DESCRIPCIÓN, INDEXACIÓN, BÚSQUEDA Y ADQUISICIÓN DE SECUENCIAS DE VÍDEO MEDIANTE DESCRIPTORES MPEG-7

Jordi Delcor Ballesteros
Verónica Pérez Noriega
Tutor: Javier Ruiz Hidalgo
Universidad UPC, 2006

AGRADECIMIENTOS

Primero de todo agradecerle al profesor del departamento de Teoría de la Señal y Comunicación Javier Ruiz Hidalgo, tutor del proyecto, su dedicación, paciencia, ayuda prestada y la velocidad de contestación de e-mails.

Agradecerle al departamento de TSC de la EUETIT el hecho de prestarnos las instalaciones oportunas para la realización del proyecto.

Y agradecer por último a compañeros, amigos y familias su apoyo y ánimo en los momentos más difíciles.

A todos ellos, MUCHAS GRACIAS.

Verónica Pérez Noriega

Jordi Delcor Ballesteros

ÍNDICE

1. OBJETIVOS

Hoy en día existe una gran cantidad de información audiovisual en formato digital. Esto es debido al gran desarrollo de las nuevas tecnologías de comunicación y al uso generalizado de internet. Este mundo tan innovador, ofrece cada vez más fuentes de material audiovisual disponibles para los usuarios, como pueden ser bases de datos personales o profesionales y difusión de video (TV). Pero lo más importante no es tanto el material en concreto, sino tener un método eficiente y sencillo de buscarlo, gestionarlo, adquirirlo y recuperarlo.

En el caso de que la información que queremos buscar es de tipo texto, la situación es más sencilla, ya que tanto para consultar como para describir el documento utilizamos la misma herramienta, un conjunto de palabras. Pero en el caso de las imágenes o de los videos la situación empeora. Una imagen puede ser interpretada de diferentes formas según quien la observe, por lo tanto escoger las características más apropiadas que faciliten la consulta a los usuarios, es más complicado.

Nos podemos encontrar delante de tres situaciones determinadas cuando disponemos de una colección de material audiovisual. Por una banda cuando queremos acceder a una imagen, conociendo que se encuentra dentro de la base de datos. Por otra banda cuando se navega en busca de imágenes desconocidas y finalmente cuando se utiliza un buscador introduciendo palabras claves. Esta última opción es la manera más utilizada para acceder a imágenes o videos, pero desgraciadamente es la opción con un resultado más ineficiente de búsqueda. Los buscadores más comunes, por ejemplo Google, realizan consultas utilizando características de la imagen como son el nombre, el tamaño, o un texto adjunto. Pero como se puede prever dichos atributos no son suficientes para identificar completamente una imagen.

Son necesarias unas herramientas que permitan una gestión y procesado de dicho material con un método eficiente y automático. En definitiva un entorno que haga posible extraer características del contenido, a fin de manipularlas y elaborar unos índices productivos de búsqueda.[4]

La idea principal del proyecto se basa en generar descriptores MPEG-7 para la indexación de imágenes y videos.

Y los objetivos ha llevar a cabo son la programación de dichos descriptores, junto a la realización de sus archivos de extracción y comparación correspondientes.

El orden establecido para llevar a cabo estos objetivos ha sido el siguiente:

En primer lugar la familiarización con el software de desarrollo, Soft_Image[1]. Lo cual conlleva el entendimiento del modo de acceso, la forma de crear funciones y el modo de compilación además de la comprensión del manejo de imágenes y vídeos mediante la estructura imagen y diversas funciones de lectura, escritura y procesado.

Posteriormente la familiarización con las herramientas de descripción de contenido, los descriptores MPEG-7. Tanto en el sentido semántico del término, ¿Qué son? ¿Que tipos hay? ¿Cual es su utilidad?, como en el sentido técnico, ¿Qué estructura tiene un descriptor? ¿Cómo creo un descriptor?

Una vez definidas las ideas principales del proyecto, se procede a la programación de los cuatro descriptores seleccionados junto a sus respectivos ejecutables.

[1] Veáse apartado 7. Herramientas utilizadas.

Finalmente, una vez superados todos los objetivos y con la realización del proyecto terminado, el material creado será asequible para todos los usuarios de Soft_Image. Tanto proyectistas que trabajen en el campo de la indexación de videos e imágenes como para todos los profesores o doctorados del departamento de TSC.

Las aplicaciones más directas, de estos descriptores, pueden ser la indexación de información audiovisual en bibliotecas digitales y en servicios culturales como museos de historia o galerías de arte. La selección de material multimedia de difusión en la televisión o la radio o hasta la creación de un sistema web para la indexación, búsqueda y adquisición de videos almacenados en una base de datos.

2. MÉTODOS DE INDEXACIÓN DE IMÁGENES

2.1. LOS METADATOS Y SUS ESTÁNDARES

La palabra metadatos tiene un significado que, etimológicamente hablando, querría decir "más allá de los datos", no obstante el significado exacto no está definido actualmente. Una definición que se usa a menudo, es que los metadatos son "datos sobre datos", es decir, se trataría de un objeto que habla de otro objeto.

El concepto de metadato, se ha generalizado hoy en día a cualquier tipo de información descriptiva sobre recursos, incluyendo los que no son de naturaleza digital, de manera que aún siendo reciente el término metadato, el uso de este tipo de información, se remonta siglos atrás. Así, por ejemplo, podríamos considerar metadatos toda la información que podemos encontrar en catálogos de libros, de tarjetas y hoy en día, en catálogos en línea.

Con todo esto, podríamos decir, de manera formal, que un metadato es un dato que se encarga de mantener un registro sobre el significado, contexto o propósito de un objeto informativo, de tal forma que permita descubrir, entender, extraer y administrar dicho objeto. En general, este tipo de información, se crea de manera corta y concisa, con un tamaño menor al del objeto que describen, de manera que se facilita su almacenamiento e intercambio. Sin embargo, si acotamos la definición de metadatos dándole un sentido más estricto, los metadatos sólo serían posibles en un contexto digital y en red ya que sólo dentro de este contexto se pueden utilizar los metadatos con la función que les caracteriza, que es la de la localización, identificación y descripción de recursos, legibles e interpretables por máquina.

Los metadatos, no solo permiten describir colecciones de objetos, también nos permiten identificar los procesos en los que están involucrados.

2.2. EVOLUCIÓN HISTÓRICA DE LOS METADATOS

Los metadatos tienen sus raíces en el catálogo, no obstante, los criterios de clasificación se limitaban simplemente al orden alfabético.

A principios del siglo XX, estos criterios de clasificación empiezan a sofisticarse, sobretodo a partir de la década de los años sesenta, en la cual, los métodos de producción en masa hicieron necesario disponer de múltiples copias de los catálogos existentes. Por este motivo, se hizo necesario desarrollar un sistema de estándares de codificación de este tipo de información, que son los que hoy en día conocemos como estándares de metadatos

Los primeros metadatos digitales y sus bases, empezaron a generarse a finales del siglo XX, con la aparición de numerosos estándares de codificación, lenguajes y protocolos que se usan en la generación de catálogos, como por ejemplo:

- MARC (machine readable cataloguing)
- ISO-23950
- SGML (standard generalized markup language)
- DTD (document type definition)
- The warwik framework

2.3. ESTÁNDARES DE METADATOS

Un estándar de metadatos es una colección de palabras clave o estructuras que definen distintos conceptos, cubriendo de esta manera un cierto campo del conocimiento estable y no muy extenso. Se usan para describir los aspectos más importantes del significado de declaraciones estructuradas dentro de este campo del conocimiento.

La generación de estándares de metadatos, es una inversión de futuro porque permitirá trabajar con ellos a largo plazo sin darle importancia al cambio de tecnología. No obstante esta estandarización, es muy dificultosa y, todavía hoy, no existe ningún estándar que tenga aceptación global.

Existen distintos estándares de metadatos, cada uno de ellos con distintos esquemas de descripción. En los distintos estándares, cada objeto se describe por medio de una serie de atributos y el valor de estos atributos es el que puede servir para recuperar la información. Dependiendo de la clase de metadatos puede existir: información sobre elementos de datos o atributos, información sobre la estructura de los datos, información sobre un aspecto concreto, etc. De forma general, podemos encontrar metadatos referidos a:

- el contenido (concepto)
- aspectos formales (tipo, tamaño, fecha, lengua, etc.)
- información del *copyright*
- información de la autentificación del documento o recurso
- información sobre el contexto (calidad, condiciones o características de acceso, uso, etc.)

Los estándares que se están desarrollando, se podrían dividir en categorías de acuerdo con los propósitos generales de cada marco de metadatos:

- Administrativos: se refieren a información provista para facilitar la administración de los recursos, datos sobre como y cuando se creo un objeto, que actividades se han efectuado en relación al contenido, etc.
- Descriptivos y de descubrimiento: hacen referencia a la información necesaria para encontrar, describir y distinguir cada uno de los objetos de información.
- Técnicos: son los estándares de metadatos relacionados con elementos que describen como funciona o como ha de ser interpretado un sistema. Un ejemplo serian los metadatos que describen el formato de una imagen digital
- Modelos: estándares de metadatos que tienen relación con las piezas de un objeto de información compuesto, en términos de cómo se interrelacionan cada uno de sus componentes.

