grabando sola o una en la que aparezca una imagen de la base de datos.
- Tener un adecuado control de errores: lectura errónea de un fichero, valores anormales en las medidas de calidad o duplicados, elección de un método no disponible como opción…
- Generar en un modo *on-line* con mínimo retardo una curva (multievaluada) con los distintos indicadores obtenidos que podrá ser utilizada posteriormente para filtrar trozos del vídeo o incluso hacer una primera edición automática del mismo.

A continuación se describen los módulos que conforman el programa.

5.3.1.2 Módulos

El programa contiene dos partes principales: la captura de datos y el examen en cuanto a calidad y duplicados de cada *frame* I, que tienen su razón de ser debido a la deseable integración de esta aplicación en IVOnLA [43]. El programa implementado parte de los resultados que devuelve IVOnLA (a saber: coeficientes DCT e imágenes DC de cada *frame*), pero no forma parte aún de éste. La información necesaria sobre los *frames* se encuentra tanto en archivos .dct como en imágenes .ppm, que necesita ser previamente procesada (labor del módulo de captura de datos). De esta manera, el módulo de calidad y duplicados recibe como argumentos los mismos datos que en su momento le brindará IVOnLA, y retorna la información de forma que sea inteligible y útil para IVOnLA, haciendo así posible la integración.

5.3.1.2.1 Captura de datos

El programa, aunque dependiente de los resultados de IVOnLA [43], puede ejecutarse a la vez o después que éste. IVOnLA se encarga de procesar el vídeo, almacenando

información de cada *frame* en una carpeta auxiliar a mucha más velocidad que lo que tarda la aplicación en tratar esta información.

La información relevante para el programa se concentra en archivos .dct e imágenes .ppm. Los ficheros .dct guardan detalles sobre el tipo de *frame* (I, P o B), ancho y alto del *frame* en bloques DCT y valor de los bloques DCT para cada una de las componentes Y, U y V (anexo C sobre espacios de color y anexo E para referencias sobre MPEG). Las imágenes .ppm contienen miniaturas de los *frames* construidas a partir de los coeficientes DC.

El primer propósito del programa consiste en permitir al usuario elegir entre los distintos métodos para duplicados. Para ello se muestra un menú por pantalla que se repite hasta que se haya seleccionado una opción correcta. La elección del método es muy importante, ya que determina por completo los pasos siguientes. Cuando el módulo de calidad y duplicados se encuentre integrado en IVOnLA, será necesario que esta operación se realice previamente.

FIGURA 5-8: Menú de selección del método para detectar duplicados

Una de las posibles aplicaciones del programa radica en detectar cuándo un *frame* contiene una claqueta, una carta de ajuste, etc. para su filtrado (ejemplo dentro de las evaluaciones *TRECVID BBC Rushes Summarization* [25]). Para ello se hace uso de los métodos de duplicados comparando cada uno de los *frames* I con una base de datos de imágenes tipo (de esta forma podría extenderse sin problema a otro contenido) tal y como se explicó en el apartado 3.4.2.2. Esta base de datos consiste en un conjunto de archivos binarios que recopilan los valores RGB o coeficientes DCT (según el método) de cada una de las imágenes que la conforman. Como la lectura de disco es más lenta que el acceso a memoria, y las comparaciones con la base de datos se realizan para cada *frame*, en este punto del programa se cargan las representaciones de las claquetas (atendiendo al método elegido) en una tabla almacenada en memoria a la que se accederá en cada iteración.

Los ficheros .dct se almacenan con el nombre del vídeo y las palabras "_DCT_x.dct", siendo x el número del *frame* escrito con 4 dígitos (aunque este valor es modificable si se

cambia a su vez en IVOnLA). Cuando se lee uno de estos archivos, la primera información que aparece es el tipo de *frame* de que se trata (I, P o B). Si se comprueba que es I, se lee el resto del fichero almacenando los datos (alto y ancho del *frame* en bloques DCT, valor de los coeficientes DCT) en una estructura. A continuación se abre la imagen .ppm correspondiente a ese *frame* para guardarla al igual. Esta estructura contiene campos como el método de duplicados elegido antes, la tabla de representaciones de la base de datos, el número de *frame* I de que se trata y el tipo de *frame*, aparte de los valores citados anteriormente. Es uno de los argumentos de entrada del módulo de calidad y duplicados, que tendrá que encargarse de rellenar IVOnLA cuando se encuentre integrado.

La salida del módulo de calidad y duplicados consiste en una lista con un número determinado de estructuras. Este número es exactamente el correspondiente a la cantidad de *frames* I que han de ser examinados para decretar si la secuencia corresponde a una grabación involuntaria (apartado 3.4.2.1), y puede ser modificado según las características del vídeo. Cada una de las estructuras contiene información sobre *frames* I consecutivos, como la calidad global que se le adjudica, su exposición, su coeficiente de *blur*, el número de imagen de la base de datos al que se parece, su representación según duplicados, si se consideró duplicado del *frame* I anterior y el número de *frame* que es según el vídeo.

Con estas dos estructuras se accede al módulo de calidad y duplicados.

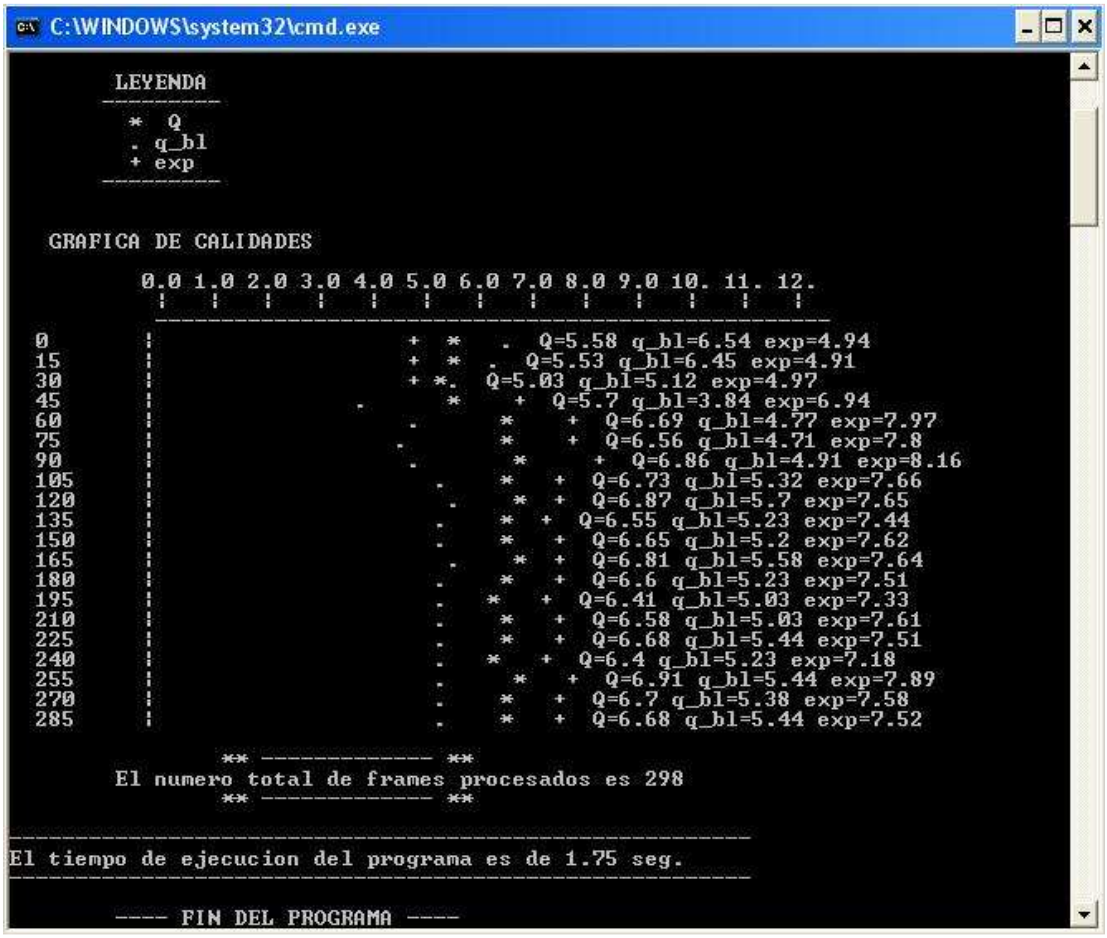


FIGURA 5-9: Gráfica de calidades de la secuencia "table.m1v"

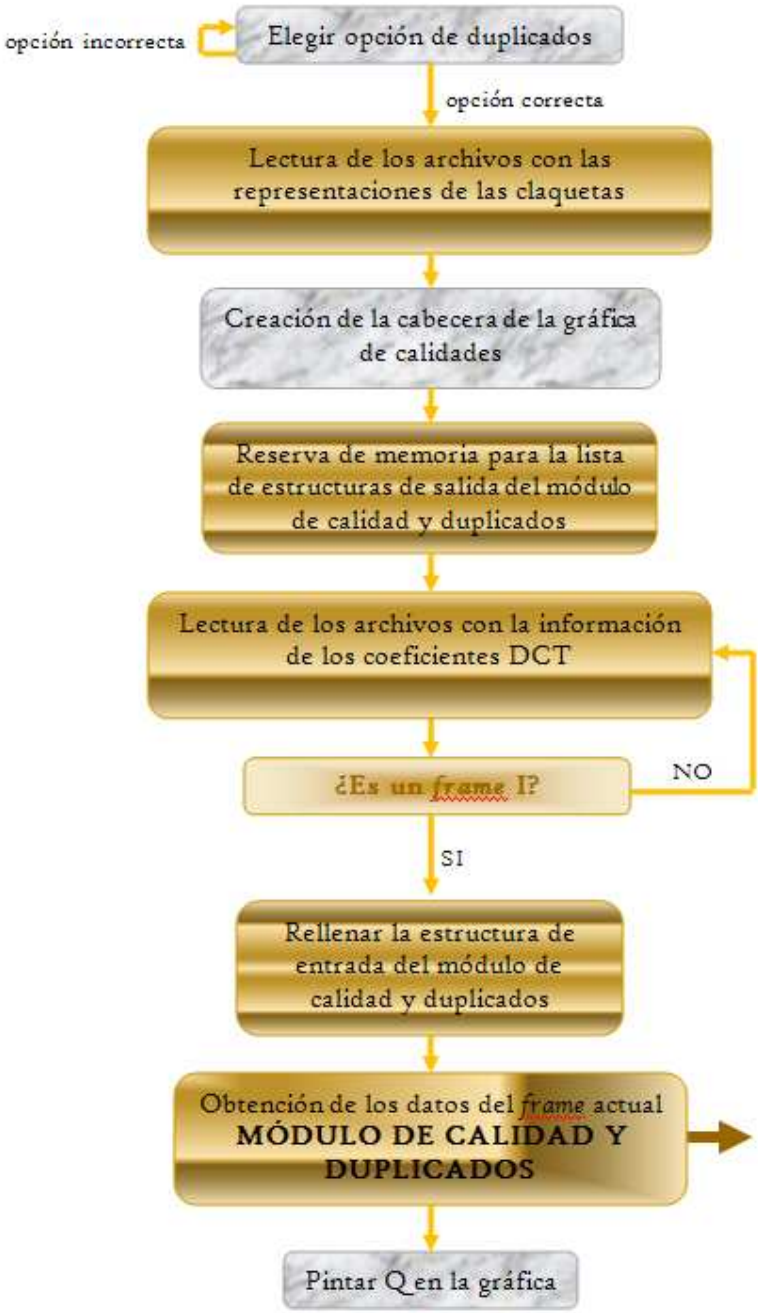


FIGURA 5-10: Diagrama de bloques del módulo de captura de datos

Una vez se ha terminado de analizar un *frame*, se muestran por pantalla su medida de exposición, su coeficiente de *blur* y su calidad global en la gráfica de calidades. Esta gráfica presenta una curva con los diferentes valores obtenidos a medida que avanzan los exámenes sobre los *frames*. De esta forma se adquiere una idea de la categoría que tiene la secuencia procesada (figura 5-9).

En la figura 5-10 se imprime el diagrama de bloques del módulo de captura de datos.

5.3.1.2.2 Módulo de calidad y duplicados

El módulo de calidad y duplicados es el que se encarga de dar valores a los campos de la estructura de salida. Antes de realizar los análisis de calidad, de duplicados, etc., modifica los valores de las matrices DCT, que se leen de IVOnLA desordenadas (los bloques DCT mantienen el orden en zigzag introducido por el codificador) y sin escalar.

Este módulo es plenamente integrable en IVOnLA [43].

5.3.1.2.2.1 Calidad

Los algoritmos de calidad que se implementan en la aplicación de vídeo son los que trabajan a nivel de bloque, lógicamente. A saber: DCT para calcular el coeficiente de *blur*, entropía y coeficientes AC para obtener la exposición (apartado 4.3.2.1).

Para el coeficiente de *blur* sólo es necesario hacer uso de la componente Y, ya que el factor de *blur* se obtiene de la imagen en escala de grises. Para obtener la exposición, sin embargo, se necesitan las 3 componentes YUV (anexo C, espacios de color) y la imagen .ppm. La entropía se calcula con la imagen creada a partir de los coeficientes DC. La cuantía de variación de los coeficientes AC, no obstante, ha de obtenerse mediante el procesamiento de las 3 matrices DCT. Los valores de entropía y coeficientes AC se pesan con los factores 0.7 y 0.3 respectivamente.

Finalmente, las medidas de *blur* y exposición se suman ponderadas por los pesos 0.4 y 0.6.

El coeficiente de *blur*, la exposición y la calidad global se almacenan en la estructura de salida.

5.3.1.2.2.2 Duplicados

La detección de duplicados se realiza en dos pasos, según cuál sea el objetivo. En primer lugar se compara la representación del *frame* actual con la representación del anterior, a fin de determinar si pueden considerarse. En segundo lugar, se compara la representación del *frame* actual con cada una de las representaciones de las imágenes de la base de datos (claquetas, cartas de ajuste…) hasta que la detección sea positiva o no queden más representaciones.

5.3.1.2.2.2.1 Cámara grabando sola

El algoritmo implementado se basa en el método explicado en el apartado 3.4.2.1. En él se determinaba que el número de *frames* I que tenían que dar positivo en la comparación para que la secuencia pudiese considerarse una grabación involuntaria (en una secuencia de vídeo de contenido general) era de 17, lo que correspondía a 10.2 segundos de vídeo. Este número (17) es la cantidad de estructuras de salida que se encuentran almacenadas en la lista que recibe el módulo de calidad y duplicados como argumento. Por supuesto, esta cantidad es modificable según el tipo de vídeo que se esté analizando (no será lo mismo una película de acción que la filmación de un documental, donde puede ser que el presentador permanezca a la espera de que algo ocurra y esta información sea de interés para el espectador).

Para empezar, se calcula la representación de duplicados correspondiente al *frame* actual, y se compara con la del *frame* anterior (almacenada en la lista). Si se consideran duplicados, se activa en la estructura de salida del *frame* actual un *flag* que lo indique.

Antes de retornar los valores de salida, se comprueba si el 100% de los *flags* de la lista de estructuras de salida se encuentran activados. Si es así, se determina que la cámara se ha quedado grabando sola o que no ocurre nada en la escena, mostrando un mensaje por pantalla que informe desde qué *frame* se considera que sucede esto. Si la posterior comparación sigue devolviendo *true*, el mensaje se limita a anunciar que la grabación parece seguir siendo involuntaria o que nada está ocurriendo. En ambos casos se podría decidir filtrar estos trozos a posteriori, si bien la causa original, aunque similar, fuese distinta.

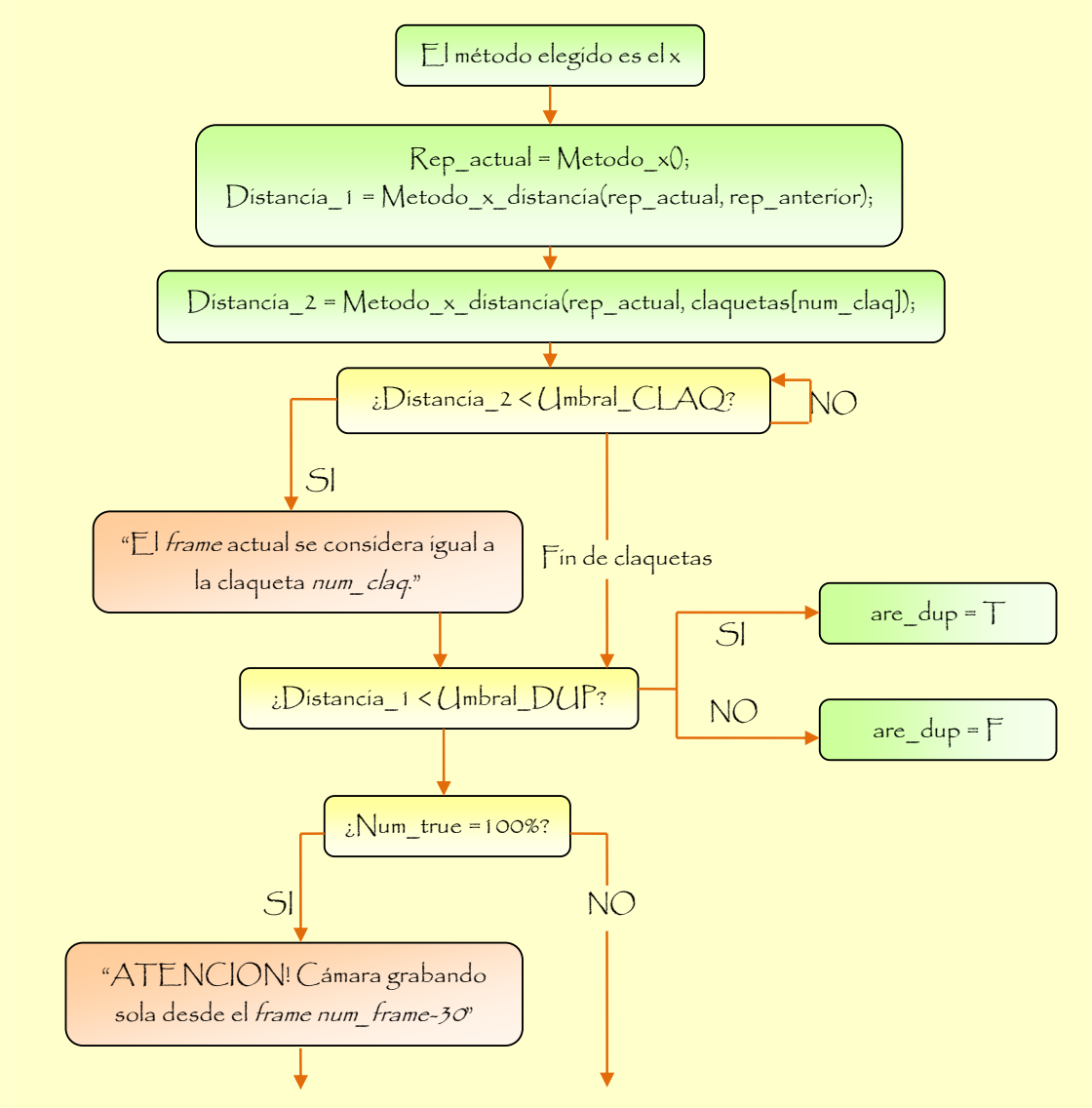


FIGURA 5-11: Diagrama de módulos del análisis de duplicados

Esta comprobación introduce un retardo en la obtención de resultados por parte de IVOnLA, ya que la información resultante sobre si el *frame k* se incluye dentro de una secuencia de grabación involuntaria no están disponibles hasta que se procesa el *frame k+17*. Sin embargo, este retardo no impide que el módulo sea integrable.