Aún así, esta clasificación, no define con exactitud unos límites definidos entre estándares de metadatos, ya que muchas veces, éstos se pueden incluir en más de uno de los grupos nombrados. Es por esta razón, que se puede utilizar un diagrama triangular como el de la figura 1, que contiene algún ejemplo, para hacer una clasificación de los estándares existentes:

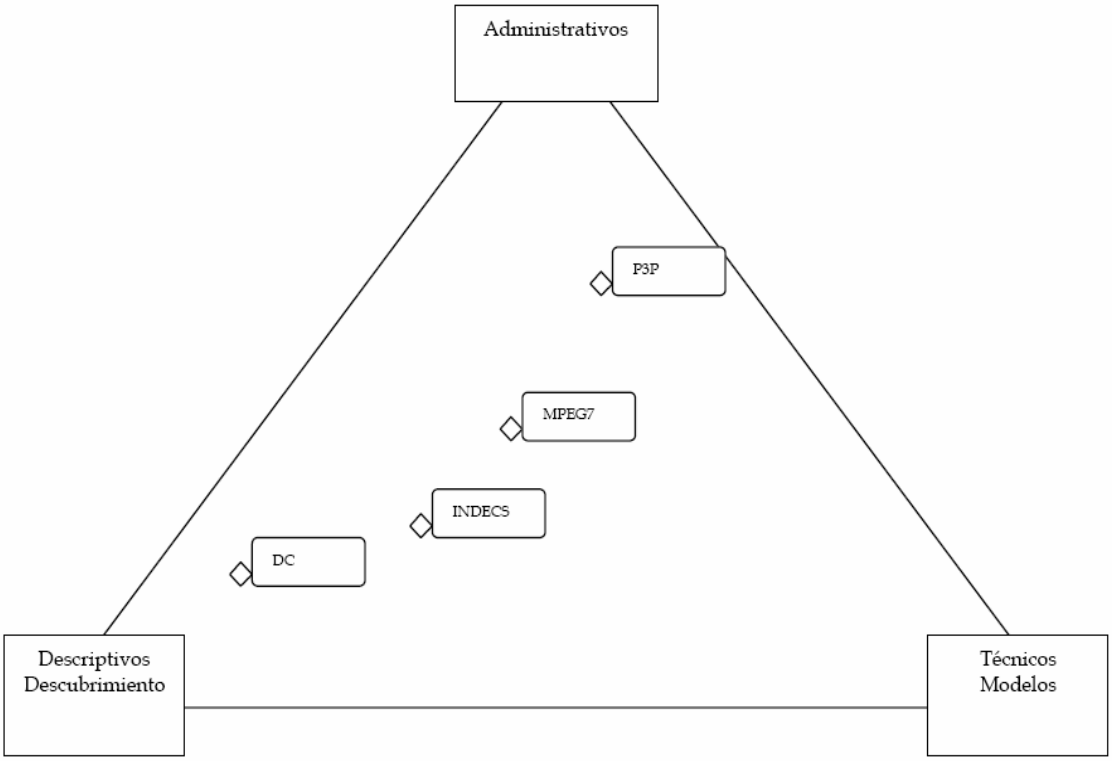


Fig. 1 Diagrama de clasificación de estándares de metadatos

Dicho todo esto, a continuación se presenta una muestra de estándares de metadatos, seleccionada a partir de los innumerables ejemplos existentes, con ellos también, se añade una referencia a partir de la cual se puede encontrar más información interesante sobre cada caso:

Metadatos para la descripción:

- DC: *Dublin Core Metadata Initiative* [11]. http://dublincore.org/
- METS: *Metadata Encoding and Transmission Standard.* http://www.loc.gov/standards/mets/ Se trata de un esquema para describir objetos de bibliotecas digitales complejas que utiliza el lenguaje XML schema y asocia metadatos administrativos

y descriptivos. El estándar es mantenido por la Network Development and MARC Standards Office de la Biblioteca del Congreso Permite describir separadamente archivos digitalizados (por ejemplo las distintas páginas de un libro).

- MODS: *Metadata Object Description Schema.* http://www.loc.gov/standards/mods/ Es un esquema de metadatos descriptivo que se deriva del MARC 21y que permite crear la descripción de recursos originales o seleccionar los registros existentes en MARC 21. Utiliza el lenguaje y la sintaxis XML y puede utilizarse como un formato específico de la Próxima Generación de Z39.50.
- EAD: *Encoded Archival Description.* http://www.loc.gov/ead/ Se trata de un proyecto internacional que desarrolla pautas para el marcado de textos electrónicos (novelas, obras de teatro, poesía, etc.) y se enfoca al campo de las humanidades.
- TEI: *Text Encoding Initiative* http://www.tei-c.org/
- IFLA: *Metadata Resources for Digital Libraries.* http://www.ifla.org/II/metadata.htm
- CIMI: *Computer Interchange of Museum Information.* http://www.cimi.org/ (El Consorcio cerró sus operaciones en 2003).

Metadatos para presentaciones:

- MCF: *Meta Content Framework.* http://www.textuality.com/mcf/NOTE-MCF-XML.html y http://www.w3.org/TR/NOTE-MCF-XML/ (para colecciones de información en red usando XML)

Metadatos para la industria y el comercio electrónico:

- UDEF: *Universal Data Element Framework*. http://www.udef.org/

Metadatos para multimedia:

- MPEG-21: *Multimedia Framework*. http://www.chiariglione.org/mpeg/standards/mpeg-21/mpeg-21.htm (Metadatos para colecciones de vídeo, álbumes musicales, etc.)
- MPEG-7: *Multimedia Content Description Interface*. http://www.chiariglione.org/mpeg/standards/mpeg-7/mpeg-7.htm (Metadatos para contenido audiovisual, esto es, para describir contenido multimedia)

Metadatos para la educación y el aprendizaje:

- IMS: *Instructional Management Systems*. http://www.imsproject.org/
- GEM: *The Gateway to Educational Materials*. http://www.thegateway.org/
- LOM: *Learning Object Metadata*. http://ltsc.ieee.org/wg12/

Metadatos para el gobierno y la administración:

- EdNA Metadata Standard: *Education Network Australia*. http://www.edna.edu.au/edna/go/pid/385
- AGLS: *Australian Government Locator Service*. (AGLS).
- GILS: *US Government Information Locator Service*. http://www.usgs.gov/gils/

Metadatos geoespaciales:

- CSDGM:

  Content Standard for Digital Geospatial Metadata.

  http://www.fgdc.gov/metadata/contstan.html

Metadatos generales:

- W3C Metadata activity:

  http://www.w3.org/Metadata/Activity.html
- W3C Semantic Web activity: http://www.w3.org/2001/sw/

3. INTRODUCCIÓN AL MPEG-7

La indexación de información existe desde hace siglos. En un principio la indexación se realizaba manualmente, pero con la llegada de la digitalización y el aumento considerable de archivos multimedia se hace necesario la creación de estándares de codificación.

Con dicho propósito surge MPEG, un grupo de trabajo de ISO/IEC que se encarga del desarrollo internacional de normas para el procesado, compresión, descompresión y codificación tanto de vídeo como de audio.

El primer estándar que surgió fue MPEG-1, destinado a la compresión de video y audio, y de donde nacieron las normas dedicadas a los CD's de video y al formato MP3. Dos años después ya estaba en el mercado un conjunto de nuevas normas, MPEG-2, responsable de definir las pautas para el servicio de TV por satélite, por cable y para formato DVD. Posteriormente se presenta el tercer estándar denominado MPEG-4, el cual proporciona un cambio muy positivo en la línea de interpretación de vídeos de las normas anteriores. Se introduce el concepto de objeto, la imagen no se analiza píxel a píxel sino por objetos visuales, los cuales posen un significado independiente. "La información no es del tipo en este punto de la pantalla hay este valor de luminancia, sino en esta secuencia aparece un coche y se mueve con esta trayectoria" [7]. Un nuevo concepto destinado a aplicaciones multimedia para móviles y televisión de consumo doméstico.

Hasta este momento todos los estándares expuestos compartían una similitud, representaban con bits el contenido del material audiovisual en si mismo. Pero a partir de aquí esta situación avanza cuando surge uno de los estándares más innovadores de MPEG. Con MPEG-7 la filosofía es distinta. El objetivo ahora es codificar "los bits que

hablan de los bits" [6], es decir, los datos que describen la información. Introduciendo de este modo el concepto de metadato[3].

3.1 DESCRIPCIÓN Y OBJETIVOS

MPEG-7 o también llamado "Interfaz de descripción de contenidos multimedia", se creó en el año 2001 para estandarizar la extracción de características basadas en el contenido de los diferentes tipos de información multimedia.

El cuarto estándar de MPEG nos proporciona una completa gestión del material audiovisual tanto en la búsqueda o filtrado como en el acceso o reproducción.

Las características principales que describen MPEG-7 son las siguientes [6]:

- El tipo de dato audiovisual considerado va desde el audio, la voz, imágenes fijas, video, mallas en 3D y gráficos hasta la información de como los objetos están distribuidos en las escenas.
- Las descripciones del material multimedia pueden ser de dos tipos:
    - Información sobre el contenido, como puede ser el autor, el genero, el título o el formato.
    - Información existente en el contenido, la cual permite describir el elemento a través de significado semántico (descripción de alto nivel), relacionado con la interpretación del contenido o significado estructural (descripción de bajo nivel) el cual permite la extracción automática de color, forma texturas, sonidos, etc …

      Los descriptores de alto nivel se caracterizan por ser eficientes y directos pero poco flexibles mientras que los de bajo nivel son genéricos, flexibles y permiten búsquedas "inteligentes" [5].

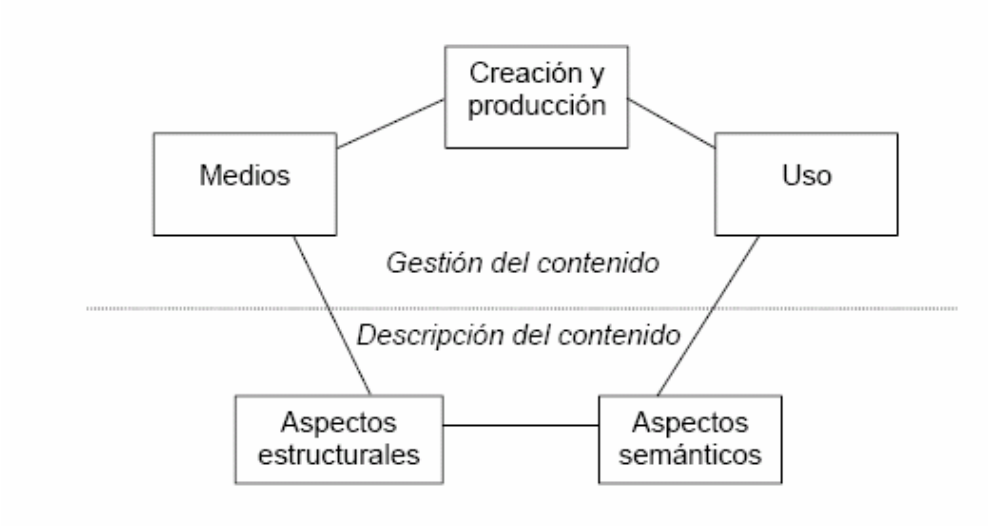


Fig. 2 Gestión y descripción del contenido

- La descripción es independiente del formato que tengan los datos audiovisuales. Las normas definidas no dependen de cómo esta codificada o almacenada la información. MPEG-7 no solamente describe contenidos codificados en MPEG sino que es capaz de describir desde un video en formato VHS hasta una imagen impresa en un papel.
- Aprovechando las ventajas de MPEG-4, este nuevo estándar ofrece la posibilidad de hacer descripciones referentes a partes más específicas del material. Pueden ser referidas a objetos o las relaciones temporales y espaciales dentro de la escena.
- Uno de los puntos fuertes del estándar se centra en las posibilidades que ofrece en el momento de transmitir las descripciones. MPEG-7 permite multiplexarlas con el contenido, con la ventaja de tenerlas físicamente asociadas al material, o transmitirlas por separado junto con un puntero el cual señala el objeto fuente [5].