5.3.1.2.2.2.1 Detección de claquetas, cartas de ajuste…

Las representaciones de las imágenes que conforman la base de datos se encuentran almacenadas en una tabla que no varía a lo largo del programa. La comparación es muy sencilla: se mide la distancia entre la representación de duplicados del *frame* actual y la de una imagen de la base de datos según el método seleccionado al principio de la aplicación por el usuario. Si la verificación es negativa, se continúa con el proceso hasta que no haya más representaciones en la tabla. Si es positiva, se termina el análisis, mostrándose por pantalla un mensaje que avisa de que el *frame* actual sea posiblemente una claqueta, pues el examen ha dado positivo para la número n. El diagrama de módulos que ilustra este proceso se muestra en la figura 5-11.

5.3.2 Resultados

5.3.2.1 Detalles sobre el conjunto de vídeos de prueba

Debido a que la elección de umbrales es muy dependiente del tipo del vídeo con el que se trabaje, se han elegido de prueba vídeos de contenido general, todos ellos aún sin editar.

Para probar el funcionamiento de los algoritmos con segmentos de cámara fija y *frames* de baja calidad se han utilizado dos vídeos de 36 y 32 minutos de duración. El primero se trata de una grabación de una serie infantil, con mayoría de secuencias en parques, y con 24 segmentos de cámara fija, además de algún que otro plano desenfocado. Un detalle importante es que este vídeo cuenta con varias conversaciones entre personajes que, con un umbral inadecuado, pueden confundirse con grabación involuntaria. El segundo consiste en la filmación de un capítulo de una serie, que se desarrolla en un cine. En este vídeo, la mayoría de planos son oscuros, aunque hay *frames* sobreexpuestos y multitud de cruces de gente del reparto por delante de la cámara (homogeneidad, borrosidad). Ambos vídeos comienzan con las clásicas barras de colores y no contienen ninguna claqueta ni carta de ajuste.

En cuanto a las claquetas, el vídeo de prueba es de una duración de 30 minutos y cuenta con 28 claquetas, aparte de varios cortes en negro y barras de colores al inicio.

5.3.2.2 Calidad

Como ocurría en el caso de las imágenes, los resultados obtenidos en cuanto a calidad son subjetivos. Ambos vídeos de prueba cuentan con cortes en negro (que se producen cuando se apaga la cámara para cambiar de plano o escena) que son reconocidos sin problema por los algoritmos. Las secuencias iniciales con carta de barras de colores también serían eliminadas debido a su uniformidad.

En cuanto al resto de secuencias, los algoritmos discriminan el 100% de planos en los que se cruza una persona u objeto demasiado cerca de la cámara, o aquellos en los que el

fondo no se encuentra claramente definido (planos desenfocados). La detección de secuencias con sobre/sub exposición es más relativa, puesto que depende del tipo de vídeo que se esté analizando. El umbral recomendado en el capítulo de Diseño (apartado 3.4.1) es aceptable para vídeos con iluminación media, pero es necesario que sea modificable con el fin de imponer unas condiciones más relajadas o más exigentes. El vídeo del cine, por ejemplo, contiene multitud de planos mediocremente iluminados que, aún así, deben incluirse en la grabación final. Esta condición permite que también se acepten como buenos otros planos que no son tan deseables.

Resumiendo, el funcionamiento de los algoritmos es muy satisfactorio aunque los resultados dependan, finalmente, del tipo de vídeo con el que se esté trabajando.

5.3.2.3 Detección de secuencias con cámara grabando sola

Los resultados conseguidos descubriendo secuencias de cámara grabando sola han sido prácticamente de un 100% de aciertos tanto con los vídeos de entrenamiento como con los de prueba. La mayor diferencia entre algoritmos radica en el número de falsos positivos que se han producido.

El Método 1 (con imágenes representativas de tamaño 8x8 o 16x16) tan solo comete errores en el caso de que los personajes se encuentren manteniendo una conversación alejados, puesto que las diferencias entre *frames* I en estas secuencias es mínima. La diferencia que se produce entre los diferentes tamaños de imágenes representativas se reduce a los tiempos de ejecución (apartado 5.3.2.5).

El Método 2 se consideraba uno de los más útiles en la detección de imágenes duplicadas debido a que era capaz de emparejar fotografías con el mismo contenido, pero desplazado ligeramente. Esta propiedad del *Color Layout* es un inconveniente en detección de cámara grabando sola, puesto que sigue considerando que la cámara está fija cuando se producen pequeños movimientos tanto del aparato como de lo que esté sucediendo en escena.

5.3.2.4 Detección de claquetas, cartas de ajuste, etc.

El vídeo de prueba utilizado para medir los resultados de funcionamiento de los tres algoritmos contiene 26 cortes: barras de colores al inicio, un tramo en blanco, otro en negro al final, y 23 claquetas.

Al igual que sucedía cuando se realizó el entrenamiento, se da el caso de que hay veces que los algoritmos detectan el corte en blanco o negro que precede a la claqueta, pero no ésta en sí. Estos casos se reflejan bajo el título "Detecciones", donde se incluyen además los aciertos.

	Detecciones	Aciertos	Fallos	Falsos positivos
Método 1 (8x8)	25	15	1	0
Método 1 (16x16)	24	14	2	3
Método 2	21	9	5	6

TABLA 5-8: Resultados de detección de claquetas

Sorprendentemente, el Método 2 es el que peor funciona descubriendo claquetas. Tras los resultados obtenidos con imágenes, lo esperado era que, ya que era de los que mejor descubría duplicados en los que el objeto repetido estaba desplazado, emparejara en mayor grado las imágenes de la base de datos con las del vídeo.

El Método 1, en sus dos versiones, ofrece buenos resultados.

5.3.2.5 Tiempos de ejecución

Los tiempos medidos reflejan la demora de los algoritmos en trabajar con los mismos 2000 *frames*.

	Método 1 (8x8)	**Método 1 (16x16)**	**Método 2**
Tiempo (s)	116	106	111

TABLA 5-9: Tiempos de ejecución

Tiempos de ejecución en un ordenador con procesador Intel Core 2 Duo a 2.40 GHz y con 3.25 GB de RAM.

6 Conclusiones y trabajo futuro

6.1 Conclusiones

Para concluir el proyecto se puede decir que se han cumplido los objetivos propuestos en su inicio. Se han estudiado y desarrollado algoritmos, tanto a nivel de píxel como a nivel de bloque, que permitan la valoración de una imagen según su homogeneidad, exposición y borrosidad y se ha obtenido una relación óptima que los vincule: la función de calidad. Además, se ha conseguido obtener una serie de representaciones de una imagen (a nivel de píxel y a nivel de bloque) que permiten la comparación entre ellas y la posterior clasificación de dichas imágenes como duplicadas o no, según diferentes medidas y umbrales. Para probar el funcionamiento de los algoritmos implementados, se ha desarrollado una aplicación que trabaja ordenando y numerando imágenes según su fecha de toma, su calidad y su pertenencia a un grupo similar. Asimismo, se ha creado una interfaz gráfica que facilita el entendimiento y la visualización de las clasificaciones que realizaría la aplicación con distintos umbrales y pesos. Para terminar, se ha ampliado a secuencias de vídeo la utilización de los algoritmos a nivel de bloque, proporcionando una herramienta de valoración de *frames* en cuanto a calidad así como de detección de secuencias determinadas (cámara grabando sola) y tomas concretas (claquetas, cortinillas, cartas de ajuste…).

La conclusión final tras todos los estudios realizados en este Proyecto Fin de Carrera sería que no hay algoritmo que funcione mal, siempre que concuerde con los objetivos finales de la aplicación donde se ejecute.

Cada algoritmo (ya sea de medida de calidad o de detección de duplicados) podría considerarse el más adecuado según las características de las imágenes o vídeos con los que va a trabajar y las exigencias del usuario. De esta forma, existirán sets de imágenes que, por su contenido, se clasifiquen mejor con un algoritmo que con otro o que, dependiendo de las requerimientos del usuario, la elección radique en el número de falsos positivos o falsos negativos.

6.1.1 Calidad

El algoritmo de la transformada de Haar trabaja excepcionalmente con primeros planos donde el fondo de la imagen es uniforme, como fotos de carné. Es capaz de discernir claramente si los rostros se encuentran borrosos a pesar de ocupar menos de un 50% de la imagen. Además, es el algoritmo de calidad menos sensible al ruido, lo que lo hace ideal para la detección de imágenes subexpuestas, donde en muchas ocasiones aparece el típico ruido de sal y pimienta. Como contrapartida, es menos útil clasificando paisajes.

El operador de Sobel es de los mejores clasificando paisajes, edificios o fotografías de contenido montañoso, puesto que reconoce y valora las pequeñas variaciones que se producen en cielos, rocas, etc. Sin embargo, las fotografías en las que aparecen personas, independientemente del fondo, no son su fuerte. Es, además, muy sensible al ruido.

El algoritmo de la DCT, en sus dos versiones (nivel de píxel y nivel de bloque), trabaja mejor con fotografías de contenido general que con paisajes, precisamente porque su punto débil son los fondos uniformes como cielos o las paredes blancas que aparecen en las fotos de carné. Al igual que le sucedía a Sobel, la DCT aumenta perceptiblemente la calidad de las imágenes ruidosas.

En cuanto al cálculo de la exposición, la entropía ha demostrado ser una buena medida de detección de imágenes o *frames* sobre/sub expuestos, aunque la inclusión de la evaluación adicional de los coeficientes AC de la DCT mejora en mucho los resultados obtenidos.

En cuanto a vídeo, concluir que resultaría adecuado relajar los umbrales de calidad según el tipo de secuencia, ya que en muchas ocasiones la borrosidad del fondo o la oscuridad de la imagen, por ejemplo, han de ser perdonadas a cambio de primeros planos o emoción.

6.1.2 Duplicados

A nivel de píxel, los algoritmos que mejor funcionan a nivel general son *Dominant Color* y *Color Layout*, aunque en paisajes montañosos el Color Medio tenga más aciertos que *Color Layout*. *Dominant Color* realiza una búsqueda exhaustiva de los colores representativos de una región (en este caso de tamaño 8x8) que funciona muy bien, pero que también produce un alto porcentaje de falsos positivos si las imágenes tienen colores similares, como sucede con fotografías del suelo, de arboledas o de carnet. *Color Layout* tiene a su favor que, al examinar el tipo de variaciones que se encuentran en las imágenes, es capaz de emparejar fotografías en las que el objeto se encuentra desplazado ligeramente en una con respecto a la otra.

A nivel de bloque, el Método 2 (Método 1 (8x8) y *Color Layout*) es el que mejores resultados ofrece detectando duplicados de cualquier tipo, aunque no supera en aciertos a ninguno de los algoritmos a nivel de píxel. El tamaño de la imagen representación en el Método 1, curiosamente, no afecta a los resultados finales.

Por último, en cuanto a la intervención de los algoritmos en el programa de vídeo, señalar que, pese a no estar pensados para la detección de claquetas (en mi opinión considero que la aplicación de algoritmos de detección de objetos obtendría unos resultados más óptimos), los éxitos conseguidos avalan su funcionamiento. No obstante, son ideales para la localización de secuencias en los que se queda la cámara encendida.

6.2 Trabajo futuro

Como sugerencias para mejorar el funcionamiento de los algoritmos implementados citaría el utilizar algunas características de los descriptores, como *SpatialCoherency* en *Dominant Color*, probar otro tipo de submuestreo cuando se trabaja con *Color Layout* o los métodos a nivel de bloque, estudiar si se puede realizar algún tipo de filtrado de ruido en algoritmos como el del operador de Sobel o la DCT o hacer posible la comparación de imágenes verticales y apaisadas en la aplicación Repositorio de imágenes.

Sin embargo, creo que el trabajo futuro más fructífero sería el de seguir imaginando, investigando, implementando y probando cualquier algoritmo o método que tenga posibilidades de tener éxito, puesto que es así como de verdad se avanza.

Referencias

[1] X. Hua, L. Lu, H. Zhang, "Photo2Video – A system for automatically converting photographic series into video", *IEEE Transactions on Circuits and Systems for Video Technology*, 16(7):803-805, July 2006.

[2] X. Hua, L. Lu, H. Zhang, "Content Based Photograph Slide Show with Incidental Music", *Proc. of IEEE ISCS 2003*.

[3] X. Hua, L. Lu, H. Zhang, "Automatically Converting Photographic Series into Video", *Proc. of ACM Multimedia 2004.*

[4] H. Tong, M. Li, H. Zhang, C. Zhang, "Blur Detection for Digital Images Using Wavelet Transform", *Proc. of ICME 2004*.

[5] X. Marichal, W. Ma, H. Zhang, "Blur Determination in the compressed domain using DCT information", *Proc. ICIP'99*.

[6] M. Cannon, "Blind Deconvolution of Spatially Invariant Image Blurs with Phase", *IEEE Trans. Acoustics, Speech, Signal Processing*, 24(1):58-63, February 1976.

[7] T. S. Hua, K. Tan., B. C. Ooi, "Fast Signature-based Color-Spatial Image Retrieval", *Proc. of ICMCS'97*.

[8] Y. Ke, R. Sukthankar, L. Huston, "Efficient near-duplicate detection and sub-image retrieval", *Proc. of ACM Multimedia 2004.*

[9] R. L. Lagendijk, J. Biemond, "Basic Methods for Image Restoration and identification", *A. Bovik, Handbook of Image & Video Processing, Academic Press,* 1999.

[10] J. G. Proakis, M. Salehi, "Communication Systems Engineering", *Prentice Hall*, 2nd Edition, 2002.

[11] P. Marziliano, F. Dufaux, S. Winkler, T. Ebrahimi, "A no-reference perceptual blur metric", *Proc.of the ICIP 2002*.

[12] V.V. Vasudevan, B.S.Manjunath, J.-R. Ohm, A. Yamada, "Color and texture descriptors", *IEEE Transactions On Circuits and Systems for Video Technology*, 11(6):703-715, June 2001.

[13] "Text of ISO/IEC 15938-3/FDIS Information Technology - Multimedia Content Description Interface – Part 3 Visual", July 2001.

[14] "Text of ISO/IEC 15938-8/DTR (Extraction and Use of MPEG-7 Descriptions)", March 2002.

[15] "Text of ISO/IEC 15938-8/DAM1", March 2004.

[16] K. Sheppard, D.G. Marcellin, M.W. Marcellin, B.R. Hunt, "Blur identification from vector quantizer encoder distortion", *IEEE Trans. On Image Processing*, 10(3):465-470, March 2001.

[17] G. Pavlovic, A.M. Tekalp, "Maximum likelihood parametric blur identification based on a continuous spatial domain model", *IEEE Trans. On Image Processing*, 1(4):496-504, October 1992.

[18] F. Rooms, A. Pizurica, "Estimating image blur in the wavelet domain", *Proc. of PorRISC 2001*.

[19] F. J. Ceballos, "Curso de programación C/C++", *RA-MA Editorial*, 1995.

[20] http://opencvlibrary.sourceforge.net/

[21] "TIFF Revision 6.0", June 1992.

[22] JEITA CP-3451 "Exchangeable image file format for digital still cameras: Exif version 2.2", April 2002.

[23] http://www.codeguru.com/cpp/g-m/bitmap/displayingandsizing/article.php/c4939

[24] http://www.mesh-ip.eu/?Page=Project
[25] http://www-nlpir.nist.gov/projects/trecvid/
[26] J. Oliver Gil, M. Pérez Malumbres, "Compresión de imagen y vídeo: Fundamentos teóricos y aspectos prácticos", *Editorial de la UPV*, 1ª Edición.
[27] C. S. Burrus, "Introduction to wavelets and wavelets transforms a primer", *Prentice Hall*, 1998
[28] H-G. Stark, "Wavelet and wavelet applications in signal processing", *Springer*, 2005.
[29] JPEG, "ISO/IEC IS 10918-1 | ITU-T Recommendation T.81", 1994.
[30] http://www.jpeg.org/
[31] http://es.wikipedia.org/wiki/Modelo_de_color_RGB
[32] http://msdn.microsoft.com/en-us/library/ms536560(VS.85).aspx
[33] http://es.wikipedia.org/wiki/Modelo_de_color_HSV
[34] http://www.virtual-drums.com/frozen-cameleon.php
[35] http://dba.med.sc.edu/price/irf/Adobe_tg/models/cieluv.html
[36] http://en.wikipedia.org/wiki/CIELUV_color_space
[37] http://en.wikipedia.org/wiki/YCbCr
[38] http://es.wikipedia.org/wiki/YUV
[39] R. Lukav, "Color image processing methods and applications", *CRC Taylor & Francis*, 2007.
[40] José María Martínez Sánchez, "Apuntes de la asignatura Televisión Digital", curso 2006-2007.
[41] ISO/IEC 11172-2 (MPEG-1 Video), "Coding of Moving Pictures and Associated Audio for Digital Storage Media at up to About 1.5 Mbit/s—Part 2: Video", 1993.
[42] ISO/IEC 13818-2 (MPEG-2 Video), "Generic coding of moving pictures and associated audio information—Part 2: Video", 1994.
[43] "Mesh Content Analysis Modules (2nd version)", Deliverable D2.5, FP6-027685 Mesh Project, March 2008.

Glosario

DCT	Discrete Cosine Transform
DFT	Discrete Fourier Transform
EXIF	Exchangeable Image File Format
GoP	Group of Pictures
HSV	Hue Saturation Value
IDCT	Inverse Discrete Cosine Transform
IDE	Integrated Development Environment
ISO/IEC	International Standards Organization/ International Electrotechnical Commission
IVOnLA	Image & Video OnLine-Analysis module
JFIF	JPEG File Interchange Format
JPEG	Joint Photographic Experts Group
MFC	Microsoft Foundation Class
MPEG	Moving Picture Experts Group
NTSC	National Television System Committee
OpenCV	Open Source Computer Vision
PAL	Phase Alternating Line
TIFF	Tagged Image File Format

Anexos

A Manual del programador

A.1 Aplicación Repositorio de Imágenes

A.1.1 Gestión de archivos según calidad

A.1.1.1 Lectura del path

El primer acontecimiento con el que se encuentra el usuario que ejecuta el programa es la solicitud por pantalla de un *path* (o ruta) en el que se localicen las imágenes que desee clasificar. La variable *path* se consulta a lo largo de todo el programa y se considera una variable pública.

Para determinar cuántas imágenes hay en la carpeta señalada por el usuario, se hace una llamada al sistema (*system()*) que vuelque el resultado de ejecutar el comando *dir* en ese *path*. El resultado se almacena en la misma carpeta analizada en un archivo de texto (.txt). A continuación, un bucle se encarga de leer el fichero creado y de contar cuántas terminaciones .jpeg o .JPEG hay escritas en él. Este detalle impide que el programa trabaje con cualquier otro tipo de formatos de imagen. Si el número de imágenes JPEG es cero, se informa de ello al usuario y se sale del programa.

A.1.1.2 Pasos previos a la clasificación

Las dos variables que se crean y rellenan en este módulo del programa son también de utilidad en el módulo siguiente, el de gestión de archivos según duplicados. Ambas ayudan a minimizar el tiempo de ejecución, sacrificando memoria por tiempo.

La primera de ellas es una tabla que almacena todos los nombres de las imágenes presentes en la carpeta original junto con sus calidades. En el módulo 1 es necesario calcular la calidad de las fotos debido a que se envían a la carpeta "*Trash*" en función del valor de ésta. Como el cálculo de la calidad es uno de los que más tiempo conlleva, el almacenamiento de los valores y la consecuente búsqueda en la tabla por identificador (en este caso, el nombre) merece la pena.

La segunda variable es otra tabla, pero en esta sólo se almacenan las distintas fechas de las carpetas que se vayan creando. De esta forma, cuando el método 2 tenga que encontrar duplicados dentro de ellas, no será necesario ejecutar otro *dir* e investigar dentro del fichero para descubrir sus nombres.

La longitud de ambas tablas de corresponde con el número de imágenes que haya en el fichero. En el caso de la tabla de calidades, el máximo número de caracteres que un nombre puede tener es de 240 (longitud máxima de los nombres de los archivos en Windows). Las fechas tienen un tamaño fijo: 11 caracteres. Los 10 que corresponden a números o guiones más el carácter de fin de línea (14-02-2003_).

A.1.1.3 Algoritmo de clasificación

El algoritmo de clasificación se encarga de leer una a una las líneas del archivo creado tras ejecutar el *dir*. Comprueba si la línea en cuestión contiene información de imagen (.jpeg o .JPEG) y, si es así, obtiene el nombre de la fotografía, presente en esa línea. Accediendo a las cabeceras de dicha imagen, extrae la fecha en la que fue tomada, si es que existen campos EXIF, de la forma en la que se explica en el anexo F. Si la imagen no tuviera campos EXIF, la fecha se establece como "*Unknown*". Es necesario cambiar el símbolo ':', que aparece en la cabecera JPEG como separador entre el año, el mes y el día, por el símbolo '-', puesto que la fecha será posteriormente el nombre de una carpeta, y Windows no permite que en los nombres de archivos y carpetas aparezca ':'.

Como el objetivo es crear una carpeta cada vez que se encuentre una nueva fecha, y conociendo que el comando *dir* no las ordena necesariamente por este campo, es obligado comprobar si la fecha recién obtenida no ha sido ya procesada. Esto es tan sencillo como revisar la tabla de fechas de la que se hablaba en el apartado anterior (A.1.1.2).

Si es una nueva fecha, se copia como nueva en la lista y se crean las carpetas correspondientes (comando *mkdir*). Si no lo es, se pasa directamente a trabajar con la imagen.

En esta parte del programa (trabajo con la imagen), se carga la fotografía con OpenCv y se llama a los algoritmos de calidad (4.3.1.1 o 4.3.2.1). Según el valor obtenido, la imagen se mueve a una u otra carpeta (comando *move*) y se almacenan su nombre y su calidad en la tabla de calidades (apartado A.1.1.2).

A.1.2 Gestión de archivos según duplicados

A.1.2.1 Lectura de los paths

Como el análisis se efectúa por carpetas, para poder acceder a éstas es necesario contar con su nombre. En un principio, esta operación se podría realizar de la misma forma que en el módulo 1: mediante un archivo de texto con los resultados de ejecutar *dir*.

Pero contando con la presencia de la tabla de fechas, basta con recorrerla entrada a entrada aplicando los puntos siguientes (A.1.2.2, A.1.2.3, A.1.2.4 y A.1.2.5) a cada carpeta.

Tras acceder a una de ellas, ahora sí se crea un archivo .txt con el listado de las imágenes presentes en la carpeta. Si el número de imágenes es cero (caso que podría suceder si todas las fotografías de esa fecha son de mala calidad) o uno, no existe comparación posible, y se pasa a estudiar el siguiente directorio.

A.1.2.2 Creación de la tabla

La tabla es la variable principal del programa. Consiste en un listado de *n* filas, donde *n* sería el número de imágenes encontradas en una carpeta de fecha en concreto.

Cada fila contiene información de cada imagen: nombre (*name*), calidad (*Q*), de qué imagen de mayor calidad es ésta duplicada (*dup*), qué posición ocupa entre las que son duplicadas entre sí según la calidad (*orden*), cuáles son las imágenes que se consideran duplicadas suyas (*mod*), su representación de duplicados (*cl*) y la longitud del campo representación (*length*).

name	*Q*	*dup*	*orden*	*mod*					*representación*					*length*
	0.0	-2	0		F	F		F	(0,0,0)	(0,0,0)	(0,0,0)		(0,0,0)	0
	0.0	-2	0			F		F	(0,0,0)	(0,0,0)	(0,0,0)		(0,0,0)	0
	0.0	-2	0					F	(0,0,0)	(0,0,0)	(0,0,0)		(0,0,0)	0
. . .							…					…		
	0.0	-2	0						(0,0,0)	(0,0,0)	(0,0,0)		(0,0,0)	0

FIGURA A-1: Campos de la variable "tabla" utilizada en el programa

La tabla se rellena leyendo una a una las líneas del archivo .txt. Si la línea contiene información de imagen, se recoge su nombre, mediante el que se obtiene su calidad de la tabla de calidades que se creó en el módulo 1 (A.1.1.3). Estos datos se guardan en la tabla, y los campos *mod*, *dup* y *orden* se inicializan íntegramente a *false*, 0 y -2 respectivamente.

A.1.2.3 Aplicación de los algoritmos

Este paso se encarga de calcular la representación de duplicados (algoritmos de detección de duplicados, apartado 4.3.1.2 o 4.3.2.2), es decir, la imagen en miniatura que representará a la fotografía a la hora de su comparación. La representación, junto con su longitud (64 en el caso de *Color Layout*, 1/64 de las dimensiones de la imagen si se trata del Color Medio, 64 (8x8) o 256 (16x16) si se escoge el Método 1…), se guardan en los campos de la tabla apropiados.

Calculadas sus representaciones, es el momento de determinar qué imágenes son duplicadas de otras, y esto se refleja en la tabla rellenando el campo *mod*.

El algoritmo va recogiendo una a una las entradas de la tabla, comparándolas mediante el método correspondiente al algoritmo de duplicados con el que se está trabajando. Para ello se extrae la representación de la imagen que se está estudiando y la de las *n* que van a continuación suya, se calcula la distancia que hay entre ellas y, si es menor que un umbral, se pone a true la casilla adecuada en el campo *mod*.

	name	*Q*	*dup*	*orden*	*mod*									
					0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
0	Im_0.jpg	7.49	-2	0		F	F	F	F	F	F	F	F	F
1	Im_1.jpg	7.16	-2	0			T	F	F	F	F	F	F	F
2	Im_2.jpg	7.18	-2	0				F	F	F	F	F	F	F
3	Im_3.jpg	7.43	-2	0					T	F	F	F	F	F
4	Im_4.jpg	7.68	-2	0						T	F	F	F	T
5	Im_5.jpg	7.95	-2	0							F	F	F	F
6	Im_6.jpg	7.78	-2	0								F	F	F
7	Im_7.jpg	7.20	-2	0									F	F
8	Im_8.jpg	8.15	-2	0										T
9	Im_9.jpg	8.02	-2	0										

FIGURA A-2: Ejemplo de "tabla" con los campos *mod* calculados

A.1.2.4 Organización de duplicados

El algoritmo a continuación es el que se encarga de realizar toda la ordenación de las imágenes duplicadas. Hay que tener en cuenta el caso de que una imagen puede ser duplicada de una segunda, y esta segunda de una tercera, pero que pueden no serlo la primera y la tercera. Si sucede, la aplicación trata a las tres como duplicadas. Es por eso que la lista de fotografías duplicadas entre sí debe ser una lista enlazada a la que se le puedan seguir uniendo elementos a medida que se estudian nuevas entradas.

El campo *dup* es importante porque indica al algoritmo si esa imagen no es duplicada de ninguna o aún no ha sido procesada (-2), si está en pleno análisis (-1) o cuál es la imagen original de la que es duplicada (índice de la imagen).

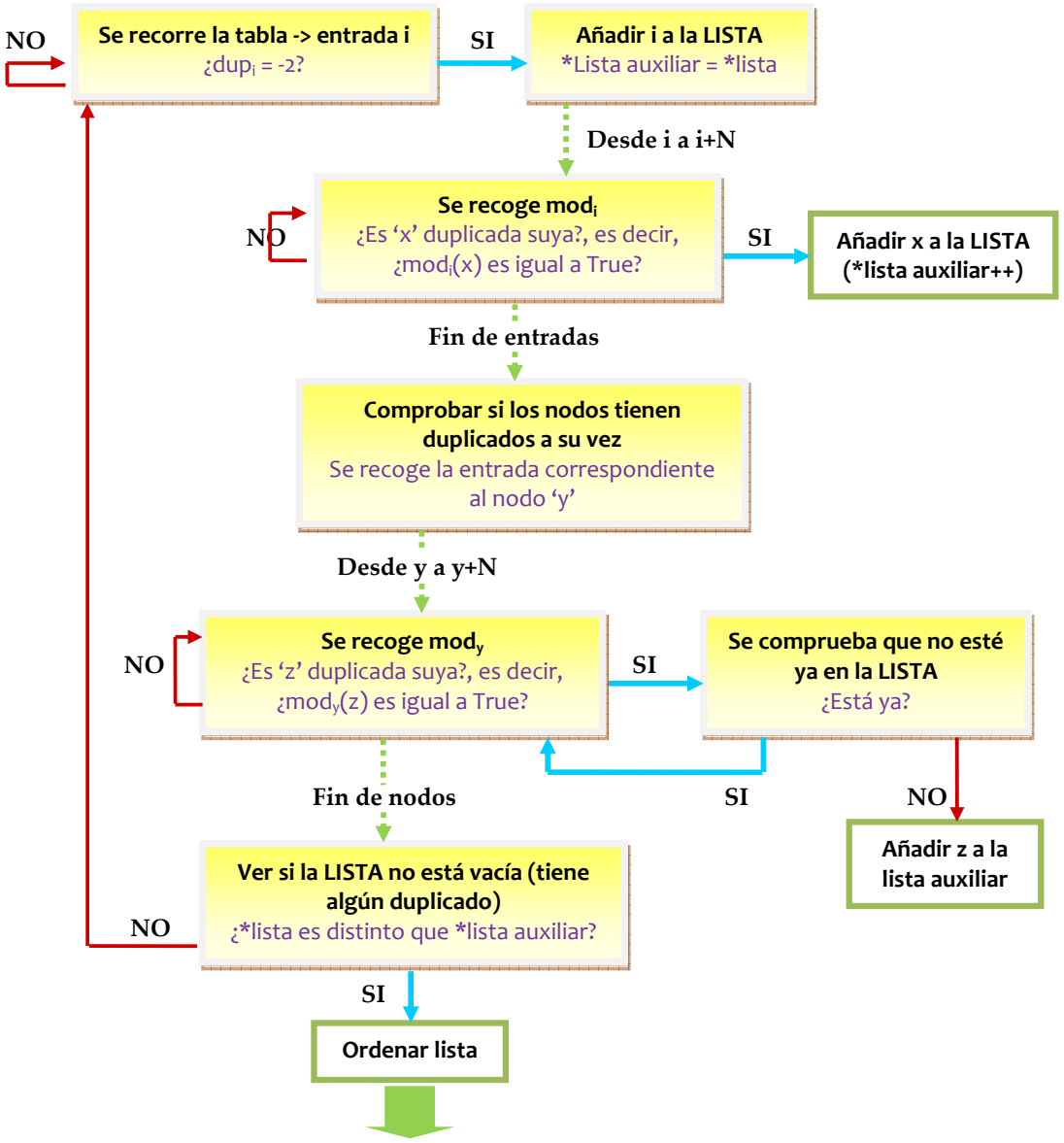


FIGURA A-3: Diagrama de bloques de la creación de la lista enlazada

El algoritmo consiste en lo siguiente: se recorre la tabla desde el primer elemento hasta el penúltimo, examinando el campo *dup* de cada entrada. Si está a -2, se procede a crear una lista con los índices de las imágenes que, según *mod*, son duplicadas suyas. Si ha habido algún acierto, se calcula el número de nodos que se encuentran en la lista y se crea una tabla con los índices de las imágenes involucradas. Esta tabla es la que se ordena de mayor a menor según calidad. Para terminar, se cambian los campos *dup* de todas las entradas por el índice de la que tiene mayor calidad, y en *orden* se escriben los índices de la tabla de ordenación, de manera que quede claro en qué orden van las imágenes que son duplicadas de la que aparece en *dup*.

Los diagramas de bloques de las figuras A-3 y A-4 explican esquemáticamente la creación de la lista enlazada y la ordenación por calidad de todas las imágenes duplicadas relacionadas en ella.

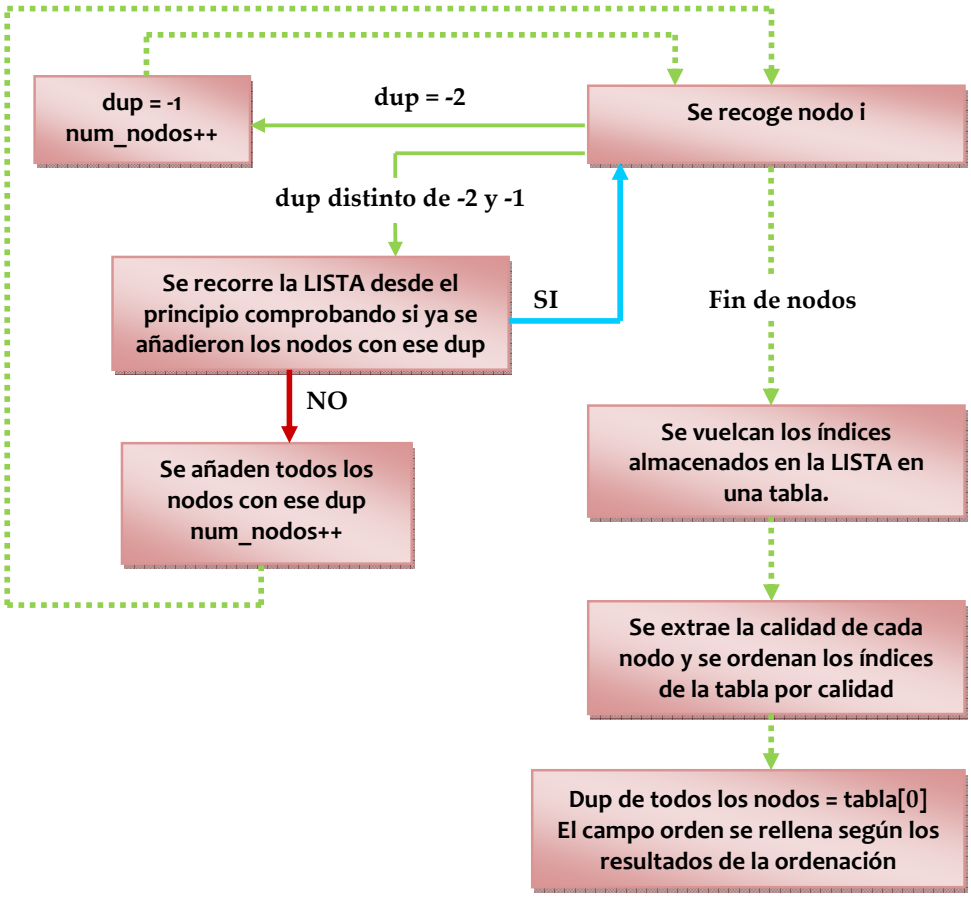


FIGURA A-4: Diagrama de bloques de la ordenación por calidad de las imágenes duplicadas

Si *lista* (que apunta al primer nodo de *LISTA*) es distinto de *lista auxiliar* (que apunta al último nodo), es porque en el proceso de creación de la lista enlazada *LISTA* ha habido algún acierto, y la entrada *i* de la tabla tenía algún duplicado. Si es así, el siguiente paso es añadir los nodos que faltan y ordenarlos por calidad. Existe la posibilidad de que haya nodos sin añadir a la lista porque se puede dar el caso de que la entrada *i* se relacione con la *j*, pero ninguna de ellas con la entrada *k*, y que en un análisis posterior la entrada *k* resulte duplicada de *j*. Este y otros casos se estudiarán seguidamente con un ejemplo.