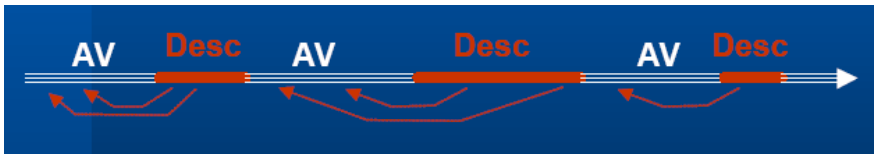


Fig. 3 Descripciones y contenidos multiplexados

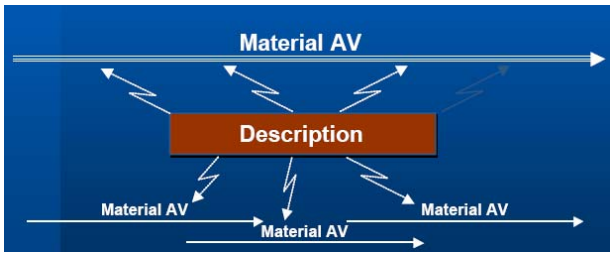


Fig. 4 Descripciones y contenido separados

- Tanto la generación de descripciones, la cual engloba la extracción de características, indexación y segmentación, como la consumición de dichas descripciones gracias a máquinas de búsqueda son aspectos no considerados por las normas MPEG-7. En el esquema de la parte inferior se puede observar una aplicación real de MPEG-7 donde los recuadros de color granate corresponden a los procesos que se pueden realizar con libertad, ya que no están definidos por el estándar, mientras que el recuadro central de color azul es donde se aplican las normas estandarizadas.[5]

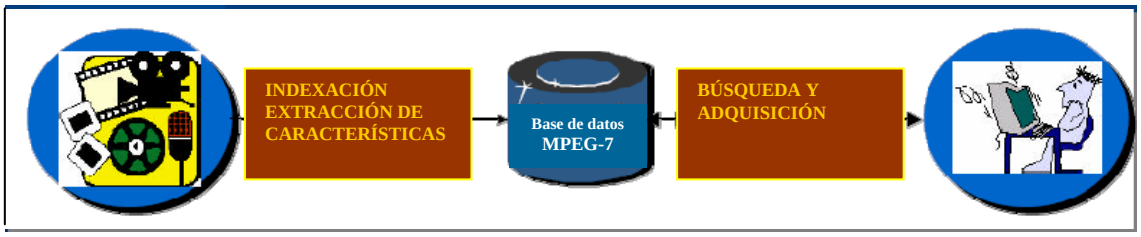


Fig. 5 Aplicación real de MPEG-7

- La explicación de que haya procesos no definidos por MPEG-7, donde el usuario tiene libertad de actuación, viene dada por la necesidad de expansión del estándar. El objetivo es definir los mínimos elementos posibles que garanticen interoperatividad. [5]

3.2 ARQUITECTURA DE FUNCIONAMIENTO

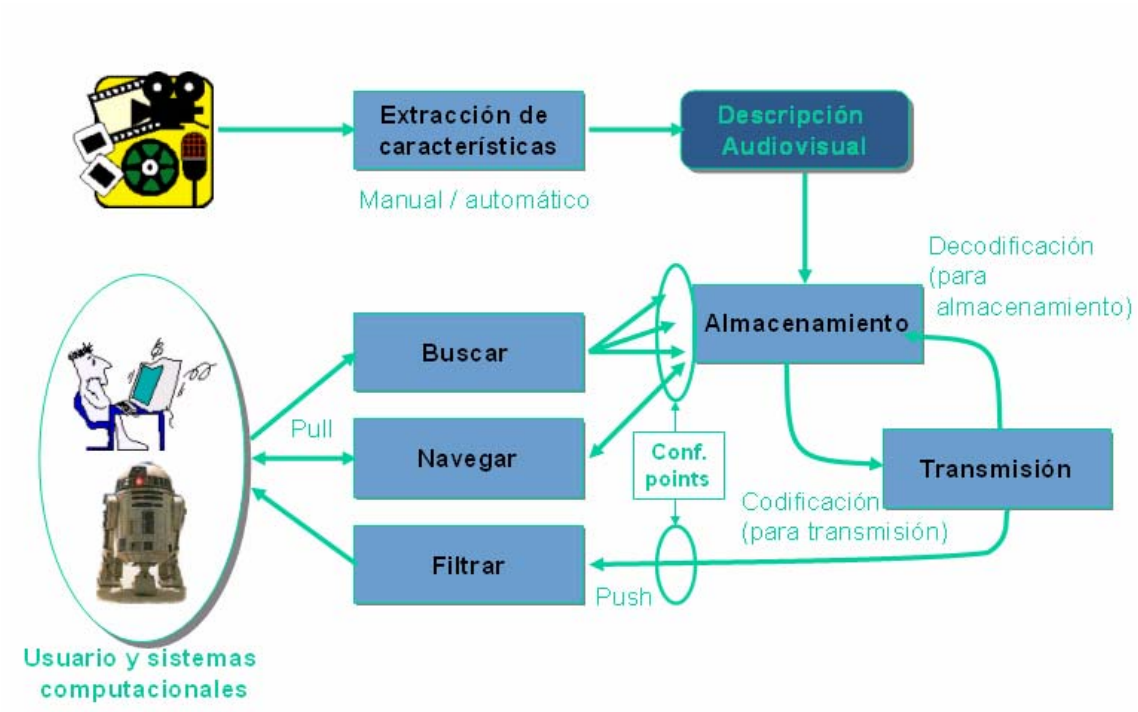


Fig. 6 Arquitectura de funcionamiento de posibles mecanismos que utilicen MPEG-7

Para poder entender mejor en que consiste MPEG-7 y que utilidades aporta a sus usuarios hace falta entender y analizar la estructura de funcionamiento que nos muestra el esquema de la Fig. 6.

Como se puede observar, al principio de la cadena es donde se sitúa el contenido multimedia, ya sea un video, una imagen o un sonido. Seguidamente de lo que se trata es de extraer las características de dicho material obteniendo así una descripción determinada. Entonces el material audiovisual junto a su descriptor se puede codificar para transmitirlo o descodificar para almacenarlo.

Una vez que todo esta almacenado en la base de datos pueden ocurrir dos tipos de situaciones diferentes:

Por una banda la aplicación de tipos *pull* ("Explorar y buscar"), donde los procesos controlados por el cliente formulan peticiones al almacén de descriptores, obteniendo como respuesta un grupo de descripciones que cumplen las condiciones solicitadas. De esta manera el usuario puede extraer la información asociada, manipularla, etc…

Y por otra banda la aplicación tipo *push* ("Filtrado y selección automática"), donde el cliente ya conoce las descripciones existentes y selecciona una para realizar una acción programada, como puede ser filtrar una imagen. [3]

3.3. COMPONENTES

El estándar MPEG-7 esta formado por los siguientes componentes [2]:

- **MPEG-7 Systems**: Describe las herramientas necesarias para desarrollar descripciones MPEG-7 para un eficiente transporte, almacenamiento y arquitectura del terminal.
- **MPEG-7 Description Definition Language**: Lenguaje que especifica los esquemas de descripción permitiendo la extensión y modificación de los existentes. Es un lenguaje basado en XML.
- **MPEG-7 Visual**: Conjunto de herramientas de descripción para descripciones visuales.
- **MPEG-7 Audio**: Conjunto de herramientas de descripción para descripciones de audio.
- **MPEG-7 Multimedia Description Schemes**: Conjunto de herramientas de descripción con características genéricas y descripciones multimedia.

- **MPEG-7 Reference Software**: Implementación de las partes relevantes de software del estándar MPEG-7 en estado normativo
- **MPEG-7 Conformance Testing**: Normas y procedimientos para la aprobación de las implementaciones de MPEG-7.
- **MPEG-7 Extraction and use of descriptions**: Material informativo (en forma de informe técnico) sobre la extracción y el uso de algunas herramientas de descripción.
- **MPEG-7 Profiles and levels**: Conjunto de normas y perfiles.
- **MPEG-7 Schema Definition**: Especificación del esquema utilizado en la DDL (Description Definition Language).

Una vez comentadas las características principales y las partes que forman MPEG-7 lo más adecuado es central el tema en la parte del estándar que más interesa en la realización del proyecto, los descriptores.

4. DESCRIPTORES MPEG-7

El estándar MPEG-7 define una serie de descriptores que permiten analizar y caracterizar el contenido de fuentes audiovisuales para su posterior indexación, búsqueda o comparación. Este conjunto de descriptores, incluye, como posiblemente se puede intuir, dos grandes bloques: descriptores de audio y descriptores de video. A su vez, éstos segundos, también conocidos como descriptores visuales, que son los que realmente trascienden en este proyecto, se clasifican según su función [2].

En la figura 7 se puede observar ésta división:

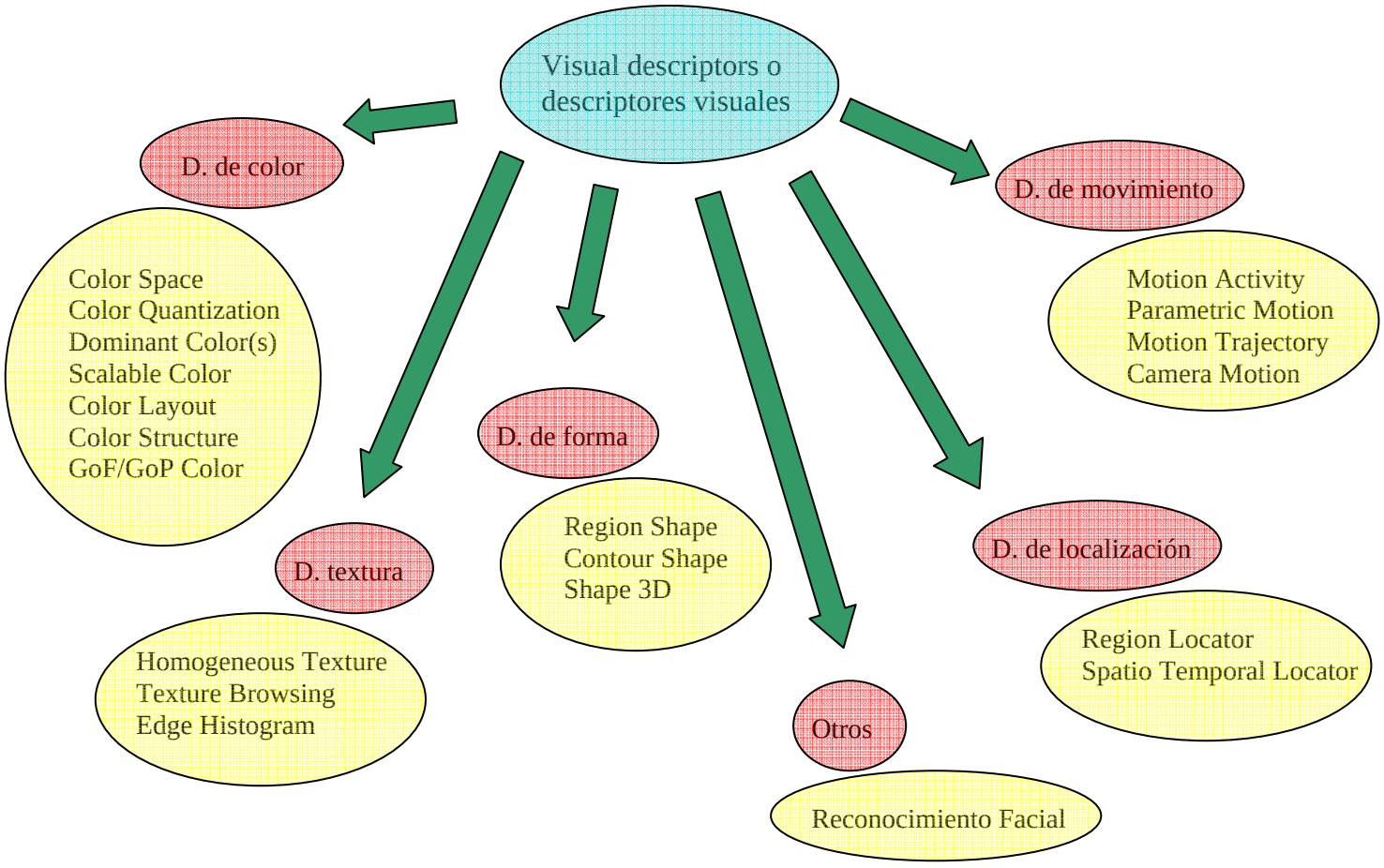


Fig. 7 Esquema de clasificación de los descriptores visuales

4.1. DESCRIPTORES DE COLOR

Color Space

Consiste en un tipo de datos que especifica el espacio de color en el cual se expresan o trabajan los otros descriptores de color. Los espacios de color que contempla son los siguientes:

RGB (Red, Green, Blue)
YCrCb (Luminancia, Crominancia)
HSV (Hue, Saturation, Value)
HMMD (Hue, maximum, minimum, difference)
Matriz de transformación lineal con referencia a RGB
Monocromo

Color Quantization

Este descriptor define una cuantificación uniforme de un espacio de color determinado. El número de valores que el cuantificador produce es configurable de manera que posee una flexibilidad elevada que le da una amplia gama de usos. Dentro de MPEG-7 este descriptor se combina con descriptores del color dominante, para hacer comparables por ejemplo dos resultados de un determinado descriptor.

Dominant Color(s)

Este descriptor es el más conveniente para ser utilizado en imágenes o zonas de ellas, en las cuales un pequeño número de colores es suficiente para caracterizar la información cromática de la región determinada. Seria aplicable por ejemplo en imágenes de banderas o marcas determinadas. En este caso la cuantificación se usa para extraer un reducido número de colores que sean suficientes como para

caracterizar la imagen o región. También se calcula una coherencia espacial entre estos colores y dónde están situados la cual cosa se utilizará en algoritmos de similitud.

Scalable Color

Este descriptor consiste en un histograma de color en el espacio HSV, codificado con una medida de Haar. Su representación se puede escalar de manera que se adecue lo máximo al tamaño de datos con el que se quiere trabajar. Este descriptor es útil en comparaciones imagen-imagen o en búsquedas basadas en características de color. La fiabilidad de la búsqueda aumenta proporcionalmente al número de colores distintos que se tengan en cuenta en el histograma.

Color Layout

Este descriptor permite representar la distribución espacial del color dentro de las imágenes de una manera muy compacta, con lo cual representa una herramienta de gran utilidad a la hora de buscar imágenes a partir de modelos determinados, y lo hace con gran eficiencia y velocidad. Su fácil cálculo permite también usarlo en la comparación de secuencias de imágenes, en las cuales se precisa un análisis de similitud entre cada una de sus componentes.

Las grandes ventajas de este descriptor son:

- No depende ni del formato, ni de la resolución, ni del margen dinámico de las imágenes o videos en que se use. Por este motivo, puede usarse para comparar imágenes o videos con distintas resoluciones o para comparar imágenes enteras con partes de imágenes.

- El software/hardware que requiere es relativamente mínimo (usa solamente 8 bytes por imagen cuando trabaja por defecto). Esto lo convierte en un descriptor adecuado para ser utilizado en dispositivos móviles en los que los posibles recursos se ven limitados por la capacidad del hardware.
- Las características que analiza de una imagen las representa en dominio frecuencial de manera que los usuarios pueden manipular con facilidad su contenido ciñéndose a la sensibilidad del sistema visual humano.
- Permite trabajar con distintas precisiones de descripción de manera que se agudizan las comparaciones cuando sea necesario.

Color Structure Descriptor

Este descriptor caracteriza la distribución de los colores en una imagen. Construye una especie de histograma de color, en el cual tendrán mayor importancia a los colores que más se reparten por la imagen. El descriptor divide la imagen en bloques de 8x8 píxeles y analiza dentro de estos bloques los distintos colores que aparecen, incrementándolos así en el histograma. A diferencia de un histograma de color, permite distinguir entre dos imágenes que tengan la misma cantidad de píxeles de un color pero con distinta distribución de estos píxeles. Este descriptor es útil para comparaciones imagen-imagen y añade funcionalidades distintas a las del histograma de color que permiten mejorar la búsqueda de similitud en determinados tipos de imágenes, como por ejemplo las imágenes de naturaleza.

GoF/GoP Color (Group of Frames/Group of Pictures)

Este descriptor es una extensión del Scalable Color, que a diferencia de éste, que está definido para imágenes inmóviles, se aplica a

secuencias de video o secuencias de imágenes fijas. Este descriptor da la posibilidad de calcular de tres formas distintas el histograma de color:

- Histograma promedio: toma de cada imagen de la secuencia el promedio de los valores del histograma.
- Histograma de mediana: toma de cada imagen de la secuencia el valor central del conjunto de valores del histograma. Es más fiable ante errores o picos de intensidad de la imagen.
- Histograma de intersección: toma de cada imagen de la secuencia el mínimo del conjunto de valores del histograma, para así ver cual es el color "menos común" en el conjunto de imágenes.

4.2. DESCRIPTORES DE TEXTURA

Homogeneous Texture

Este descriptor emergió como una importante herramienta a la hora de buscar y escoger dentro de grandes colecciones de imágenes de gran similitud visual. Este descriptor utiliza un banco de 30 filtros que permite obtener una afinada descripción de las distintas texturas de la imagen para poder comparar de esta manera con las de otras. Es una herramienta muy útil por ejemplo para distinguir determinadas zonas en imágenes aéreas, por ejemplo, cultivos.

Texture Browsing

Este descriptor especifica la caracterización perceptiva de una textura, la cual es similar a la caracterización de ella que hace un ojo humano, en cuanto a términos de regularidad, tosquedad y direccionalidad. Es útil para búsquedas y clasificaciones a "grosso modo" de texturas. Su implementación es parecida a la del anterior.

Edge Histogram

El Edge Histogram es un descriptor que nos facilita información sobre el tipo de contornos o bordes que aparecen en la imagen. Trabaja dividiendo la imagen en 16 sub-imágenes y es capaz de analizar en ellas el tipo de bordes existentes con el uso de distintos filtros que le permiten diferenciar si son bordes horizontales, verticales, oblicuos o aleatorios. Su utilización principal es la comparación imagen-imagen, especialmente en imágenes de naturaleza con una gran no-uniformidad de contornos. Su uso es muy útil también en combinación con el de otros descriptores como por ejemplo el histograma de color.

4.3. DESCRIPTORES DE FORMA

Region Shape

La forma de un objeto en una imagen, puede consistir en una sola región o un conjunto de regiones como vemos en las siguientes imágenes:

Fig. 8 Ejemplos de imágenes con las que trabaja Region Shape

Este descriptor permite clasificarlas según esta característica, de manera que podemos comparar formas de distintas imágenes y ver por ejemplo si se trata del mismo objeto u objetos similares.
Las grandes ventajas de este descriptor son su reducido tamaño y su velocidad, hay que tener en cuenta que el tamaño de los datos necesarios para su representación está fijado en 17,5 bytes.

Contour Shape

A diferencia del anterior, este descriptor en lugar de analizar el conjunto de regiones que dan lugar a una forma, relaciona ésta última con su contorno. Se caracteriza por representar muy bien las características de contorno con lo que facilita posteriores búsquedas y recuperaciones, es robusto ante movimientos, ante oclusiones en las formas y ante distintas perspectivas, y es sumamente compacto.