Para comprender mejor el funcionamiento del algoritmo en todos los casos, se realiza a continuación un ejemplo basado en la tabla de la figura A-2.

La primera entrada a procesar es la 0, es decir, la imagen de nombre *Im_0.jpg*. Como su campo *dup* es igual a -2, se crea una lista enlazada cuyo primer nodo tiene como dato 0, que es el índice de la tabla donde se encuentra esta imagen. Cuando se examinan sus campos *mod*, se comprueba que no tiene duplicados, y se elimina la lista creada.

La siguiente entrada es la de índice 1, y como su *dup* está a -2, se examina. En este caso sí que hay un acierto en uno de sus campos *mod*, correspondiente a la imagen con índice 2, que es añadido a la lista. Tras examinar todos los *mod*, se pasa a estudiar los de la entrada 2, que no tiene más duplicados. Al permanecer todos los *dup* a -2, el número de nodos calculado por el algoritmo se mantiene a 2. Como la imagen asociada a la entrada 2 tiene una mayor calidad que la de la entrada 1, el campo *dup* de ambas se fija a 2, el campo *orden* de 1 a 2 y el *orden* de 2 a 1.

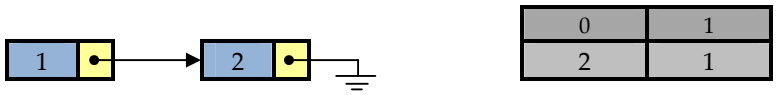


FIGURA A-5: Lista de la entrada 1 y tabla de ordenación según calidad correspondiente

Esta vez la entrada a continuación (la 2) tiene el campo *dup* a 2, por lo que no se trabaja con ella.

La entrada 3, sin embargo, sí presenta un *dup* a -2, y además es duplicada de la imagen de la fila 4. La entrada 4, a su vez, tiene a *true* las casillas mod asociadas a las imágenes en las posiciones 5 y 9. Al no encontrarse ninguna de ellas aún en la lista, son agregadas también. Examinando primero la 5 y después la 9, se observa que no existe ningún acierto más. Como todas ellas tienen su dup a -2, el número de nodos de la lista sigue siendo 4 que, ordenados de mayor a menor calidad, fijan finalmente su *dup* a 9. El campo *orden* pasa a ser: 4 para la entrada 3, 3 para la 4, 2 para la 5 y 1 para la 9, la de mayor calidad.

La tabla a continuación refleja los cambios realizados hasta este punto.

					mod									
	name	*Q*	*dup*	*orden*	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
0	Im_0.jpg	7.49	-2	0		F	F	F	F	F	F	F	F	F
1	Im_1.jpg	7.16	2	2			T	F	F	F	F	F	F	F
2	Im_2.jpg	7.18	2	1				F	F	F	F	F	F	F
3	Im_3.jpg	7.43	9	4					T	F	F	F	F	F
4	Im_4.jpg	7.68	9	3						T	F	F	F	T
5	Im_5.jpg	7.95	9	2							F	F	F	F
6	Im_6.jpg	7.78	-2	0								F	F	F
7	Im_7.jpg	7.20	-2	0									F	F
8	Im_8.jpg	8.15	-2	0										T
9	Im_9.jpg	8.02	9	1										

FIGURA A-6: Estado de la tabla antes de examinar la entrada 4

El análisis de las entradas 4 y 5 es rápido, puesto que no se crea lista para ninguna de ellas debido a que sus *dup* se encuentran a 9.

Las entradas 6 y 7 tienen todos sus campos *mod* a false, y no se realiza ninguna ordenación.

La entrada 8, por el contrario, sí tiene duplicados, y además es un caso especial. Cuando comienza a estudiarse, la lista que pasa al algoritmo de ordenación es de 2 nodos (el 8 y el 9). Pero como el nodo 9 tiene su *dup* a 9, se entiende que todos los duplicados de la imagen *Im_9.jpg* lo sean también de 8. El número de nodos que se cuentan es de 5 (los 4 de la lista anterior y el 8), que son los que se ordenan por calidad.

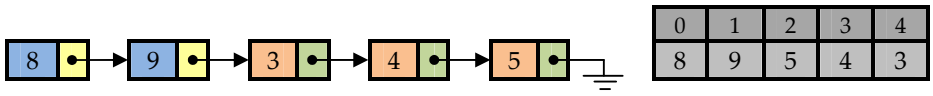


FIGURA A-7: Lista de la entrada 8 y tabla de ordenación según calidad correspondiente

La última fila de la tabla, la 9, nunca se analiza porque todos sus *mod* están a *false*. Esto es porque ya se han calculado todas las distancias entre ella y las demás imágenes, por lo que no aporta información nueva. El aspecto final de la tabla antes de pasar al siguiente módulo de renombrar imágenes es el siguiente:

					mod									
	name	*Q*	*dup*	*orden*	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
0	Im_0.jpg	7.49	-2	0		F	F	F	F	F	F	F	F	F
1	Im_1.jpg	7.16	2	2			T	F	F	F	F	F	F	F
2	Im_2.jpg	7.18	2	1				F	F	F	F	F	F	F
3	Im_3.jpg	7.43	8	5					T	F	F	F	F	F
4	Im_4.jpg	7.68	8	4						T	F	F	F	T
5	Im_5.jpg	7.95	8	3							F	F	F	F
6	Im_6.jpg	7.78	-2	0								F	F	F
7	Im_7.jpg	7.20	-2	0									F	F
8	Im_8.jpg	8.15	8	1										T
9	Im_9.jpg	8.02	8	2										

FIGURA A-8: Estado de la tabla al término del módulo de organización de duplicados

A.1.2.5 Cambio de nombre a las imágenes

El último paso de este módulo consiste en renombrar las imágenes de forma que quede claro cuáles son duplicadas de cuál y en qué orden deberían ser conservadas (en orden descendente de calidad).

La metodología es la siguiente: la imagen que no tenga duplicados (campo *dup* a -2), mantiene su nombre original; la fotografía que cuente con duplicados y sea la de mayor calidad (campo *orden* a 1), agrega a su nombre inicial el símbolo '_' y a continuación el número 1; si la imagen es duplicada de otra pero no es la de mejor calidad, absorbe el nombre de la que esté en la entrada que indica su *dup* y le añade el símbolo '_' seguido de su número de *orden*.

Cuando todos los nombres de la tabla estén cambiados, se hace una llamada al sistema con el comando *rename*, que los modifica en el fichero donde se encuentren guardadas.

Siguiendo con el ejemplo del apartado anterior, la carpeta final ofrecería este aspecto:

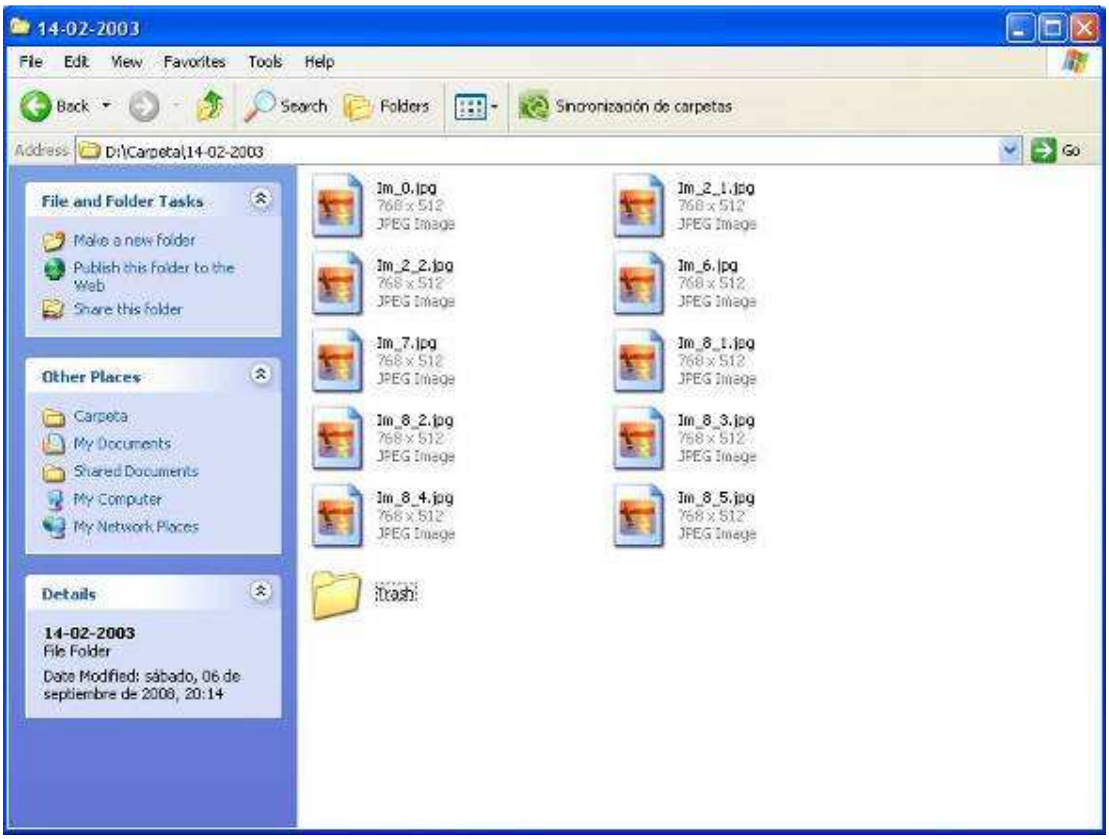


FIGURA A-9: Aspecto final de la carpeta del ejemplo

Ejecutados estos pasos, la carpeta de imágenes cuyo *path* se dio al comienzo del programa queda organizada por calidad y duplicados.

A.2 Interfaz gráfica

La explicación secuencial de los módulos permite un mejor entendimiento del funcionamiento de la interfaz, mostrando los distintos resultados que se pueden producir según qué botones sean pulsados en cada módulo.

A.2.1 Ventana principal: Inicio

Esta ventana principal permanecerá abierta a lo largo de toda la ejecución, y en ella se mostrarán las miniaturas de las imágenes estudiadas con sus datos correspondientes, la fecha en la que se tomaron las fotografías expuestas, los umbrales y los pesos con los que se ha realizado el cálculo y, en el caso de que el número de imágenes en la carpeta sea muy elevado y haya que dividir la exposición por partes, se imprimirán las coordenadas de la parte que se está observando.

Al inicio de la aplicación surge una ventana grande con un área en blanco y cinco botones: *Nuevo Path, Recalcular, Fecha Siguiente, Imagen Siguiente* e *Imagen Anterior*. Los cuatro últimos se encuentran deshabilitados cuando arranca la aplicación dejando como única opción a pulsar el botón *Nuevo Path*.

A.2.2 Ventanas "Nuevo Path" y "Métodos"

Este módulo comienza una vez el botón *Nuevo Path* ha sido presionado. Se encarga de recoger la ruta de la carpeta objetivo, de comprobar que no es la misma que la anterior, de recoger la elección de los Métodos que desee utilizar el usuario para los cálculos y comparaciones y de demostrar que existen imágenes en esta carpeta.

Antes incluso de mostrar la nueva ventana, la primera acción es deshabilitar el botón *Fecha Siguiente*, por si fuera el caso de que tras haberse producido ya una ejecución se desease trabajar con otro directorio. Inmediatamente después se abre la ventana, conteniendo un mensaje con las instrucciones de su funcionamiento y un campo *Edit Control* para la introducción de los datos.

En el caso de que la ventana se cierre con el botón *Cancel* o con el aspa, el programa vuelve al inicio a la espera de que se vuelva a pulsar el botón *Nuevo Path*. Si, por el contrario, la ventana se cierra con el botón *OK*, la aplicación recoge el contenido de la barra y comprueba si se han llegado a introducir datos, mostrando una ventana de aviso (*AfxMessageBox*) con el mensaje "No se han introducido datos" si la variable estuviese vacía.

Habiendo asegurado que existe una ruta, se abre el diálogo que permite la elección de los algoritmos que se utilizarán para los cálculos. En la interfaz que trabaja con los algoritmos a nivel de píxel, se trata de una ventana con dos *Group Box* que certifican que sea imposible marcar dos métodos para el mismo fin. La que implementa los algoritmos a nivel de bloque cuenta con un único *Group Box* para elegir entre los métodos.

Si esta ventana de diálogo se cancela o se cierra con la cruz, se regresa al punto de inicio, donde sólo se puede pulsar el botón *Nuevo Path*. Si se cierra pulsando *OK*, se comprueba que se han elegido algoritmos tanto para calcular el factor de calidad como para resolver si dos imágenes son duplicadas. En el caso de que no haya sido así, se lanza un mensaje de aviso y se vuelve a la ventana inicial.

Si todo el proceso se ha desarrollado correctamente, se realiza otra comprobación más para descartar que el *path* introducido sea igual que el anterior, tras lo que se regresaría al comienzo. Si, llegados a este punto, el *path* es nuevo, se borrarían las imágenes con extensión .bmp creadas en el directorio anterior (si se tratase de una ejecución tras otra) y se almacenaría la ruta en una variable necesaria a lo largo del resto del programa.

El esquema de flechas refleja los resultados que se obtendrán según qué pasos se den en este primer módulo del programa.

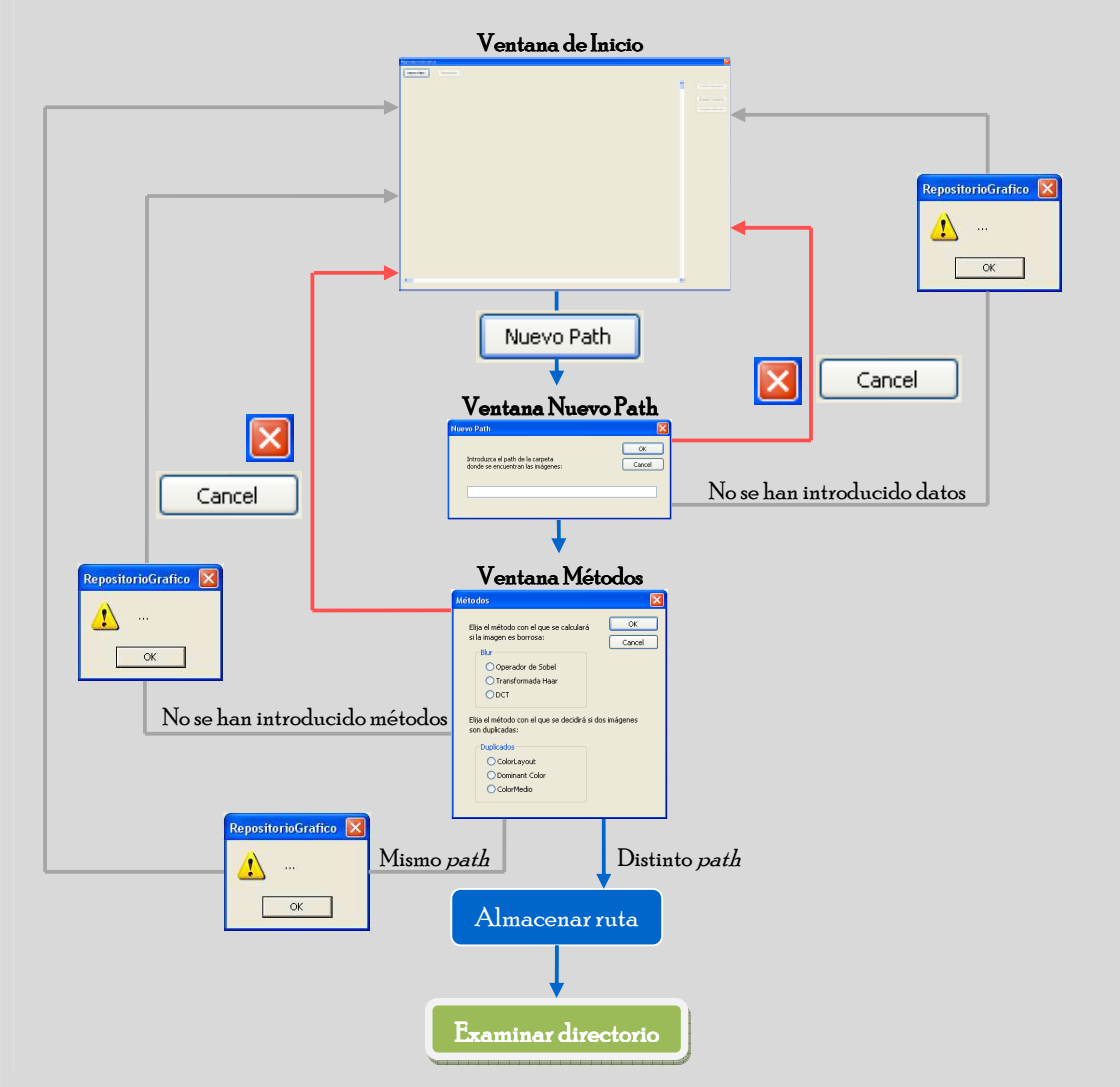


FIGURA A-10: Diagrama de flechas del módulo de introducción de ruta y elección de algoritmos

A.2.3 Ventana principal: Examinar el directorio

El módulo "Examinar directorios" se encarga de realizar una de las tareas principales del programa. Al igual que sucedía con la aplicación repositorio de imágenes (5.2 y A.1), el repositorio gráfico también utiliza una variable *tabla* para almacenar la información obtenida, ya sea un valor de calidad, el nombre de la imagen o qué otras imágenes se consideran duplicadas suyas.