Shape 3D

El Shape 3D permite describir con detalle la forma de mallas en 3D. Herramienta que hoy en día debido al continuo desarrollo de las tecnologías multimedia es de gran utilidad.

4.4. DESCRIPTORES DE MOVIMIENTO

Camera Motion

Es un descriptor que da información sobre los movimientos que efectúa la hipotética cámara que toma la secuencia de imágenes.

Motion Trajectory

Este descriptor nos permite analizar la trayectoria de un objeto en una secuencia de imágenes, la cual se consigue con la localización en tiempo y espacio de un punto representativo del objeto determinado.

Parametric Motion

Consiste en describir el movimiento de ciertos objetos en una cadena de imágenes. Estos objetos se definen como regiones en la imagen, y su movimiento se registra de una manera compacta como un conjunto de parámetros. Este descriptor permite diferenciar numerosos tipos de movimiento elementales como translaciones, rotaciones, "zooms", de manera que cualquier otro movimiento se puede especificar como una combinación de estos.

Motion Activity

Podríamos decir que se trata de un descriptor que intenta evaluar la "intensidad de la acción" en una secuencia de imágenes, de manera parecida a como lo percibiríamos nosotros los humanos. Es decir, aportar información que permite diferenciar escenas lentas, rápidas, a cámara lenta, etc.

4.5. DESCRIPTORES DE LOCALIZACIÓN

Region Locator

Este descriptor permite la localización de determinadas regiones en una imagen.

Spatio Temporal Locator

Seria como el anterior pero aplicado a secuencias de imágenes de manera que localiza determinadas regiones analizando tiempo y espacio.

4.6. RECONOCIMIENTO FACIAL

Como su nombre indica, este descriptor permite comparar caras para analizar su parecido o buscar caras con alto parecido a una en concreto. Esto se consigue relacionando distintas posiciones de la cara con las intensidades de la imagen en esa posición, de manera que creamos unos datos de referencia que luego pueden compararse con otros para analizar la similitud.

5. DESCRIPTORES REALIZADOS

5.1. EDGE HISTOGRAM

Edge Histogram es el descriptor que se encarga de representar la distribución espacial de cinco tipos de contornos en regiones locales de una imagen. Es capaz de reconocer cuatro contornos direccionales, es decir, con una dirección claramente marcada, y un contorno de tipo no direccional. El análisis de los bordes juega un papel muy importante en la percepción de las imágenes, ya que es un método que nos permite la recuperación de imágenes con significado semántico similar.

De este modo el objetivo es encontrar imágenes semejantes a otras imágenes, ya sea a través de ejemplos o de esbozos, especialmente en el caso de imágenes naturales con distribuciones de bordes no uniformes. En este contexto la eficacia de reconocimiento de las imágenes puede verse significativamente mejorada si dicho descriptor es combinado con otros descriptores, como los descriptores de color.

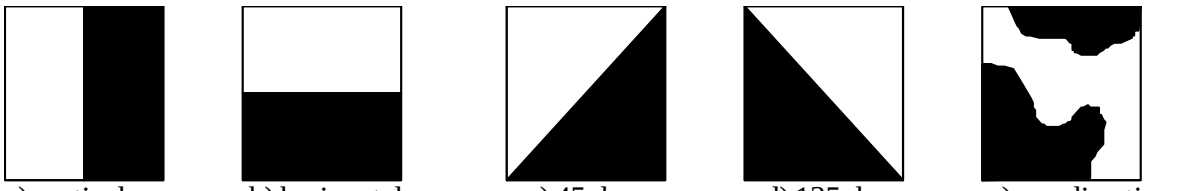


Fig. 9 tipos de bordes reconocidos por Edge Histogram

Edge Histogram divide la imagen en 16 sub-imágenes (4x4 bloques no solapados), de los cuales tras un posterior procesado se obtiene un total de 80 valores denominados *Bincounts* que forman el histograma de bordes que caracteriza dicha imagen.

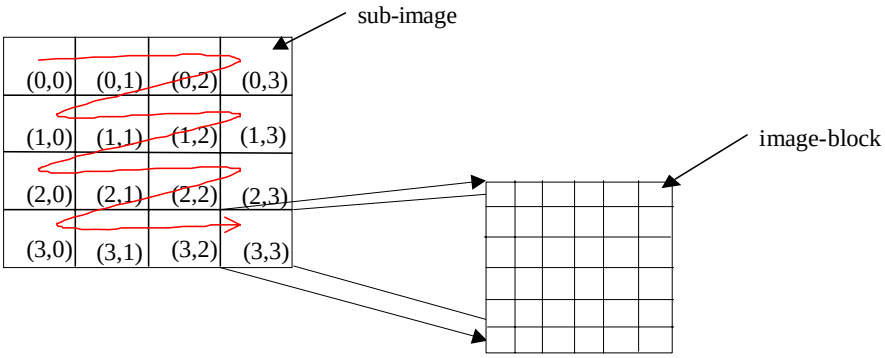


Fig. 10 Proceso de la división por bloques de la imagen

La implementación de este descriptor se ha realizado de la siguiente manera:

Primeramente se crea la estructura de tipo descriptor denominada *VdEdgeHistogram*, la cual esta formada por un vector de 80 valores tipo short encargado de almacenar los valores que forman el histograma de bordes y que lleva el nombre de *Bincounts*. Se compone de 80 valores ya que podemos analizar 5 tipos de bordes diferentes por cada uno de las 16 sub-imágenes en las que se divide una imagen (16x5=80).

Bincounts[k]	Definición
Bincounts[0]	Bordes vertical en sub-image (0,0)
Bincounts[1]	Bordes horizontal en sub-image (0,0)
Bincounts[2]	Bordes de 45 grados en sub-image (0,0)
Bincounts[3]	Bordes de135 grados en sub-image (0,0)
Bincounts[4]	Bordes no direccional en sub-image (0,0)
Bincounts[5]	Bordes vertical en sub-image (0,1)
…	…
Bincounts[74]	Bordes no direccional en sub-image (3,2)
Bincounts[75]	Bordes vertical en sub-image (3,3)
Bincounts[76]	Bordes horizontal en sub-image (3,3)
Bincounts[77]	Bordes de 45 grados en sub-image (3,3)
Bincounts[78]	Bordes de 135 grados en sub-image (3,3)
Bincounts[79]	Bordes no direccional en sub-image (3,3)

Tabla 1 Representación del vector Bincounts

La definición de la estructura junto con las cabeceras de todas las demás funciones y acciones necesarias en la especificación del descriptor son guardadas en el archivo de cabecera *VdEdgeHistogram.h*

En segundo lugar se crean las funciones comunes para un descriptor. La función de alocar memoria para la estructura del descriptor *VdEdgeHistogramAlloc*, donde tenemos como salida un puntero de tipo *VdEdgeHistogram* que apunta a dicha estructura. La acción de liberar memoria *VdEdgeHistogramFree*, donde le entramos el puntero que apunta a la estructura y por consiguiente lo destruimos. La función de extracción de valores de descriptor *VdEdgeHistogramGetBin*, donde le entramos un vector que apunta a una estructura *VdEdgeHistogram* junto con una posición entre 0 y 79 correspondiente al vector *Bincounts*, y como resultado obtenemos un valor tipo short que corresponde al valor de la posición introducida. La acción de copiado de descriptores *VdEdgeHistogramCopy*, la encargada de copiar el contenido de un descriptor origen en otro destino del mismo tipo. Y finalmente las funciones de lectura y escritura a disco del descriptor, las cuales permiten manipular el descriptor tanto en formato binario, decimal o XML. Sus nombres son *VdEdgeHistogramToFile* y *VdEdgeHistogramFromFile* y han sido creadas por el tutor del proyecto Javier Ruiz Hidalgo.

Se puede observar que en este tipo de descriptor la función *VdEdgeHistogramSet* no existe, ya que realizaría el mismo proceso que la función *VdEdgeHistogramCalculate* que será explicada más adelante.

Seguidamente se crean las funciones específicas del descriptor. Las principales son *VdEdgeHistogramCalculate*, la cual tiene como función

obtener el vector de valores *Bincounts* a partir de una imagen de entrada, y *VdEdgeHistogramSimilarity* que es la responsable de medir la similitud entre dos descriptores de entrada dando como respuesta un valor entre 0 y 1.

VdEdgeHistogramCalculate dispone como parámetros de entrada de un puntero de tipo *struct image*, que es la estructura que utiliza Soft_image para trabajar con imágenes, y un puntero de tipo *VdEdgeHistogram*, que será utilizado para introducir los valores *Bincounts* calculados en dicha función.

En primer lugar es necesario comprobar que los parámetros de entrada no están vacíos, lo que se gestiona con un comando que imprime por pantalla error en el caso que suceda esta situación. Como se ha comentado anteriormente lo primero que se necesita para calcular dicho descriptor es dividir la imagen de entrada en 16 bloques, 4x4 bloques no solapados, por lo tanto lo primero que se comprueba es que las dimensiones en anchura y altura de la imagen sean múltiples de cuatro para poder hacer un procesado correcto. En caso contrario también se obtendrá un mensaje por pantalla advirtiendo del error.

Seguidamente se calculan las dimensiones que tendrán los bloques, ya que esta medida variará dependiendo del número de píxeles de la imagen de entrada.

El objetivo es ir separando cada una de las 16 sub-imágenes como si se tratase de una imagen independiente para poder procesarlas por separado. Para ello se crean dos variables que fijan la posición del bloque con el cual se esta trabajando, como se observa en la figura 10. Por ejemplo si las dos variables son *x* e *y*, y las dos valen 0, se corresponde con el bloque de la esquina superior izquierda. El recorrido de los bloques se hace de izquierda a derecha, tal y como nos indica la tabla de *Bincounts* que muestra la Tabla 1. Ya que se puede observar

que el primer bloque ha analizar es el (0,0) y el siguiente el (0,1). A continuación se fijan las posiciones máximas de los píxeles que forman el bloque tanto en sentido vertical como horizontal, para facilitar el posterior recorrido de la imagen inicial por esa zona, ya acotada. De este modo cada bloque es almacenado en una nueva estructura *struct image*, que ya anteriormente se le ha sido alocada memoria para ella y para sus datos.