Uno de los objetivos del programa es permitir al usuario establecer los pesos que intervienen en la función de calidad (a nivel de píxel) y también los de exposición (entropía y coeficientes AC, a nivel de bloque), y que éste los cambie para repetir los cálculos cuantas veces desee. Calcular los valores de borrosidad, entropía, etc. que contribuyen a la medida de la calidad de la imagen es una de las partes del programa que más tiempo de ejecución necesita, así como la obtención de las representaciones para la detección de duplicados. Es importante comprender que sólo es preciso realizar estas operaciones una sola vez. Por mucho que cambien las contribuciones a la función de calidad o los umbrales, las medidas que dependen únicamente de la imagen no van a variar.

Este módulo se responsabiliza de realizar los cálculos que van a permanecer constantes a lo largo del programa y de almacenar los datos obtenidos en la tabla para su posterior promediado, comparación, etc.

name	*Q*	*ch*	*q_bl*	*dup*	*orden*			*mod*			*representación*					*length*
	0.0	0.0	0.0	-2	0		F	F		F	(0,0,0)	(0,0,0)	(0,0,0)		(0,0,0)	0
	0.0	0.0	0.0	-2	0			F		F	(0,0,0)	(0,0,0)	(0,0,0)		(0,0,0)	0
	0.0	0.0	0.0	-2	0					F	(0,0,0)	(0,0,0)	(0,0,0)		(0,0,0)	0
. . .									…					…		
	0.0	0.0	0.0	-2	0						(0,0,0)	(0,0,0)	(0,0,0)		(0,0,0)	0

FIGURA A-11: Campos de la variable "tabla" utilizada en la interfaz gráfica a nivel de píxel

name	*Q*	*q_bl*	*ch*	*varACs*	*dup*	*orden*			*mod*			*representación*				*length*
	0.0	0.0	0.0	0.0	-2	0		F	F		F	(0,0,0)	(0,0,0)	(0,0,0)		(0,0,0)	0
	0.0	0.0	0.0	0.0	-2	0			F		F	(0,0,0)	(0,0,0)	(0,0,0)		(0,0,0)	0
	0.0	0.0	0.0	0.0	-2	0					F	(0,0,0)	(0,0,0)	(0,0,0)		(0,0,0)	0
. . .										…					…		
	0.0	0.0	0.0	0.0	-2	0						(0,0,0)	(0,0,0)	(0,0,0)		(0,0,0)	0

FIGURA A-12: Campos de la variable "tabla" utilizada en la interfaz gráfica a nivel de bloque (se añade una columna para la nueva medida de la variación de los coeficientes AC)

Las acciones que lleva a cabo el módulo "Examinar directorios" son principalmente tres: crear, renombrar y eliminar archivos .txt, rellenar la tabla de información y habilitar y deshabilitar los botones adecuados.

El primer paso consiste en liberar la variable *tabla* y en borrar el archivo *names.txt* (por si ya se ha ejecutado el programa al menos una vez y se ha introducido un nuevo *path*). A continuación se crea el archivo *names.txt* que contiene la información equivalente a ejecutar el comando *dir* en el directorio dado. De esta forma se comprueba que la carpeta contiene alguna imagen JPEG y es una manera sencilla de obtener sus nombres.

En este punto es el momento de crear y llenar, al menos parcialmente, la variable *tabla*. El algoritmo funciona del modo siguiente: la fecha en la que fue tomada la imagen cuyo nombre está presente en la primera línea del archivo se convierte en la fecha con la que se trabajará primero; a medida que se van leyendo líneas, se clasifican las imágenes por fecha de forma que los nombres de las que coincidan con la fecha de trabajo se introducen en una tabla auxiliar y los de las que no coincidan se guardan un archivo de apoyo para cuando se trabaje con la fecha siguiente; cuando se ha terminado de leer el fichero, se crea la variable *tabla* con un número de filas igual al número de imágenes que se hayan obtenido, y se almacenan en ella los nombres.

Seguidamente, se borra el fichero antiguo y se renombra el archivo de apoyo con el nombre de *names.txt*. Si se da el caso de que no existiera ninguna imagen en la carpeta, se lanza un aviso informando de ello y pidiendo una nueva ruta, con lo que se regresaría a la ventana inicial. Si, por el contrario, sí se hubieran encontrado imágenes, el paso siguiente es deshabilitar los botones *Imagen Siguiente* e *Imagen Anterior*, que podían hallarse en funcionamiento si se hubiera realizado una ejecución previa.

FIGURA A-13: Aviso emergente si la ruta introducida no contiene ninguna imagen JPEG

Para terminar, se recorre la *tabla* calculando las medidas de calidad (coeficiente de *blur*, entropía, variación de los coeficientes AC…) y las representaciones de duplicados (*Dominant Color*, *Color Layout*, Método 1, Método 2…) utilizando los algoritmos que seleccionó anteriormente el usuario. La implementación de estos métodos no varía con respecto a la desarrollada para la aplicación repositorio de imágenes explicada en 5.2.

Si el resultado del proceso es el deseado, se lanza un mensaje de información indicando que es posible el paso al siguiente módulo, y se habilita el botón *Recalcular*.

FIGURA A-14: Ventana que informa al usuario del siguiente paso a seguir

El diagrama de módulos desarrollado a continuación aclara el proceso de funcionamiento que sigue el bloque "Examinar el directorio".

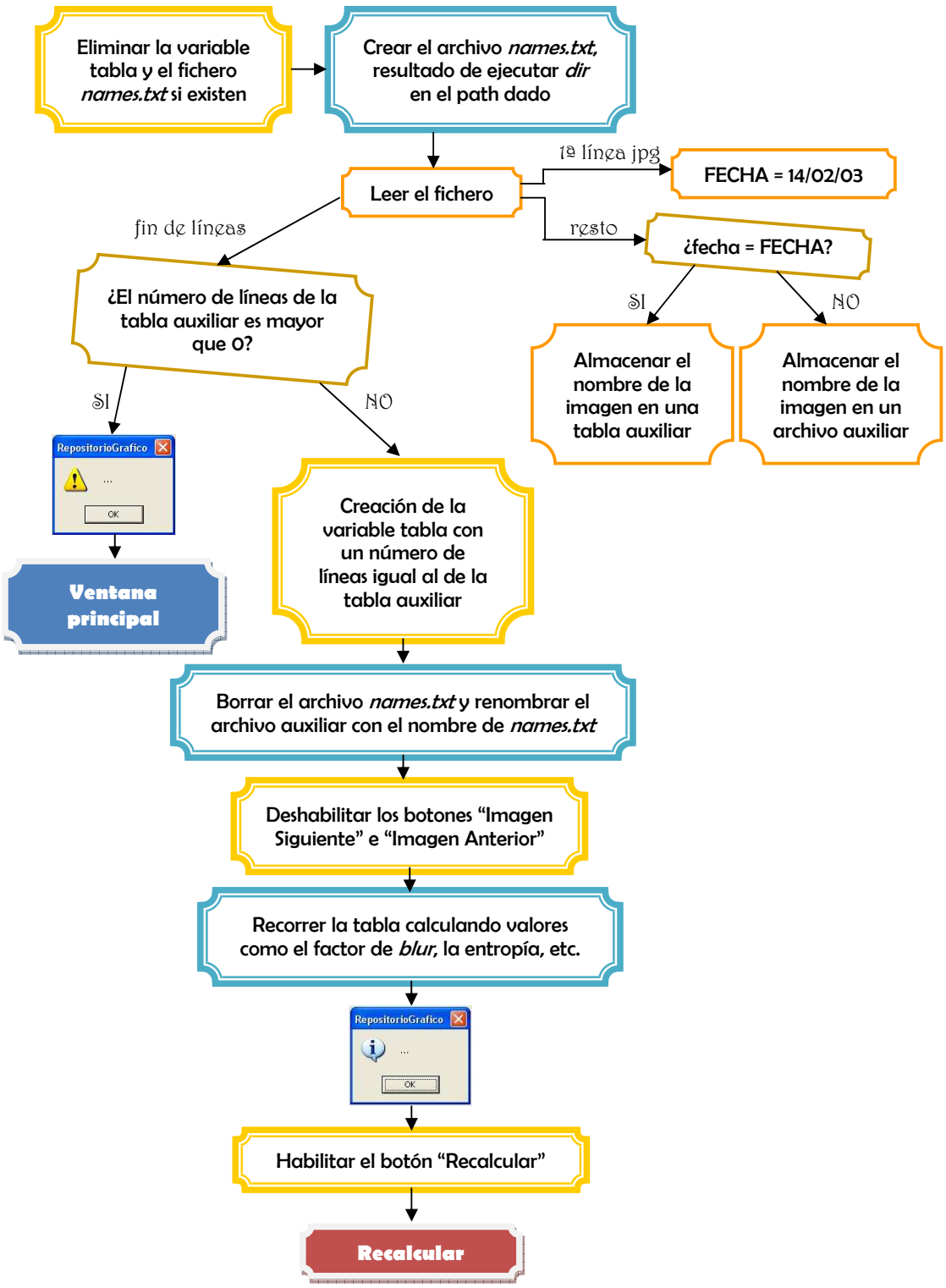


FIGURA A-15: Diagrama de módulos que ilustra sobre el funcionamiento del bloque *Examinar el directorio*

De vuelta a la ventana principal sólo existen dos posibles opciones: pulsar el botón *Nuevo Path* o el botón *Recalcular*. Si se produjo una confusión y no se deseaba analizar la ruta con la que se ha trabajado en "Examinar el directorio", se puede introducir una nueva ruta, con lo que se eliminaría la tabla anterior y el archivo *names.txt* del *path* inicial. Si la ruta era correcta, el siguiente paso es acceder a la ventana que permite la introducción de los datos: la ventana "Recalcular".

A.2.4 Ventana "Recalcular"

La ventana "Recalcular" tiene dos variantes, según la interfaz gráfica que esté en funcionamiento. Si los algoritmos funcionan a nivel de píxel, los campos a introducir se reducen a los umbrales de calidad y duplicados y los pesos que ponderan la entropía y el factor de *blur* en la función de calidad. Si se trabaja a nivel de bloque, a estos campos se les añaden los pesos de entropía y coeficientes AC que participan en el cálculo de la exposición.

Para encaminar al usuario, se inicializan los campos con unas cantidades orientativas, y se indica en un lateral los rangos en los que se mueven los valores solicitados.

El proceso que se inicia tras pulsar *OK* es un proceso largo, por lo que se muestra un mensaje de información que notifica que los datos se han recogido correctamente. Si, por lo contrario, se pulsa el botón *Cancel*, el mensaje que aparece avisa sobre la cancelación del inicio del proceso.

Mensaje tras pulsar OK en la ventana "Recalcular" **Mensaje tras pulsar *Cancel* en la ventana "Recalcular"**

FIGURA A-16: Mensajes tras "Recalcular"

Después de recoger los datos introducidos, pero antes de comenzar a buscar duplicados, ordenar las fotos por calidad, etc., se crean y se imprimen en pantalla la fecha del primer grupo de imágenes a estudiar y los valores escogidos por el usuario.

A.2.5 Ventana principal: Recalcular

El primer paso es restablecer las variables a sus posiciones iniciales por si se diera el caso de que no fuera la primera ordenación que se solicita. En esta etapa de *Reset*, se deshabilitan los botones *Imagen Siguiente* e *Imagen Anterior*, se habilita el botón *Fecha Siguiente*, se elimina cualquier resto de imágenes .bmp creadas anteriormente por el programa y se inicializan los campos *dup* a -2, los campos *orden* a 0 y los campos *mod* a *false*.

Además, hay otras dos variables, *maxdup* y *num_orig*, que es necesario establecer a 1 y 0 respectivamente. Estos datos se utilizan en el módulo "Pintar" para conocer cuál es la altura (número de imágenes originales) y anchura (número máximo de imágenes duplicadas de otra original) que tendrá el mosaico final.

Ahora que se conocen los pesos involucrados en la función de calidad, es el momento de rellenar el campo *Q* de la variable *tabla*.

El siguiente paso es dar valores a los campos *mod* de la tabla, que indican qué imagen es duplicada de otra. Este proceso se realiza de la misma forma que la explicada en el apartado A.1.2.4, solo que en este caso el umbral aplicado al comparar es el impuesto por el usuario.

En este punto se crea una tabla auxiliar donde se van a guardar los índices de las imágenes originales, es decir, aquellas que no tienen ningún duplicado o que son las de mayor calidad en una lista de duplicadas. Esta tabla se utilizará más adelante para construir la imagen mosaico mostrada finalmente en la ventana principal.

Al igual que sucedía en la aplicación Repositorio de Imágenes (A.1), tras establecer los campos *mod* se procedía calculando las relaciones que existían entre las imágenes duplicadas, con el fin de poderlas ordenar mediante los campos *dup* y *orden*. La única diferencia con respecto al apartado A.1.2.4 es que, cuando se determina el número de nodos que forma parte de una lista enlazada, se compara con la variable *maxdup* de manera que si el número de nodos es mayor que ésta, *maxdup* pasa a tener el valor del número de nodos. Esto es equivalente a decir que *maxdup* termina conteniendo el mayor número de duplicados de una imagen original.

Ahora, si el valor de *dup* de una entrada de la *tabla* es igual a -2, u *orden* es igual a 0, el índice de esta entrada se almacena en la tabla auxiliar. Cada entrada añadida hace que aumente en una unidad la variable *num_orig*, de forma que al final *num_orig* contiene el número de imágenes originales. A continuación, este *array* de índices se ordena por calidad.

Realizados estos cálculos, es el momento de pasar al siguiente módulo, encargado de pintar el mosaico de miniaturas.

A.2.6 Ventana principal: Pintar

Este apartado se divide principalmente en dos partes: la que se ocupa de crear la imagen global y de guardarla en disco, y la que recoge la imagen y la muestra en la ventana principal.

La imagen a mostrar se compone de imágenes más pequeñas (de 200x300 píxeles de tamaño) con estos elementos: miniatura de la imagen original (120x180 píxeles), representación de duplicados con la que se ha realizado la comparación (se incluye si hay espacio suficiente) y letreros con información sobre calidad, coeficiente de *blur*, entropía, etc.

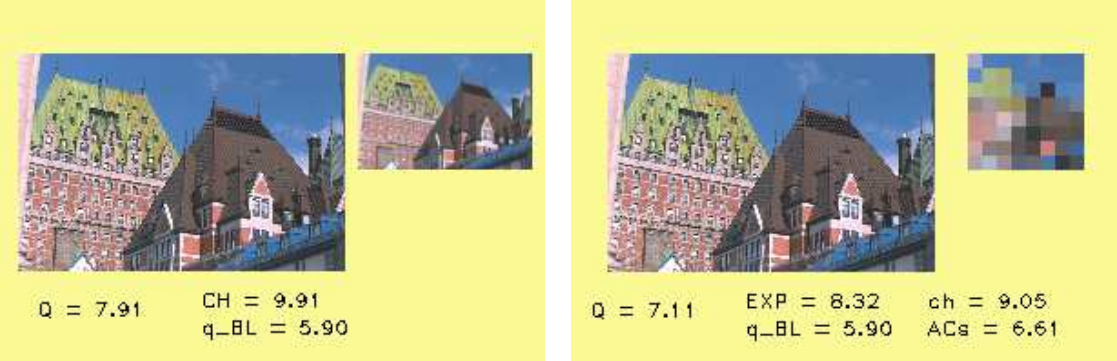


FIGURA A-17: Información mostrada en la ventana principal sobre una imagen cuando se trabaja a nivel de píxel (izquierda) y a nivel de bloque (derecha)

Las imágenes originales se organizarán de mayor a menor calidad en el eje vertical y las imágenes duplicadas se colocarán de mayor a menor calidad en el eje horizontal, seguidas de la imagen original de la que son copia. Las miniaturas de aquellas cuya calidad no supere el umbral propuesto se mostrarán en escala de grises, para que la visualización sea más sencilla.

Por tanto, la altura de la imagen global, en píxeles, será igual a la variable *num_orig* multiplicada por 200, y la anchura igual a *maxdup* por 300. La imagen global se crea con un fondo amarillo pálido y es la que finalmente se mostrará en pantalla. Las imágenes que componen el puzzle se insertan sobre la global mediante las funciones de OpenCv *cvSetImageROI* y *cvCopy*, que permiten seleccionar dónde se colocará y qué tamaño tendrá la figura a insertar.

El siguiente objetivo es rellenar de contenido la imagen global creada. Para ello se recorre el *array* de índices definido en el apartado anterior, que contenía los índices de las imágenes originales ordenadas por calidad. Cada vez que se adquiere un índice, se crea la imagen mosaico correspondiente. Para ello, la primera tarea es construir una imagen de fondo, si no existe. El fondo no es más que una imagen de 200x300 píxeles que se copia continuadamente para convertirse en una imagen mosaico, y se mantiene almacenada a lo largo de toda la ejecución, eliminándose cuando se cierre la aplicación. Sobre la copia de la imagen fondo se insertan los distintos elementos:

- La imagen JPEG con la que se relaciona el índice obtenido se reduce mediante la función *cvResize* y, si se da el caso de que su calidad es menor que el umbral introducido, se convierte a escala de grises. Ocupa la región de la izquierda.
- La imagen que la representa para la comparación de duplicados sólo se calcula si no excede los límites de la imagen base. Si, por lo contrario, es demasiado pequeña (como en el caso de que sea de 8x8 o 16x16 píxeles), se amplía hasta que tenga unas dimensiones de 64x64 píxeles. Se coloca arriba a la derecha.
- Los textos se posicionan con la función de OpenCv *cvPutText*, que permite establecer la situación del rótulo en la imagen y su color.

Una vez introducida la imagen mosaico en la global, se comprueba si ésta tenía duplicados leyendo su campo *dup*. Si es mayor o igual que cero, se busca cuál es el siguiente duplicado de mayor calidad (observando el campo *orden*), se crea su imagen mosaico y se inserta en la posición que le corresponda en la imagen global.

Terminadas de imprimir todas las imágenes de la variable *tabla*, se procede a guardar la imagen global. Este paso es necesario porque, para poder mostrar una imagen en una ventana, ésta tiene que encontrarse en disco y tener formato .bmp. Existe un problema añadido en cuanto a que la interfaz no será capaz de mostrarla si la imagen guardada es demasiado grande. La solución parece encontrarse en dividir la imagen global en piezas si la imagen global supera los 7200 píxeles de ancho o los 7000 de alto.

Si se da el caso, el programa lanza un aviso informando sobre este trabajo añadido. Las porciones serán de tamaño 1200x2400 píxeles como máximo, es decir, estará compuesta de 6x8 imágenes mosaico. El lateral derecho y la parte inferior de la imagen pueden ocupar menos, si las dimensiones de la imagen global no son múltiplo de las medidas de la porción.

Aviso emergente si la imagen global supera los 7000x7200 píxeles

Aviso cuando se haya terminado de almacenar las porciones en disco

FIGURA A-18: Avisos emergentes

Las porciones de la imagen se irán visualizando una por una según el usuario pulse los botones *Imagen Siguiente* o *Imagen Anterior*. Además, para hacer más sencilla la navegación, se imprimirán, debajo de estos botones, las coordenadas de la imagen mostrada dentro de la imagen global.

El número de piezas existente se almacena en una nueva variable (*pieces*) y las imágenes que se van obteniendo se guardan en la misma carpeta con la fecha a la que pertenecen y número de orden que siguen (de izquierda a derecha y de arriba a abajo).

Si la imagen es lo suficientemente pequeña para ser mostrada, se guarda sin ningún tipo de división.

Cuando se ha terminado el proceso se arroja otro aviso que informe de ello (figura A-18) y se llama a la función que decide qué imagen es la que se tiene que mostrar. Si la variable *pieces* es mayor que 1, se habilitan los botones adecuados (*Imagen Anterior* no se habilita si es la primera porción e *Imagen Siguiente* se deshabilita cuando se haya llegado a la última), se determina la ruta de la imagen y se especifican las coordenadas. Tras estos pasos la imagen está lista para ser mostrada. El método seguido para imprimir una imagen .bmp en una ventana con *scrollbars* es el explicado en [23].

Cada vez que se pulsa el botón *Imagen Siguiente*, se aumenta el contador que numera a las porciones y se llama de nuevo a esta función. Se sigue el mismo proceso con el botón *Imagen Anterior*, salvo que disminuyéndolo. Una ventaja de tener las imágenes guardadas previamente es que esta navegación es muy rápida.

A.2.7 Ventana principal: Botón "Fecha Siguiente"

Cuando el usuario se ha cansado de visualizar los resultados de la primera fecha, puede pulsar el botón *Fecha Siguiente*. Desde este punto se llama de nuevo al módulo "Examinar directorio" con el fin de determinar si existe fecha siguiente. Si hay éxito, la variable tabla se rellena con la nueva información y el programa se prepara para que vuelva a pulsarse el botón *Recalcular*.

FIGURA A-19: Existe fecha siguiente y la tabla se ha rellenado con nueva información

Si no había imágenes tomadas en una nueva fecha, se avisa de ello al usuario y se le permite pulsar *Recalcular* para seguir trabajando con las imágenes de la fecha anterior o pulsar *Nuevo Path* para examinar las imágenes de una nueva ruta.

FIGURA A-20: Aviso que informa de que en la carpeta no hay más imágenes que visualizar

B Wavelets

B.1 Introducción

B.1.1 Propiedades

Las wavelets son un conjunto de funciones que, juntas, son capaces de representar una señal. Más en concreto, se trata de filtros que permiten descomponer la señal en múltiples bandas.

Matemáticamente, son funciones que han de cumplir ciertas propiedades y que son generadas, mediante dilataciones y traslaciones, a partir de una única función. Uno de estos requerimientos es el que da nombre a las wavelets (u onditas), que tienen que cumplir que su dé como resultado 0, es decir, que deben ondear por encima y por debajo del eje x.

En general, la función base wavelet (a partir de la cual se generaría toda una familia de wavelets), debe cumplir estas tres propiedades:

- Debe ser posible expresar cualquier función en términos de wavelets, por lo que debe existir una serie que permita descomponer una función en términos de coeficientes y funciones wavelet base.
- Las wavelets deben tener localización espacial y frecuencial. Al descomponer una señal mediante series de Fourier, es posible detectar componentes frecuenciales presentes en la señal, pero no se puede situar una zona concreta de la señal. Mediante wavelets es posible descomponer una señal de manera que se puedan localizar componentes frecuenciales en cierta parte de la señal original.
- Las wavelets han de tener algoritmos rápidos, a fin de que puedan ser usadas en aplicaciones informáticas.

Estas tres propiedades hacen que las transformadas wavelets sean de utilidad en este proyecto, ya que permiten representar una señal (la imagen) de forma que se puedan analizar sus componentes frecuenciales (bordes, zonas uniformes…) en el lugar donde se encuentran originalmente y, además, realizar esta operación computacionalmente.

B.1.2 Funciones generadoras

Las wavelets se agrupan en familias de funciones que son generadas a partir de la dilatación y traslación de funciones base. En realidad, una familia se genera a partir de ϕ, función de escalado y ψ, función base o función wavelet. ϕ se encarga de proporcionar las frecuencias más bajas de la señal (es capaz de representar la señal con el menor número de componentes), mientras que ψ cumple la función de filtro paso-alto.

Una misma generación de wavelets tendrá el mismo nivel de escalado (o dilatación) respecto a la función base, y la única diferencia entre ellas será la situación en la que se encuentran sobre el eje x, es decir, serán todas iguales pero con distinta traslación. La

primera generación de wavelets "hijas" se representará como $\psi_{1,k}$, donde el primer subíndice será la generación o, lo que es lo mismo, el nivel de dilatación, y el segundo la traslación. De forma similar podríamos definir las distintas generaciones de hijos a partir de la función de escalado.

B.2 Wavelets de Haar

La wavelet de Haar es la más sencilla de todas las familias wavelets. Las funciones base y escalado que las generan se definen a continuación:

$$\psi(t) = \begin{cases} 1, & si\ 0 \le t < 0.5 \\ -1, & si\ 0.5 \le t < 1 \\ 0, & resto \end{cases} \qquad \phi(t) = \begin{cases} 1, si\ 0 \le t \le 1 \\ 0, resto \end{cases}$$

En la figura B-1 se puede ver la representación gráfica de ambas funciones.

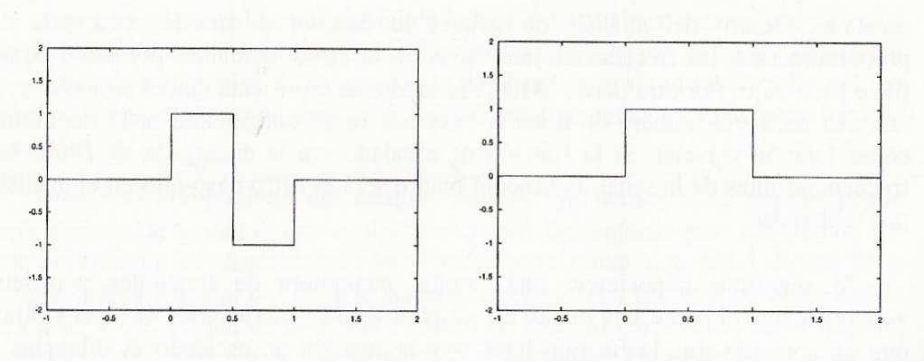


FIGURA B-1: Función Haar ψ(t) y función Haar ϕ(t)

Una de las propiedades más importantes de la transformada de Haar es que no sólo se expande en el intervalo 0-1, sino que sirve como base ortogonal a todas las funciones que se pueden definir en el eje de la recta real. Una segunda propiedad es que interesa que conserve la energía, es decir, que la suma del cuadrado de los valores iniciales sea igual a la suma del cuadrado de los coeficientes obtenidos con la nueva base. Si esto se cumple en la base ortogonal, podemos decir que la base es ortonormal. La versión oficial de la transformada de Haar es, por tanto:

$$\psi_{n,k}(t) = \sqrt{2^n}\psi(2^n t - k), \qquad 0 \le k \le 2^n - 1$$

B.3 Aplicación de wavelets a imágenes

La aplicación principal de las wavelets a imágenes es la compresión. Como las imágenes naturales tienen zonas bastante homogéneas, que presentan algunos cambios bruscos (bordes), esto las hace susceptibles de ser tratadas con wavelets. De hecho, el estándar para la compresión JPEG 2000, que actualiza JPEG (anexo D), usa wavelets en lugar de la DCT. Una característica de la wavelets es que se adaptan al nivel de detalle de la señal, gracias a que los coeficientes de frecuencias más altas cubren un menor espacio de la señal, lo que representa otro factor en favor de su uso en compresión de imágenes.

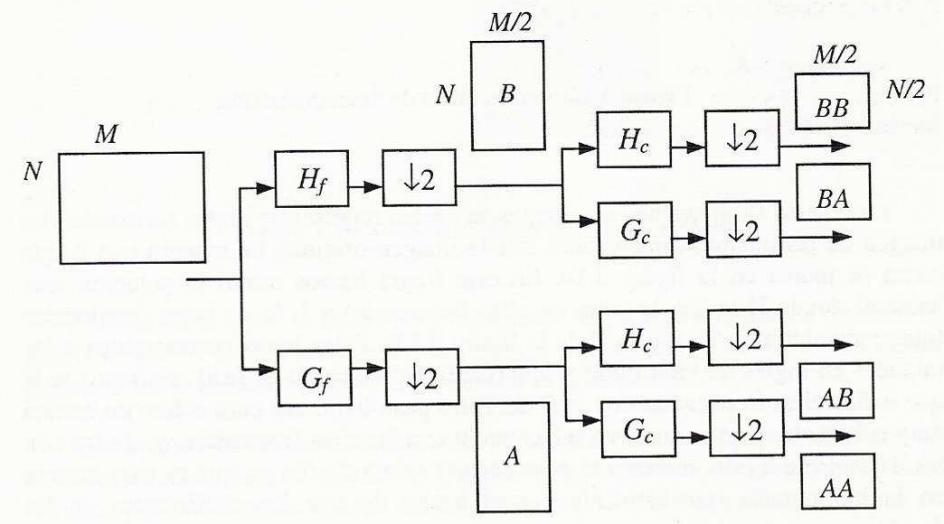


FIGURA B-2: Descomposición de una imagen en un nivel

En la figura B-2 se muestra cómo se realiza la descomposición de una imagen en un nivel. A partir de una imagen de tamaño MxN, se aplican un par de filtros paso-alto y paso-bajo, dando lugar a dos imágenes de dimensiones (M/2)xN, una con las componentes de baja frecuencia horizontales (B) y otras con los de altas frecuencias horizontales (A). Si se repite el mismo proceso sobre estas dos imágenes, pero ahora aplicando los filtros por columnas, se vuelven a obtener dos nuevas subimágenes por imagen, ahora de dimensiones (M/2)x(N/2), una con las componentes de baja frecuencia (BB), otra con las altas frecuencias verticales (BA), una más con las altas frecuencias horizontales (AB) y una última con las componentes de altas frecuencias diagonales (AA).

Seguidamente se muestra el resultado de aplicar la transformada Haar de primer nivel a una imagen, donde se aprecia cómo la transformada marca las altas frecuencias (los bordes) existentes en la imagen original, propiedad que la hace útil en este proyecto.

Imagen original **Transformada de Haar de primer nivel**

FIGURA B-3: Ejemplo de transformada de Haar

(La información de este anexo proviene de [26], pero si se desea profundizar más sobre la transformada de Haar o las wavelets, véanse [27] y [28].)

C Espacios de color

Este anexo sobre espacios de color no pretende ser más que un breve recordatorio sobre los modelos de color utilizados en el proyecto, y una explicación desde la perspectiva del porqué de su uso.

C.1 RGB

El modelo de color RGB (*Red, Green, Blue*) es un modelo basado en la síntesis aditiva, es decir, la representación de un color cualquiera se consigue a través de la suma, con distinta intensidad, de los colores rojo, verde y azul. Espacialmente, el modelo RGB se ve como un cubo 3D, donde los valores máximos de los ejes corresponderían a estos tres colores primarios: la ausencia de los tres equivaldría al negro y la colaboración de todos al blanco.

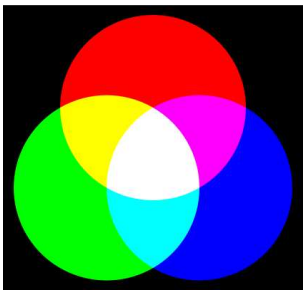


FIGURA C-1: Mezcla aditiva de colores

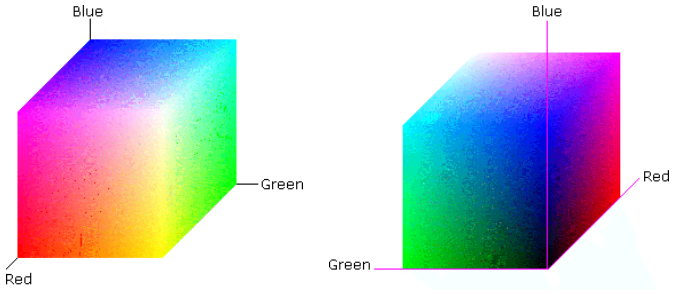


FIGURA C-2: Cubo RGB

De esta forma, cada color primario se moverá en un rango (0-1, 0-256…) realizando una aportación más o menos importante al color resultado. El rango más utilizado es el que va de 0 a 256, lo que permite que cada color primario se codifique con 1 byte (8 bits).

En este proyecto, la información contenida en las imágenes leídas con OpenCv se devuelve en un vector donde cada píxel se representa con una tríada RGB.

C.2 HSV

El modelo HSV (Tonalidad, Saturación, Valor) fue creado en 1978 por Alvy Ray Smith. Se trata de una transformación no lineal del espacio de color RGB, donde los colores se definen en términos de componentes en coordenadas cilíndricas.

La tonalidad (hue) especifica el tipo de color y se representa como un ángulo cuyos valores oscilan entre 0 y 360º. La saturación (saturation) se representa como la distancia desde el eje de brillo negro-blanco al lado del cilindro (radio) y su rango va del 0 al 100%. El valor (value), por último, simboliza el brillo del color y se representa como la altura en el eje negro-blanco. Su rango también es del 0 al 100%.

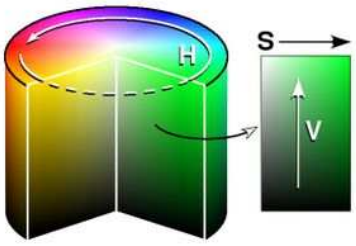


FIGURA C-3: Cilindro de colores del espacio HSV

Según [1], el modelo de color HSV es el más adecuado para realizar el cálculo de la entropía, razón por la cual se ha utilizado este espacio de color en el proyecto.

C.3 CIE LUV

El espacio de color CIE Luv es un modelo de color adoptado por la CIE (International Commission on Illumination) en 1976, con el fin de obtener una transformación sencilla del modelo CIE XYZ (1931) que permitiera obtener una representación de colores que fuera perceptivamente uniforme.

En el espacio XYZ, el modelo 2D fallaba porque no representaba adecuadamente el espacio que, originalmente, era 3D. En la figura C-5 cada línea representa la misma diferencia entre colores pero, como se puede observar, perceptivamente las líneas tienen diferentes longitudes, dependiendo de la zona en la que se hallen.

Para solucionar este problema, la CIE desarrolló el modelo Luv. En la figura C-7 se puede apreciar que, aunque la representación no es perfecta, el diagrama presenta una innegable mejoría en cuanto a uniformidad.

El espacio CIE Luv se utiliza en este proyecto para medir, mediante la distancia euclídea, la similitud entre colores, imprescindible en la detección de duplicados.

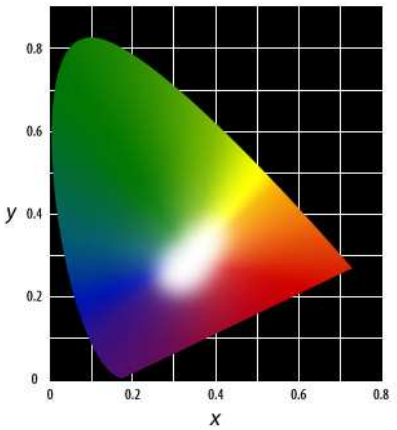


FIGURA C-4: Diagrama CIE XYZ

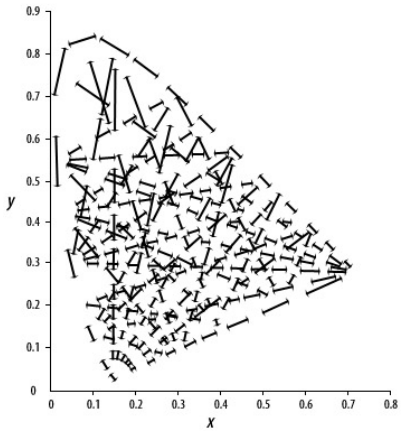


FIGURA C-5: Distancias entre colores (XYZ)

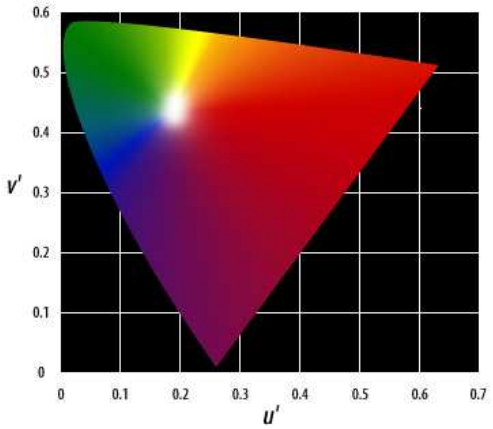


FIGURA C-6: Diagrama CIE LUV

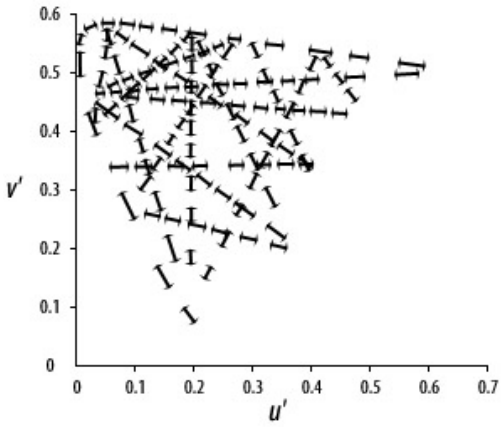


FIGURA C-7: Distancias entre colores (LUV)

C.4 YCrCb

El espacio de color YCrCb es muy conocido en aplicaciones digitales de vídeo y fotografía. Y se corresponde con la componente de luminancia, mientras que Cr y Cb se identifican como las diferencias roja y azul de las crominancias.