A partir de este momento empieza el procesado de la nueva imagen (sub-imagen), lo que se repetirá para cada una de las 16 nuevas imágenes.

Primeramente se filtra el bloque con la función denominada *DetectionEdges*.

Dicha función tiene como propósito filtrar una imagen de entrada con una imagen adecuada que detecte unos bordes determinados en la imagen inicial. Para ello se establece como parámetros de entrada un puntero tipo *struct image* que apunta a la imagen que se desea procesar, más un valor entre 0 y 3 que indica que tipo de contornos, verticales (0), horizontales (1), de 45 grados (2) y de 135 grados (3), se quiere detectar. A continuación se crean cuatro casos diferentes para cada uno de los valores que puede adoptar. Para cada caso la imagen será filtrada con una imagen de tamaño tres por tres píxeles como las que se muestran a continuación:

-1 0 1 -1 0 1 -1 0 1	1 1 1 0 0 0 -1 -1 -1	1 1 0 1 0 -1 0 -1 -1	0 1 1 -1 0 1 -1 -1 0
C. verticales	C. horizontales	C. de 45º	C. de 135º

Fig. 11 kernels para el filtrado de la imagen

Se puede comprobar la validez de estos kernels observando este ejemplo. Si disponemos de una imagen de entrada tipo:

$$\begin{bmatrix} 0 & 0 & 255 \\ 0 & 0 & 255 \\ 0 & 0 & 255 \end{bmatrix}$$

Y queremos detectar los contornos verticales de la imagen, al convolucionarla con el bloque adecuado obtenemos:

$$\begin{bmatrix} 0 & 0 & 255 \\ 0 & 0 & 255 \\ 0 & 0 & 255 \end{bmatrix} * \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 510 & 0 \\ 0 & 765 & 0 \\ 0 & 510 & 0 \end{bmatrix} \Longrightarrow \boxed{1785}$$

Como podemos observar quedan remarcados los contornos verticales, ya que la transición de 0 a un valor mayor se crea en sentido horizontal.

Por lo contrario observar lo que sucede si a una imagen que se compone de contornos horizontales se le aplica un kernel de extracción de contornos verticales.

$$\begin{bmatrix} 255 & 255 & 255 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 255 & 0 & -255 \\ 255 & 0 & -255 \\ 0 & 0 & 0 \end{bmatrix} \Longrightarrow \boxed{0}$$

Como se esperaba no salen remarcados los contornos verticales.

La imagen resultado del filtrado será almacenada en otra nueva estructura de imagen para posteriormente calcular el valor absoluto de cada uno de los píxeles. Ya que al procesar la imagen con filtros compuestos por valores negativos, el resultado estará compuesto por valores entre -255 y 255. Esto se debe solucionar ya que un valor negativo no interesa en el procesado, por lo tanto con la función de valor absoluto, cambiamos de signo a dichos valores.

Cabe decir que en las tres estructuras de imagen utilizadas en esta función, previamente se les ha alocado memoria a ellas y a sus respectivos datos. En el caso de la estructura que almacena el filtro se utiliza una función de reserva de memoria donde primeramente se deben introducir los atributos de la imagen, ya que esos atributos, ya comentados anteriormente, son conocidos desde el principio. Para las otras dos estructuras en las que se almacenan la imagen resultado del filtrado y la imagen resultado de la operación de valor absoluto, se utiliza una función que reserva memoria utilizando como plantilla otra imagen introducida. En este caso corresponde al bloque inicial introducido en la función. De este modo se puede observar como el puntero a *struct image* que devuelve *DetectionEdges* apunta a una sub-imagen de las mismas dimensiones que la sub-imagen pasada como parámetro de entrada.

Comentar también que dicha función proporciona un control de errores, avisando cuando la sub-imagen de entrada esta vacia o cuando el valor del indicador de contorno no esta comprendido entre 0 y 3.

Finalmente libera la memoria ocupada por el filtro y por la imagen filtrada, ya que no volverán a ser utilizadas en los posteriores procesos.

Señalar que aunque *DetectionEdges* no contempla la detección de contornos no direccionales, el descriptor lo exige en su definición. Por lo tanto el método de localización de dichos contornos se realiza sumando los bloques resultantes para cada uno de los cuatro tipos de contornos señalados anteriormente. Una solución lógica, ya que un contorno no direccional contiene todo tipo de contornos.

El segundo paso consiste en contar el número de píxeles que forman los contornos de la sub-imagen. Para ello se establece un valor umbral, el cual delimita que los píxeles con valor más elevado pertenecen a contornos y los de valor por debajo no pertenecen. En el caso de contornos verticales, horizontales, de 45º y de 135º el valor es de 50, pero en el caso de contornos no direccionales el valor es de 100. Esto es debido a que al calcular este último tipo de contorno se suman cuatro imágenes entre si, por lo que los valores de los píxeles son considerablemente más elevados y para llevar a cabo un recuento más preciso se debe elevar también el valor del umbral.

El siguiente procedimiento es el de normalizar el valor resultante de la suma de píxeles realizada en el paso anterior. Esto se realiza con el fin de obtener un resultado que no dependa del número de píxeles totales que dispone la sub-imagen. Ya que si la sub-imagen es de dimensiones más elevadas lo mas probable es que el resultado de la suma sea elevada, lo que por otra parte no significa que contenga más bordes que una sub-imagen de dimensiones más pequeñas. Para normalizar este valor, lo que se hace es dividirlo entre el número total de píxeles de la sub-imagen. El resultado es un valor comprendido entre 0 y 1.

Y para terminar lo único que falta es cuantificar el valor obtenido de la normalización. MPEG-7 ya nos proporciona en la definición del

descriptor, unas tablas con los valores adecuados que forman los intervalos de cuantificación, diferentes para cada tipo de contorno. Para facilitar el proceso, se crea una acción denominada *Quantablefillvalues* que se encarga de rellenar una matriz de 5x8, donde las filas corresponden cada una a un tipo de borde, con los valores de las tablas de cuantificación. Posteriormente se crean diferentes casos donde dependiendo del intervalo en que se encuentre el valor de píxeles de contorno normalizado, este se convierta en una cifra comprendida entre 0 y 8.

La acción *Quantablefillvalues* no se declara junto con las demás funciones y acciones en el archivo de cabecera *VdEdgeHistogram.h*, sino que se declara aparate, en otro archivo cabecera llamado *VdEdgeHistogram_priv.h* dedicado a las funciones más específicas del descriptor y que no se desea que sean modificadas ni usadas por funciones externas al descriptor.

Finalmente se va introduciendo el valor cuantificado de cada sub-image en la posición adecuada del vector *Bincounts* del descriptor vacío que se nos proporciona. Para saber en cada momento en que posición se debe almacenar el valor calculado, se dispone de dos contadores. Uno que nos indica el número del bloque con el que estamos trabajando (*cont*), con lo que su valor estará entre 0 y 15, y otro que nos indica el procesado de que tipo de contorno es (*edge*), con lo que su valor estará entre 0 y 4. De este modo basta con multiplicar por cinco *cont* y sumarle *edge*, para saber la posición correcta.

VdEdgeHistogramSimilarity obtiene como parámetros de entrada dos punteros de tipo *VdEdgeHistogram* que apuntan a dos estructuras llenas con los datos de dicho descriptor. Los pasos que se realizan son los siguientes, se extraen uno a uno los valores del vector *Bincount* de cada uno de los dos descriptores y se hace la diferencia. Paso seguido se

suman todas las diferencias y se normaliza el resultado dividiéndolo entre el valor máximo que puede llegar a tener esa suma. Si lo analizamos detenidamente el caso en el que dos descriptores de este tipo sean menos similares, es el caso en el que todas las diferencias tengan el máximo valor, es decir valgan 8, ya que el valor más alto es 8 y el más pequeño 0. Por lo tanto el valor máximo será esa máxima diferencia por el número de veces que se hace esa diferencia, es decir ocho por ochenta lo que viene a ser 8 x 80 = 640.

Una vez normalizado, se obtiene un valor entre 0 y 1 que es inversamente proporcional a su similitud. Ya que si la suma da como resultado un número pequeño al dividirlo entre un número grande el resultado será pequeño, y eso es justamente lo contrario a lo que se busca. El objetivo es que si son muy similares o idénticos el resultado sea 1 y si son poco similares el resultado disminuya hasta 0. Por lo tanto se niega el resultado, restándole a la unidad el valor normalizado.

Todas las funciones y acciones explicadas anteriormente son guardadas en el archivo fuente *VdEdgeHistogram.c*

Finalmente y para poder ejecutar y visualizar los resultados que nos proporciona el conjunto de funciones implementadas para *VdEdgeHistogram* se crean dos archivos fuente que nos crearan binarios ejecutables al compilarlos, uno de extracción el cual recibe una imagen y calcula su descriptor, y otro que recibe dos descriptores y calcula su similitud (para ser utilizado en otro proyecto final de carrera del departamento de TSC, consistente en realizar una web de acceso e indexación de videos realizado por Mireia Luna). Para utilizar el segundo hay que guardar los descriptores en formato binario. Los archivos mentados se denominan *B_EDGEHISTOGRAM_EXTRACTB* y *B_EDGEHISTOGRAM_SIMILARITYB.*

5.2. REGION SHAPE

La forma de un objeto puede estar compuesta por una o varias regiones como también de zonas vacías como agujeros. Region Shape utiliza todos los píxeles dentro de una imagen, que puedan describir formas. No solo es capaz de describir objetos totalmente compactos como el (a) y (b) de la siguiente figura, sino que también tiene facultades para describir uno con agujeros o zonas disjuntas como los ejemplos (c), (d) y (e) de la figura.