Este espacio sirve para "codificar" el espacio RGB, que no es eficiente a la hora de transmitir o almacenar imágenes o vídeos, ya que las componentes roja, verde y azul contienen una apreciable redundancia.

Este modelo de color se utiliza en compresión de imágenes y de vídeo, y es el modelo recomendado por MPEG [13] para implementar el algoritmo *Color Layout*, razón por la que se utiliza en este proyecto.

C.5 YUV

El modelo de color YUV define un espacio de color en términos de una componente de luminancia y dos componentes de crominancia. Está basado en los sistemas PAL Y NTSC de difusión de televisión.

(La información consultada para redactar este anexo proviene de [31], [32], [33], [34], [35], [36], [37] y [38], aunque existen múltiples fuentes sobre la colorimetría tan completas como [39].)

D JPEG

D.1 Introducción

Aunque existen muchos sistemas de compresión de imágenes, el estándar JPEG, adoptado por el Joint Photographic Experts Group, es el estándar más utilizado para comprimir imágenes. A pesar de realizar compresiones con pérdidas, JPEG consigue calidades desde buenas a excelentes lo que, junto a su sencillez a la hora de ser implementado y su versatilidad (se aplica tanto a imágenes en escala de grises como en color), lo hace un sistema muy potente.

D.2 Codificación

JPEG, como técnica, se engloba en las de "codificación de la transformada", es decir, aquellas que no comprimen la señal en sí, sino su transformada. La transformada más utilizada en codificación de imagen es la DCT. Los beneficios más directos de esta transformada son sus propiedades de compresión de la energía de la imagen y la existencia de algoritmos rápidos que ejecuten la transformada computacionalmente. La propiedad de la DCT de compactación de la energía la imagen resulta en la obtención de unos coeficientes que, en su mayoría, no tienen valores significativos, con lo que casi toda la energía queda contenida en los coeficientes que sí tienen valor.

La DCT de una imagen NxN con luminancia $x(m,n)$, $0 \le m, n \le N$-1 se calcula a partir de estas ecuaciones:

$$X(0,0) = \frac{1}{N}\sum_{k=0}^{N-1}\sum_{l=0}^{N-1} x(k,l) \qquad X(m,n) = \frac{2}{N}\sum_{k=0}^{N-1}\sum_{l=0}^{N-1} x(k,l)\cos\left(\frac{(2k+1)u\pi}{2N}\right)\cos\left(\frac{(2l+1)v\pi}{2N}\right) \quad u,v \neq 0$$

El coeficiente $X(0,0)$ es el denominado DC, mientras que el resto se denominan coeficientes AC.

El codificador JPEG se compone de 3 bloques: bloque DCT, cuantificador y codificador entrópico.

D.2.1 Bloque DCT

Una imagen no es más que un conjunto de píxeles organizados en una matriz de tamaño mxn. El primer paso para realizar la transformada DCT a la imagen es dividirla en bloques de tamaño 8x8 (tamaño escogido para conseguir un equilibrio entre complejidad y calidad). Si el número de filas o columnas de la imagen no es divisible entre 8, entonces se repite la última fila o columna hasta que lo sea. (Estos píxeles adicionales se retirarán en el decodificador.)

Después de obtener los bloques 8x8, se les aplica a cada uno de ellos la DCT. Este proceso genera 64 coeficientes DCT por cada bloque 8x8, un coeficiente DC y 63 coeficientes AC.

D.2.2 Cuantificador

Debido a la propiedad de compactación de la energía de la DCT, sólo los coeficientes de baja frecuencia tienen valores significativos.

Como el coeficiente DC es el de mayor energía, y además existe una gran correlación entre la componente DC de un bloque y la del que le precede, la cuantificación de los DC se realiza utilizando un esquema de cuantificación diferencial. Los coeficientes AC, sin embargo, se cuantifican mediante sistemas de cuantificación uniforme.

El bloque que resulta después de la cuantificación es una matriz 8x8 cuya mayoría de valores es cero, salvo en la esquina superior izquierda.

Después del proceso de cuantificación, los coeficientes DCT son ordenados en un vector siguiendo un zigzag, lo que permite obtener un vector X de longitud 64 con valores únicamente en las primeras componentes.

D.2.3 Codificador entrópico

El paso de la cuantificación es el que provoca que la compresión que realiza JPEG sea con pérdidas. Tras ello, el vector resultante se comprime mediante codificación entrópica, con el fin de no introducir más pérdidas. Esta codificación aprovecha el amplio número de ceros para reducir el tamaño de la información del bloque.

D.3 Decodificación

Para descomprimir una imagen JPEG, se siguen los pasos inversos a los vistos en el apartado anterior. La transformada inversa de la DCT se denomina IDCT, y es justo antes de su aplicación cuando se deberían recoger la imagen dividida en bloques 8x8 para su utilización en los algoritmos a nivel de bloque que se estudian en el proyecto.

(La información de este anexo proviene de [10], pero la información más completa sobre JPEG se encuentra en [29] y, para saber más sobre el futuro de la compresión de imágenes, [30].)

E MPEG

MPEG es el nombre de una familia de estándares creados para la codificación de información audiovisual (películas, vídeos, música) en un formato digital.

E.1 MPEG-1

MPEG-1 nace con la pretensión ser un estándar genérico de almacenamiento de vídeo en CDROM. El esquema de codificación que utiliza MPEG-1: Parte 2 está basado en un esquema que realiza predicción en el dominio de la imagen (espacial) y que aplica la transformación sobre el error de predicción.

Los pasos que sigue el codificador MPEG-1 comienzan con la división de la imagen YCrCb (anexo C) en bloques. Antes de codificar la información, se calcula una predicción de cada bloque. A partir de éste se genera el bloque de error de predicción (diferencia entre el original y el predicho) y, a continuación, se aplica la DCT. Los bloques transformados se cuantifican y se comprimen mediante codificación entrópica antes de ser transmitidos o almacenados.

La predicción del bloque se consigue mediante estimación de movimiento y compensación del movimiento. Estos dos pasos persiguen el objetivo de conseguir un bloque de predicción lo más parecido posible al bloque original, aunque el objeto del que forme parte el bloque se haya desplazo por la imagen. De esta forma, el bloque de error de predicción al que se aplicará la transformada contendrá los valores más pequeños posibles, obteniendo una mayor compresión.

Este proceso de predicción, sin embargo, no se aplica de la misma forma a todos los *frames*. Existen tres tipos de *frames* en MPEG: I, P y B. Los *frames* I (los que se utilizan en la aplicación de vídeo de este proyecto) pueden ser decodificados independientemente de otros *frames*, como si se tratara de una imagen JPEG (anexo D). Los *frames* P y B, sin embargo, explotan la característica temporal de los vídeos, de manera que un *frame* tipo P sólo contendrá la información diferencia entre éste y su predecesor (calculada mediante la estimación y compensación de movimiento), mientras que un *frame* tipo B también obtendrá información del *frame* posterior.

E.2 MPEG-2

MPEG-2: Parte 2 utiliza el mismo esquema de codificación que MPEG-1 pero, al estar enfocado a la transmisión y almacenamiento de vídeo para televisión digital, es más complejo y más flexible que su predecesor.

La razón de la utilización de los estándares MPEG en este proyecto proviene del deseo de integrar la aplicación de vídeo en IVOnLA [43], programa que decodifica los vídeos MPEG y obtiene para la aplicación los valores de los bloques DCT.

(El contenido de este anexo está basado en [26] y en [40], pero la información sobre el estándar MPEG es muy amplia, empezando por [41] y [42].)

F Etiquetas EXIF

En este anexo se describen las cabeceras EXIF en detalle, con el fin de explicar cómo los programas implementados realizan el acceso al campo de fecha y hora de una imagen, que se complementa con un ejemplo de extracción de este campo de una imagen JPEG.

F.1 Localización de los campos EXIF

Los marcadores JPEG cuyo identificador está entre las direcciones 0xFFE0 y 0xFFEF se denominan *Application Markers*. Estos marcadores no son necesarios para la decodificación, sino que son utilizados por aplicaciones de usuario con diferentes fines (guardar imágenes, insertar datos de configuración de la cámara digital y miniaturas de las imágenes…).

Para evitar conflictos con el formato JFIF (que hace uso del *application marker* APP0 (0xFFE0)) el que utiliza EXIF es el APP1 (0xFFE1). Todas las imágenes con campos EXIF empiezan con el SOI *Marker* (como toda imagen JPEG) e inmediatamente a continuación tienen el APP1 *Marker*.

SOI Marker	**APP1 Marker**	**Datos APP1**	**Otros markers …**
FF D8	FF E1	SSSS 457869660000 TTT…	FFXX UUUU DDD…

TABLA F-1: Cabecera JPEG

El campo SSSS indica el tamaño en bytes de TTT…, más los 2 bytes que ocupa el campo SSSS en sí mismo, más el tamaño de la cabecera EXIF, que se corresponde con el ASCII de estas mismas letras (‘E’ = 0x45, ‘X’ = 0x78, ‘I’ = 0x69, ‘F’ = 0x66) y de 0x00 dos veces al final.

F.2 Formato TIFF

Los datos EXIF están almacenados en el formato TIFF, que archiva información adicional mediante etiquetas y cuya última versión es la 6.0.1 [21]. Según este formato, los datos se estructuran dividiéndose en IFD (*Image File Directory*) que contienen a su vez información sobre la imagen en las “entradas de directorio” y punteros a los datos de la imagen en sí. En un archivo TIFF ha de haber como mínimo un IFD con una entrada.

A continuación se ilustra cómo se distribuyen los datos según el formato TIFF:

Estructura de la cabecera TIFF (8 Bytes)

Dir.	Valor	Significado
0	'II' o 'MM'	Orden de los bytes (II = Little Endian, MM = Big Endian)
2	2A 00 / 00 2A	Número que identifica el archivo como archivo TIFF (42 = 2A H)
4	0A 00 / 00 00	*Offset* del primer directorio (IFD0)
6	00 00 / 0A 00	

Estructura del primer directorio (IFD0)

Dirección	Valor				Significado
A	0B 00 / 00 0B				Nº total de entradas de directorio
A+2	TT TT	ff ff	NN NN NN NN	DD DD DD DD	Entrada de directorio 0
A+14	TT TT	ff ff	NN NN NN NN	DD DD DD DD	Entrada de directorio 1
	…				
A+2+B·12	LL LL LL LL				*Offset* del 2º directorio (IFD1)

Donde:

- TTTT: tipo de campo. Esta etiqueta especifica qué tipo de información es la que corresponde a esta entrada.
- ffff: formato de los datos. (1 = BYTE, 2 = ASCII, 3 = UNSIGNED SHORT, 4 = UNSIGNED LONG, 5 = UNSIGNED RATIONAL…)
- NN NN NN NN: número de componentes que forman los datos en esta entrada.
- DD DD DD DD: valor del dato u *offset* al valor. Si el número de componentes multiplicado por el tamaño del tipo de dato (BYTE = 1 byte, ASCII = 1 byte, etc.) es menor que 4 bytes (tamaño del campo DD DD DD DD), la información de la entrada se encuentra en DD DD DD DD. Si es mayor, los datos se encuentran en una zona de memoria cuya dirección es DD DD DD DD.

F.3 Estructura de los datos EXIF

Cuando los datos EXIF se encuentran presentes en una imagen JPEG, sólo es posible la existencia de dos IFD (IFD0 e IFD1). A su vez, los datos EXIF pueden ser almacenados en una misma imagen de alguna o de dos de estas formas: como entrada del directorio IFD0 o como IFD adicional.

En la segunda opción, la última entrada de IFD0 tiene un tipo de campo (TTTT) especial: EXIF *offset* (0x8769). Esta entrada de directorio contiene, en su campo de datos, la dirección de memoria del IFD *EXIFSubIFD*. A partir de esta dirección es donde se almacenan los campos EXIF siguiendo el mismo esquema que las entradas de directorio del formato TIFF.

Los datos que se pueden almacenar en caberas EXIF son muy variados y todos ellos se encuentran listados en [22]. El campo de fecha y hora tiene dos *tags* diferentes, dependiendo de en qué zona de la cabecera está guardado el dato. Si se encuentra en una

entrada del directorio IFD0, su *tag* será 0x0132; si se aparece en *EXIFSubIFD*, su *tag* será 0x9003.

Para comprender mejor cómo se estructuran los datos en una cabecera JPEG con campos EXIF se imprimen en el ejemplo siguiente los valores de los campos, su significado y las posiciones de memoria en las que se encuentran. El dato de fecha y hora se puede localizar tanto en una entrada de directorio como en *EXIFSubIFD*:

FIGURA F-1: Imagen de ejemplo para la extracción de los campos EXIF

En el primer paso se recogen la longitud de los datos de APP1, el orden de los bytes ('II' o 'MM') y la dirección de comienzo de IFD0.

						0	2	4	6
FF D8	**FF E1**	**14 62**	**45 78**	**69 66**	**00 00**	**49 49**	**2A 00**	**08 00**	**00 00**
SOI Marker	APP1 Marker	Longitud 5218 Bytes	E X Cabecera EXIF	I F	0 0	'I I' Little Endian	Identi-ficador TIFF	Dir. de IFD0	
						Cabecera TIFF			
		APP1 Data							

Tras trasladarse a la dirección de comienzo de IFD0, se extrae la información de las entradas:

8	10	12	14	16	18	20
09 00	**0F 01**	**02 00**	**06 00**	**00 00**	**7A 00 00 00**	
Número de entradas	**MAKE**	ASCII 1B/comp	Número de componentes		OFFSET = 122	
IFD0 Data						
APP1 Data						

La primera entrada tiene información sobre el *tag* MAKE (fabricante del equipo). Como el número de componentes (6) multiplicado por el tamaño del tipo de dato (ASCII, 1 byte por componente) es mayor que 4 bytes, el campo de datos se convierte en la dirección de memoria donde se encuentra la información.

22	24	26	28	30	32	34	36	38	40	42	44
10 01	**02 00**	**14 00 00 00**		**80 00 00 00**		**12 01**	**03 00**	**01 00 00 00**		**01 00 00 00**	
MODEL	ASCII	20 comp		OFFSET = 128		**ORIENTATION**	UShort 2B/comp	1 comp		Dato	
IFD0 Data											
APP1 Data											

La segunda entrada informa sobre el modelo de equipo utilizado para tomar la foto, que se encuentra en la posición de memoria 128. La tercera entrada indica qué orientación tiene la imagen. Como el dato es 1, indica que la fila 0 de la imagen se encuentra arriba y que la columna 0 es la de la izquierda.

46	48	50	52	54	56	58	60	62	64	66	68
1A 01	**05 00**	**01 00 00 00**		**94 00 00 00**		**1B 01**	**05 00**	**01 00 00 00**		**9C 00 00 00**	
XRES	URational 8B/comp	1 comp		OFFSET = 148		**YRES**	URational	1 comp		OFFSET = 156	
IFD0 Data											
APP1 Data											

Ambas entradas notifican sobre la resolución de la imagen (píxeles por unidad), una en dirección horizontal y otra en vertical. Los datos se encuentran en las posiciones 148 y 156.

70	72	74	76	78	80	82	84	86	88	90	92
28 01	**03 00**	**01 00 00 00**		**02 00 00 00**		**32 01**	**02 00**	**14 00 00 00**		**A4 00 00 00**	
RES UNIT	UShort	1 comp		Dato		**DATE TIME**	ASCII	20 comp		OFFSET = 164	
IFD0 Data											
APP1 Data											

La sexta entrada contiene un dato acerca de la unidad en la que se miden las resoluciones anteriores. Como vale 2, la unidad es la pulgada. La séptima entrada incluye un offset a la zona de memoria donde se encuentran la fecha y la hora en las que se tomó la imagen.

94	96	98	100	102	104	106	108	110	112	114	116
13 02	**03 00**	**01 00 00 00**		**01 00 00 00**		**69 87**	**04 00**	**01 00 00 00**		**B8 00 00 00**	
YCRCB POS	UShort	1 comp		Dato		**EXIF OFFSET**	ULong			EXIF OFFSET = 184	
IFD0 Data											
APP1 Data											

La entrada número 8 encierra el dato sobre la posición de las componentes de luminancia con respecto a las de crominancia. Su valor a 1 indica que las Y están centradas entre dos C. La última entrada no es más que el EXIF *offset*, es decir, la posición de memoria del *EXIFSubIFD* donde se encuentran el resto de entradas.

Las dos siguientes direcciones (118 y 120) que aparecen antes de que comience la primera información de la entrada 1 (dirección 122) se rellena con el *offset* de IFD1.

122	124	126	128	130	132	134	136	138	140	142	144	146
C a	n o	n '\0'	C a	n o	n _	D I	G I	T A	L _	I X	U S	0 0
Fabricante equipo			Modelo de equipo									
IFD0 Data												
APP1 Data												

148	150	152	154	156	158	160	162
B4 00	00 00	01 00	00 00	B4 00	00 00	01 00	00 00
180 píxeles por cada pulgada				180 píxeles por cada pulgada			
IFD0							
APP1							

164	166	168	170	172	174	176	178	180	182
2 0	0 1	: 0	6 :	0 9	_ 1	5 :	1 7	: 3	2 '\0'
Fecha: 9-6-2001					Hora: 15:17 y 32 seg				
IFD0									
APP1									

La siguiente posición de memoria (184) se corresponde con el EXIF *offset*. Siguiendo con la misma estructura TIFF, *EXIFSubIFD* cuenta con 27 entradas, una de ellas con la fecha y la hora de la generación de los datos. Tras localizar su *tag* (0x0390), se accede a su *offset* (posición 514). La fecha y la hora guardadas en este campo es la misma que la anteriormente leída en la entrada de directorio.

F.4 Extracción de la fecha y la hora

Para obtener el campo de fecha y hora de las cabeceras EXIF, las aplicaciones buscan, primeramente, etiquetas como SOI o APP1, con el fin de extraer los datos en relación al tamaño de APP1. Estableciendo esta longitud como límite, se analiza qué orden siguen los datos (*Little Endian* o *Big Endian*) y se busca, según este orden, los *tags* 0x3201 o 0x0132 o 0x0390 o 0x9003. Una vez localizados, se comprueba que corresponden a *tags* y no a cualquier parte de las cabeceras mirando el formato de los datos (ha de ser ASCII) y el número de componentes (forzosamente 20). Si estos ocho bytes coinciden, se almacena el *offset* y, continuando con el recorrido de la imagen, se copia el campo fecha cuando se alcance la correspondiente posición de memoria.

G Base de datos de imágenes de entrenamiento

La base de datos de imágenes de entrenamiento cuenta con tres fuentes: Internet, fotografías de la base de datos de la Universidad de Corea (http://cist.korea.ac.kr/~multimedia/data/index.htm) e imágenes con campos EXIF (http://www.exif.org/samples.html).

A continuación se lista una relación de las imágenes contenidas en cada una de estas fuentes que han sido utilizadas para establecer los umbrales y pesos en los algoritmos que se describen en este proyecto.

Base de datos de la Universidad de Corea Globally Sharp			
	Miniatura	Identificador	Tamaño (píxeles)
1		sharp(1).jpg	2048 x 1536
2		sharp(2).jpg	2048 x 1536
3		sharp(3).jpg	1280 x 960
4		sharp(4).jpg	1280 x 960
5		sharp(5).jpg	2048 x 1536
6		sharp(6).jpg	2048 x 1536
7		sharp(7).jpg	2048 x 1536
8		sharp(8).jpg	2048 x 1536
9		sharp(9).jpg	2592 x 1944
10		sharp(10).jpg	2048 x 1536
11		sharp(11).jpg	2048 x 1536
12		sharp(12).jpg	2048 x 1536

13		sharp(13).jpg	2048 x 1536
14		sharp(14).jpg	2048 x 1536
15		sharp(15).jpg	1600 x 1200
16		sharp(16).jpg	1600 x 1200
17		sharp(17).jpg	1600 x 1200
18		sharp(18).jpg	1600 x 1200
19		sharp(19).jpg	1600 x 1200
20		sharp(20).jpg	1600 x 1200

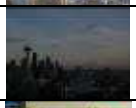
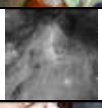
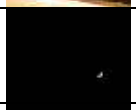
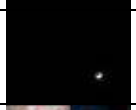
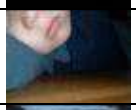
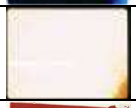
Base de datos de la Universidad de Corea **Average Quality**			
	Miniatura	Identificador	Tamaño (píxeles)
1		average(1).jpg	2048 x 1536
2		average(2).jpg	2048 x 1536
3		average(3).jpg	2048 x 1536
4		average(4).jpg	2048 x 1536
5		average(5).jpg	2048 x 1536
6		average(6).jpg	2048 x 1536
7		average(7).jpg	2048 x 1536
8		average(8).jpg	2048 x 1536

9		average(9).jpg	2048 x 1536
10		average(10).jpg	2048 x 1536
11		average(11).jpg	2592 x 1944
12		average(12).jpg	2592 x 1944
13		average(13).jpg	2592 x 1944
14		average(14).jpg	2592 x 1944
15		average(15).jpg	2592 x 1944
16		average(16).jpg	2592 x 1944
17		average(17).jpg	2592 x 1944
18		average(18).jpg	2592 x 1944
19		average(19).jpg	2048 x 1536
20		average(20).jpg	2048 x 1536

Base de datos de la Universidad de Corea Globally blurry			
	Miniatura	Identificador	Tamaño (píxeles)
1		blurry(1).jpg	1280 x 960
2		blurry(2).jpg	1280 x 960
3		blurry(3).jpg	1280 x 960
4		blurry(4).jpg	2048 x 1536

5		blurry(5).jpg	2048 x 1536
6		blurry(6).jpg	2048 x 1536
7		blurry(7).jpg	2048 x 1536
8		blurry(8).jpg	2048 x 1536
9		blurry(9).jpg	2048 x 1536
10		blurry(10).jpg	2048 x 1536
11		blurry(11).jpg	2048 x 1536
12		blurry(12).jpg	2048 x 1536
13		blurry(13).jpg	2048 x 1536
14		blurry(14).jpg	2048 x 1536
15		blurry(15).jpg	2048 x 1536
16		blurry(16).jpg	2048 x 1536
17		blurry(17).jpg	2048 x 1536
18		blurry(18).jpg	2048 x 1536
19		blurry(19).jpg	2048 x 1536
20		blurry(20).jpg	2048 x 1536

Imágenes con campos EXIF			
	Miniatura	Identificador	Tamaño (píxeles)
1		canon-ixus.jpg	640 x 480
2		fujifilm-dx10.jpg	1024 x 768
3		fujifilm-finepix40i.jpg	600 x 450
4		fujifilm-mx1700.jpg	640 x 480
5		JFIFolympus-d320l.jpg	640 x 480
6		JFIFsony-powershota5.jpg	1024 x 768
7		kodak-dc210.jpg	640 x 480
8		kodak-dc240.jpg	640 x 480
9		nikon-e950.jpg	800 x 600
10		olympus-c960.jpg	640 x 480
11		ricoh-rdc5300.jpg	896 x 600
12		sanyo-vpcg250.jpg	640 x 480
13		sanyo-vpcsx550.jpg	640 x 480
14		sony-cybershot.jpg	640 x 480
15		sony-d700.jpg	672 x 512

Imágenes de Internet			
	Miniatura	Identificador	Tamaño (píxeles)
1		Bracketing-Normal.Exposition.jpg	1333 x 1000
2		Bracketing-Sobreexposition.jpg	1333 x 1000
3		Bracketing-Subexposition.jpg	1333 x 1000
4		muneca.jpg	1136 x 852
5		ruidosa.jpg	891 x 892
6		Gato.jpg	1600 x 1200
7		Gente.jpg	1600 x 1200
8		Latas.jpg	1600 x 1200
9		Luces.jpg	1600 x 1200
10		moon.jpg	1600 x 1200
11		moon2.jpg	1600 x 1200
12		nino.jpg	1600 x 1200
13		playa.jpg	1600 x 1200
14		terror.jpg	1600 x 1200
15		wallpaper.jpg	1600 x 1200
16		wallpaper2.jpg	1600 x 1200
17		colores.jpg	256 x 192

18		colores2.jpg	256 x 192
19		grupo.jpg	1600 x 1200
20		grupo2.jpg	1600 x 1200
21		pez.jpg	400 x 266
22		pez2.jpg	400 x 266
23		paisaje_rural.jpg	500 x 375
24		Bajo1.jpg	590 x 395
25		Bajo2.jpg	800 x 1067
26		Bajo3.jpg	750 x 498
27		Sobre1.jpg	600 x 403
28		Sobre2.jpg	400 x 300
29		Sobre3.jpg	600 x 450
30		palmera.jpg	2592 x 1944

PRESUPUESTO

1) Ejecución Material

- Compra de ordenador personal (Software incluido)..... ..... 2.000 €
- Material de oficina ..... 150 €
- Total de ejecución material ..... 2.150 €

2) Gastos generales

- 16 % sobre Ejecución Material ..... 344 €

3) Beneficio Industrial

- 6 % sobre Ejecución Material ..... 129 €

4) Honorarios Proyecto

- 960 horas a 15 € / hora ..... 14400 €

5) Material fungible

- Gastos de impresión ..... 200 €
- Encuadernación ..... 60 €

6) Subtotal del presupuesto

- Subtotal Presupuesto ..... 16810 €

7) I.V.A. aplicable

- 16% Subtotal Presupuesto ..... 2689.6 €

8) Total presupuesto

- Total Presupuesto ..... 19499.6 €

Madrid, Diciembre de 2008

El Ingeniero Jefe de Proyecto

Fdo.: Irene Blasco Hernanz
Ingeniero Superior de Telecomunicación

PLIEGO DE CONDICIONES

Este documento contiene las condiciones legales que guiarán la realización, en este proyecto, de algoritmos de detección de imágenes ruidosas y duplicadas. En lo que sigue, se supondrá que el proyecto ha sido encargado por una empresa cliente a una empresa consultora con la finalidad de realizar dicho sistema. Dicha empresa ha debido desarrollar una línea de investigación con objeto de elaborar el proyecto. Esta línea de investigación, junto con el posterior desarrollo de los programas está amparada por las condiciones particulares del siguiente pliego.

Supuesto que la utilización industrial de los métodos recogidos en el presente proyecto ha sido decidida por parte de la empresa cliente o de otras, la obra a realizar se regulará por las siguientes:

Condiciones generales

1. La modalidad de contratación será el concurso. La adjudicación se hará, por tanto, a la proposición más favorable sin atender exclusivamente al valor económico, dependiendo de las mayores garantías ofrecidas. La empresa que somete el proyecto a concurso se reserva el derecho a declararlo desierto.

2. El montaje y mecanización completa de los equipos que intervengan será realizado totalmente por la empresa licitadora.

3. En la oferta, se hará constar el precio total por el que se compromete a realizar la obra y el tanto por ciento de baja que supone este precio en relación con un importe límite si este se hubiera fijado.

4. La obra se realizará bajo la dirección técnica de un Ingeniero Superior de Telecomunicación, auxiliado por el número de Ingenieros Técnicos y Programadores que se estime preciso para el desarrollo de la misma.

5. Aparte del Ingeniero Director, el contratista tendrá derecho a contratar al resto del personal, pudiendo ceder esta prerrogativa a favor del Ingeniero Director, quien no estará obligado a aceptarla.

6. El contratista tiene derecho a sacar copias a su costa de los planos, pliego de condiciones y presupuestos. El Ingeniero autor del proyecto autorizará con su firma las copias solicitadas por el contratista después de confrontarlas.

7. Se abonará al contratista la obra que realmente ejecute con sujeción al proyecto que sirvió de base para la contratación, a las modificaciones autorizadas por la superioridad o a las órdenes que con arreglo a sus facultades le hayan comunicado por escrito al Ingeniero Director de obras siempre que dicha obra se haya ajustado a los preceptos de los pliegos de condiciones, con arreglo a los cuales, se harán las modificaciones y la valoración de las diversas unidades sin que el importe total pueda exceder de los presupuestos aprobados. Por consiguiente, el número de unidades que se consignan en el proyecto o en el presupuesto, no podrá servirle de fundamento para entablar reclamaciones de ninguna clase, salvo en los casos de rescisión.

8. Tanto en las certificaciones de obras como en la liquidación final, se abonarán los trabajos realizados por el contratista a los precios de ejecución material que figuran en el presupuesto para cada unidad de la obra.

9. Si excepcionalmente se hubiera ejecutado algún trabajo que no se ajustase a las condiciones de la contrata pero que sin embargo es admisible a juicio del Ingeniero Director de obras, se dará conocimiento a la Dirección, proponiendo a la vez la rebaja de precios que el Ingeniero estime justa y si la Dirección resolviera aceptar la obra, quedará el contratista obligado a conformarse con la rebaja acordada.

10. Cuando se juzgue necesario emplear materiales o ejecutar obras que no figuren en el presupuesto de la contrata, se evaluará su importe a los precios asignados a otras obras o materiales análogos si los hubiere y cuando no, se discutirán entre el Ingeniero Director y el contratista, sometiéndolos a la aprobación de la Dirección. Los nuevos precios convenidos por uno u otro procedimiento, se sujetarán siempre al establecido en el punto anterior.

11. Cuando el contratista, con autorización del Ingeniero Director de obras, emplee materiales de calidad más elevada o de mayores dimensiones de lo estipulado en el proyecto, o sustituya una clase de fabricación por otra que tenga asignado mayor precio o ejecute con mayores dimensiones cualquier otra parte de las obras, o en general, introduzca en ellas cualquier modificación que sea beneficiosa a juicio del Ingeniero Director de obras, no tendrá derecho sin embargo, sino a lo que le correspondería si hubiera realizado la obra con estricta sujeción a lo proyectado y contratado.

12. Las cantidades calculadas para obras accesorias, aunque figuren por partida alzada en el presupuesto final (general), no serán abonadas sino a los precios de la contrata, según las condiciones de la misma y los proyectos particulares que para ellas se formen, o en su defecto, por lo que resulte de su medición final.

13. El contratista queda obligado a abonar al Ingeniero autor del proyecto y director de obras así como a los Ingenieros Técnicos, el importe de sus respectivos honorarios facultativos por formación del proyecto, dirección técnica y administración en su caso, con arreglo a las tarifas y honorarios vigentes.

14. Concluida la ejecución de la obra, será reconocida por el Ingeniero Director que a tal efecto designe la empresa.

15. La garantía definitiva será del 4% del presupuesto y la provisional del 2%.

16. La forma de pago será por certificaciones mensuales de la obra ejecutada, de acuerdo con los precios del presupuesto, deducida la baja si la hubiera.

17. La fecha de comienzo de las obras será a partir de los 15 días naturales del replanteo oficial de las mismas y la definitiva, al año de haber ejecutado la provisional, procediéndose si no existe reclamación alguna, a la reclamación de la fianza.

18. Si el contratista al efectuar el replanteo, observase algún error en el proyecto, deberá comunicarlo en el plazo de quince días al Ingeniero Director de obras, pues transcurrido ese plazo será responsable de la exactitud del proyecto.

19. El contratista está obligado a designar una persona responsable que se entenderá con el Ingeniero Director de obras, o con el delegado que éste designe, para todo relacionado con ella. Al ser el Ingeniero Director de obras el que interpreta el proyecto, el contratista deberá consultarle cualquier duda que surja en su realización.

20. Durante la realización de la obra, se girarán visitas de inspección por personal facultativo de la empresa cliente, para hacer las comprobaciones que se crean oportunas. Es obligación del contratista, la conservación de la obra ya ejecutada hasta la recepción de la misma,

por lo que el deterioro parcial o total de ella, aunque sea por agentes atmosféricos u otras causas, deberá ser reparado o reconstruido por su cuenta.

21. El contratista, deberá realizar la obra en el plazo mencionado a partir de la fecha del contrato, incurriendo en multa, por retraso de la ejecución siempre que éste no sea debido a causas de fuerza mayor. A la terminación de la obra, se hará una recepción provisional previo reconocimiento y examen por la dirección técnica, el depositario de efectos, el interventor y el jefe de servicio o un representante, estampando su conformidad el contratista.

22. Hecha la recepción provisional, se certificará al contratista el resto de la obra, reservándose la administración el importe de los gastos de conservación de la misma hasta su recepción definitiva y la fianza durante el tiempo señalado como plazo de garantía. La recepción definitiva se hará en las mismas condiciones que la provisional, extendiéndose el acta correspondiente. El Director Técnico propondrá a la Junta Económica la devolución de la fianza al contratista de acuerdo con las condiciones económicas legales establecidas.

23. Las tarifas para la determinación de honorarios, reguladas por orden de la Presidencia del Gobierno el 19 de Octubre de 1961, se aplicarán sobre el denominado en la actualidad "Presupuesto de Ejecución de Contrata" y anteriormente llamado "Presupuesto de Ejecución Material" que hoy designa otro concepto.

Condiciones particulares

La empresa consultora, que ha desarrollado el presente proyecto, lo entregará a la empresa cliente bajo las condiciones generales ya formuladas, debiendo añadirse las siguientes condiciones particulares:

1. La propiedad intelectual de los procesos descritos y analizados en el presente trabajo, pertenece por entero a la empresa consultora representada por el Ingeniero Director del Proyecto.

2. La empresa consultora se reserva el derecho a la utilización total o parcial de los resultados de la investigación realizada para desarrollar el siguiente proyecto, bien para su publicación o bien para su uso en trabajos o proyectos posteriores, para la misma empresa cliente o para otra.

3. Cualquier tipo de reproducción aparte de las reseñadas en las condiciones generales, bien sea para uso particular de la empresa cliente, o para cualquier otra aplicación, contará con autorización expresa y por escrito del Ingeniero Director del Proyecto, que actuará en representación de la empresa consultora.

4. En la autorización se ha de hacer constar la aplicación a que se destinan sus reproducciones así como su cantidad.

5. En todas las reproducciones se indicará su procedencia, explicitando el nombre del proyecto, nombre del Ingeniero Director y de la empresa consultora.

6. Si el proyecto pasa la etapa de desarrollo, cualquier modificación que se realice sobre él, deberá ser notificada al Ingeniero Director del Proyecto y a criterio de éste, la empresa consultora decidirá aceptar o no la modificación propuesta.

7. Si la modificación se acepta, la empresa consultora se hará responsable al mismo nivel que el proyecto inicial del que resulta el añadirla.

8. Si la modificación no es aceptada, por el contrario, la empresa consultora declinará toda responsabilidad que se derive de la aplicación o influencia de la misma.

9. Si la empresa cliente decide desarrollar industrialmente uno o varios productos en los que resulte parcial o totalmente aplicable el estudio de este proyecto, deberá comunicarlo a la empresa consultora.

10. La empresa consultora no se responsabiliza de los efectos laterales que se puedan producir en el momento en que se utilice la herramienta objeto del presente proyecto para la realización de otras aplicaciones.

11. La empresa consultora tendrá prioridad respecto a otras en la elaboración de los proyectos auxiliares que fuese necesario desarrollar para dicha aplicación industrial, siempre que no haga explícita renuncia a este hecho. En este caso, deberá autorizar expresamente los proyectos presentados por otros.

12. El Ingeniero Director del presente proyecto, será el responsable de la dirección de la aplicación industrial siempre que la empresa consultora lo estime oportuno. En caso contrario, la persona designada deberá contar con la autorización del mismo, quien delegará en él las responsabilidades que ostente.