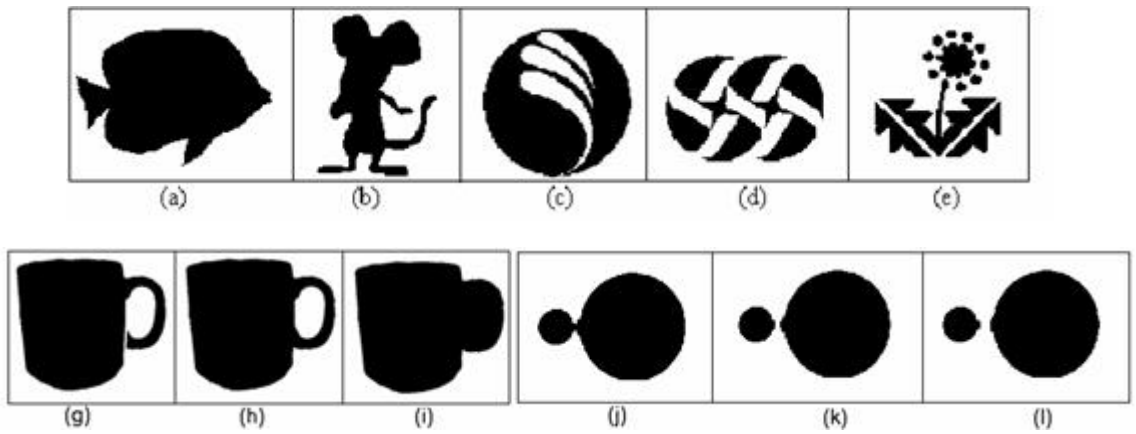


Fig. 12 Ejemplos de distintas figuras

Dicho descriptor también se caracteriza por ser robusto al ruido que se introduce inevitablemente en el proceso de segmentación, ya que produce la menor deformación en los bordes del objeto en su reconstrucción.

Los objetos de las viñetas (g), (h) y (i) son figuras muy similares que representan una taza. Lo único que las diferencia es el mango, en la imagen (g) el mango tiene una grieta en la parte inferior, mientras que en la (i) el agujero del mango no existe. La filosofía del descriptor considera que (g) y (h) son similares pero diferentes de (i), por no tener el orificio del mango.

En el segundo ejemplo se pretende mostrar una secuencia donde dos discos se van separando poco a poco. Con dicho descriptor los dos discos se consideran similares.

Otros atributos importantes a resaltar son su pequeño tamaño y un tiempo rápido de extracción. El total de datos de representación debe ocupar 140 bits y la extracción de características es sencilla con el fin de poder tener una bajo nivel de complejidad computacional ya que es conveniente para la identificación de objetos en procesado de videos.

Region Shape calcula los coeficientes ART (transformación angular radial) en 2D de orden n y m de una imagen de entrada, obteniendo una matriz de coeficientes denominados *ArtM*. Estos valores nos dan información sobre los objetos o formas que constituyen la imagen, de tal manera que cada coeficiente calculado junto a su correspondiente imagen base producida por la función base ART, proporciona información sobre la imagen inicial.

Para implementar el descriptor, en primer lugar definimos la estructura. En este caso, es una matriz llamada *ArtM* de 12 filas por 3 columnas, ya que los valores que se pretenden calcular dependen de dos parámetros, m que va de 0 a 11, y n de 0 a 2. Dicha estructura es denominada *VdRegionShape* y contiene valores tipo short.

La definición de la estructura junto con las cabeceras de todas las demás funciones y acciones necesarias en la especificación del descriptor son guardadas en el archivo de cabecera *VdRegionShape.h*

A continuación el siguiente procedimiento es la creación de las funciones comunes en todos los descriptores. Como el funcionamiento es el mismo que en las explicadas en el anterior descriptor y la única variación es el tipo de estructura con la que trabajan, con exponer sus nombres ya es suficiente. La función de alocar memoria se denomina *VdRegionShapeAlloc*, la de liberar memoria *VdRegionShapeFree*, la de copiar descriptores *VdRegionShapeCopy*, y las de lectura y escritura a disco *VdRegionShapeFromFile* y *VdRegionShapeToFile*, también creadas por nuestro tutor.

En el caso de la función de extracción de valores del descriptor si que es necesario explicarla un poco más. En el descriptor anterior al trabajar con una estructura formada por un vector, con introducirle un número como posición del valor que querías obtener ya era suficiente. Pero en este caso se trabaja con una matriz, por lo tanto, es el mismo procedimiento pero entrándole dos valores a parte del puntero tipo *VdRegionShape*. La primera posición marca la fila y la segunda posición la columna dentro de la matriz de valores.

Para *VdRegionShape* sucede lo mismo que para el anterior, tampoco es necesaria la función *Set* de escritura de valores al descriptor, ya que de ese proceso se encarga la función *VdRegionShapeCalculate*.

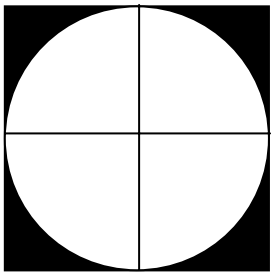
Seguidamente se implementan las funciones más importantes del descriptor *VdRegionShapeCalculate* y *VdRegionShapeSimilarity*.

VdRegionShapeCalculate lo que pretende es calcular un conjunto de coeficientes ART de orden n y m de la imagen de entrada.

ART es una transformada compleja en 2D sobre un disco de radio unidad en coordenadas polares, que depende de dos parámetros n y m. Su definición es la siguiente:

$$F_{nm} = \langle V_{nm}(\rho,\theta), f(\rho,\theta) \rangle$$
$$= \int_0^{2\pi} \int_0^1 V_{nm}^*(\rho,\theta), f(\rho,\theta) \rho d\rho d\theta$$

Tal y como podemos observar los coeficientes F_{nm} se calculan a través de la función base ART (V_{nm}) y de la función imagen en coordenadas polares (f(ρ,θ)). El producto de estas dos funciones se hace para todos los valores de la imagen dentro del disco de radio uno y finalmente se suman. Este procedimiento a su vez se repite para 12 funciones angulares (m) y 3 radiales (n).[12]



Si el cuadrado es la imagen de entrada, será calculado el producto anterior (V_{nm} * f) para todos los píxeles dentro del disco blanco de radio uno.

Fig. 13 Área de cálculo de F_{nm}

VdRegionShapeCalculate posee como parámetros de entrada un puntero tipo *struct image* que apunta a la imagen de la cual se quiere extraer la información de formas, y un puntero del tipo *VdRegionShape* que apunta a la estructura del descriptor que se pretende rellenar.

Primeramente se controla que tanto la imagen como el descriptor no valgan 0, de lo contrario se imprime un error por pantalla avisando de ello.

El siguiente paso es calcular el punto central de la imagen de entrada, lo que ayudará posteriormente a la conversión a coordenadas polares. A continuación se procede hacer un recorrido por los píxeles de la imagen para extraer de cada posición el valor y las coordenadas cartesianas correspondientes.

Una vez obtenidas dichas coordenadas lo primero es convertirlas a coordenadas polares, ya que son necesarias para trabajar con la función base ART. Para llevar a cabo esta conversión se crea una acción externa denominada *CartesianToPolar*.

CartesianToPolar necesita como parámetros de entrada para realizar su proceso, las coordenadas de *x* e *y* del centro de la imagen y las coordenada cartesianas *x* e *y* que queremos convertir. Entonces de lo que se trata es de calcular el módulo y el argumento de la posición indicada respecto del centro de la imagen. Primeramente debemos hacer un cambio de origen de coordenadas de la parte superior izquierda al centro de la imagen, y seguidamente realizar el cambio a polares. Para realizar el primer cambio, lo único que tenemos que hacer es restarle a las coordenadas de las que partimos, las del centro de la imagen. Seguidamente, a la segunda coordenada le cambiamos el signo (nótese que el eje y inicial apunta hacia abajo). Ahora ya podemos realizar el cambio a polares.

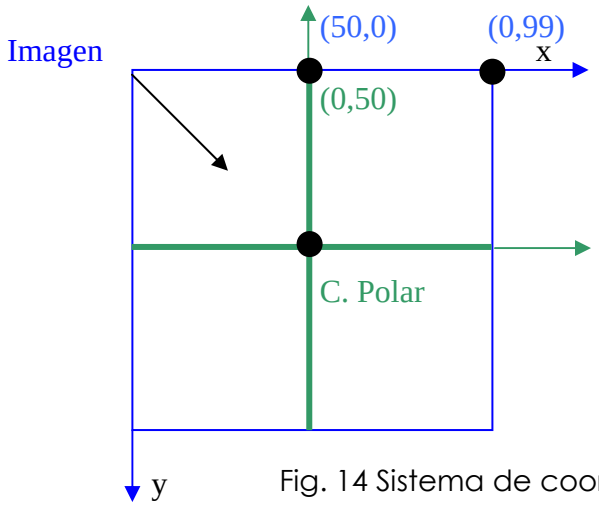


Fig. 14 Sistema de coordenadas

Tal y como podemos ver en la figura de la parte superior, en un mismo punto de la imagen las coordenadas son (50,0) respecto la parte superior izquierda. Mientras que si tomamos como origen de

coordenadas el centro de la imagen las coordenadas son (0,50). Para conseguir estos valores, las operaciones a realizar son:

(50,0)	-	(50,50)	=	(0,-50),
Coordenadas del punto respecto la parte superior izquierda		Coordenadas del centro de la imagen respecto la parte superior izquierda		Coordenadas respecto el centro de la imagen pero sin tener en cuenta la orientación del eje y

La solución es invertir el valor de y, con lo que obtenemos: (0,50).

Aclarado este concepto, se calculan las distancias en x e y y se normaliza el resultado dividiendo entre el valor máximo que corresponde a la posición del centro de la imagen. Lo único que falta entonces es calcular el módulo, elevando al cuadrado las dos distancias y haciendo la raíz cuadrada. Y calcular el argumento, haciendo la arco tangente del cociente de la distancia y entre la x. En el ejemplo anterior el resultado seria (50, 0)

Cabe comentar que esta función distingue entre dos excepciones por tal de evitar errores matemáticos. Se fija el argumento a 0 tanto si el módulo es igual a 0, como si el valor de la distancia en x es 0. La segunda condición se plantea ya que si esto ocurriera se estaría intentando hacer la arco tangente de un número infinito.

Tanto el módulo como el argumento son exportados a la función principal de *VdRegionShapeCalculate* a través de dos punteros inicializados fuera, que hacen el papel de parámetros de salida.

El siguiente paso consiste, una vez obtenidas las coordenadas polares, en calcular el valor de la función base ART, V_{nm}. Con ese fin se crea la función *ArtfunctionCalculate*.

V_{nm} como muestra su definición tiene parte real e imaginaria, las cuales son calculadas por separado.

$$V_{nm}(\rho,\theta) = A_m(\theta)R_n(\rho)$$

$$A_m(\theta) = \frac{1}{2\pi}\exp(jm\theta)$$

$$R_n(\rho) = \begin{cases} 1 & n = 0 \\ 2\cos(\pi n\rho) & n \neq 0 \end{cases}$$

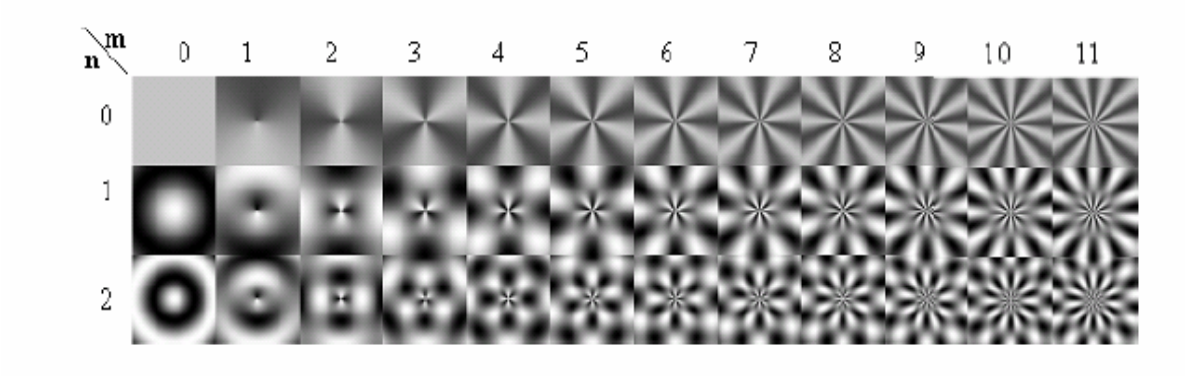


Fig. 15 Parte real de la función base ART

Para calcular dicha función son necesarios los valores del argumento, módulo, *n* y *m*, que son los parámetros de entrada. Y como salida también dispone de dos punteros los cuales apuntan al valor real e imaginario de V_{nm}.

Entonces, una vez calculada la función base ART V_{nm}, se debe multiplicar por el valor del píxel de la posición en la que se esta trabajando. Como se manipulan por separado las dos componentes, se multiplica dicho valor por las dos partes, real e imaginaria. Esto se realiza para todos los píxeles de la imagen que poseen un módulo menor o igual a 1, es decir que se comprendan dentro del disco de radio uno mostrado en la figura 13. Todos los demás píxeles son ignorados.

En este punto se habrá calculado un coeficiente F_{nm} correspondiente a un determinado valor de *n* y *m*, el cual debe ser almacenado en una tabla auxiliar de dimensiones 12x6(el número de columnas se dobla porque se deben guardar los dos componentes real e imaginario del coeficiente).

El siguiente paso consiste en hacer el módulo del coeficiente. Para después poder normalizar dicho valor con más facilidad. El módulo del coeficiente es divido entre el módulo del primer coeficiente calculado, cuando m es 0 y n es 0. Y finalmente, se debe cuantificar el valor obtenido utilizando la función creada especialmente con el nombre *Quantize*. *Quantize* obtiene el valor del coeficiente normalizado, busca cual es el intervalo de cuantificación que le pertenece y le asigna el número del 0 al 15 correspondiente.

Cabe decir que el coeficiente F_{00} solo se utiliza para la normalización de los demás coeficientes y que es igualado a 0 sin tener ningún valor de descripción. Si recordamos la definición del descriptor, mencionaba que el tamaño en disco total de los datos calculados debía ser 140 bits. Si tenemos 36 coeficientes (12x3), el tamaño que nos queda para cada uno es de 140/36=3.8 bits. Un espacio insuficiente para representar valores del 0 al 15 que necesitan 4 bits. Pero si en cambio despreciamos el coeficiente F_{00}, el cual solo nos ayuda en los cálculos, obtenemos que efectivamente tenemos bits suficiente: 140/35 = 4.

Para acabar, los 36 coeficientes de la imagen son almacenados en la matriz de la estructura del descriptor de entrada.

VdRegionShapeSimilarity calcula la similitud entre dos descriptores de entrada. Para ello recorre toda la matriz de *ArtM* [12][3] de los descriptores, extrae el valor de cada posición y realiza una cuantificación inversa. Para dicha cuantificación se utiliza la función

denominada *InverseQuantize*, la cual realiza una conversión del valor entre 0 y 15, calculado en el *VdRegionShapeCalculate*, a un valor decimal asociado. Una vez realizada dicha cuantificación los valores de los dos descriptores se restan y al resultado se le calcula el valor absoluto, para evitar cifras negativas. Esta operación se repite para todas las posiciones de la matriz con lo que al final se obtiene la suma total de las restas de todos los valores. Pero esta suma se debe normalizar porque necesitamos obtener un número entre 0 y 1, y obtener así, una medida de la similitud. Para ello, se divide el resultado total entre el valor máximo que se puede alcanzar. En este caso se corresponde a la mayor diferencia por el número de veces que se puede realizar dicha diferencia ((0.192540857-0.001763817)*35= 6.6771964). Finalmente, solo falta negar el resultado de la normalización, restándole a uno dicho resultado como se ha explicado en el anterior descriptor.

La implementación de todas las funciones y acciones explicadas quedan guardadas en el fichero llamado *VdRegionShape.c.*

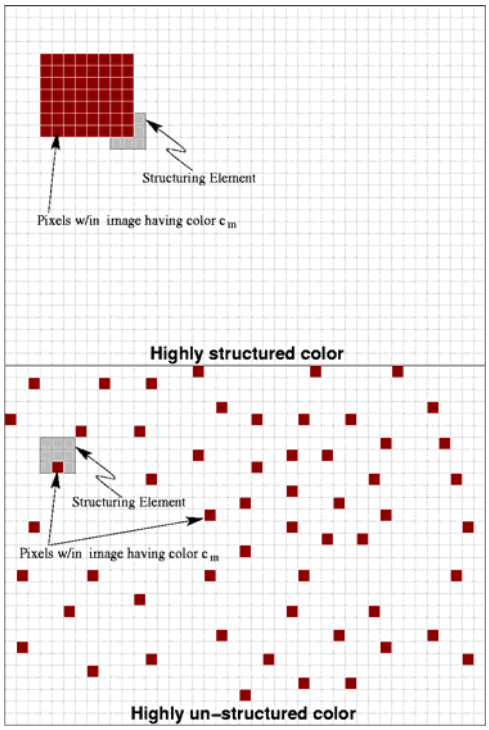
Aclarar que la declaración tanto de *Quantize*, como de *InverseQuantize* se almacena en un fichero cabecera llamado *VdRegionShape_priv.h*, ya que son funciones privadas.

Finalmente, para poder ejecutar y visualizar los resultados que nos proporciona el conjunto de funciones creadas para *VdRegionShape*, se crean dos archivos binarios como los explicados en el descriptor anterior. Los archivos se denominan: B_REGIONSHAPE_EXTRACTB y B_REGIONSHAPE_SIMILARITYB.

5.3. COLOR STRUCTURE

El Color Structure es un descriptor que se caracteriza por clasificar las imágenes teniendo en cuenta la distribución espacial de los colores. Podríamos decir que construye una especie de histograma de color de la imagen pero con la particularidad de que da relevancia, no al hecho de que un color se encuentre en un elevado número de píxeles, si no al hecho de que un color aparezca en muchas zonas de la imagen. Su funcionamiento consiste en dividir la imagen de partida en bloques de 8x8 píxeles e investigar dentro de cada uno de estos bloques los colores distintos que aparecen. Este descriptor ofrece una herramienta de comparación entre imágenes que en combinación con descriptores de análisis de contornos es muy utilizado en imágenes de paisajes.

De esta manera, este descriptor nos permitirá distinguir, por ejemplo, entre dos imágenes cuyo número de píxeles de un color es el mismo, pera la distribución de éste dentro de la imagen es distinta.



En la primera imagen del ejemplo, el color está centrado en una única zona, en cambio, en la segunda, aún existiendo el mismo número de píxeles de este color, su distribución es mucho más uniforme en todo el conjunto; así, al dividir la imagen en sub-imágenes, el número de estas que tendrán el color en su interior, será mucho mayor en la segunda que en la primera, y por lo tanto, también en el segundo caso, el valor del color en el histograma será también mayor.

Fig. 16 distintas distribuciones del color

El Color Structure que hemos implementado, trabaja en el espacio de color HMMD, y distingue 256 colores distintos. Así, el resultado de aplicarlo, será un vector de 256 posiciones, cada una de las cuales tendrá un valor proporcional al número de veces que se ha encontrado el color correspondiente en las sub-imágenes que se han analizado.

La implementación de este descriptor se ha llevado a cabo de la siguiente manera:

Definimos la estructura denominada *VdColorStructure*, la cual se constituye de dos partes, primero un valor llamado *numOfValues*, que indica la cantidad de colores que es capaz de distinguir el descriptor, en nuestro caso, la fijaremos en 256, pero se define de esta manera para posibilitar una posterior mejora y permitir que pueda ser variable. La segunda parte de la estructura es un vector denominado *value*, de tantas posiciones como el valor de *numOfValues* indique, en nuestro caso 256, que será donde se almacenen los valores del histograma.

La definición de esta estructura junto con la cabecera de todas las funciones que integran el descriptor, se sitúan en el archivo de cabeceras *VdColorStructure.h*.

Seguidamente se construyen las funciones comunes del descriptor. La función de alocar memoria para la estructura del descriptor *VdColorStructureAlloc*, la acción de liberar memoria *VdColorStructureFree*, las funciones de extracción de valores de descriptor *VdColorStructureGetNumOfValues*, que devuelve el valor, como indica su nombre, de la variable *numOfValues*, y *VdColorStructureGetValue* que recibe un entero entre 0 y 256 y devuelve el valor de dicha posición del vector *value*. La acción de copiado de descriptores *VdColorStructureCopy*, y finalmente las

funciones de lectura y escritura a disco del descriptor *VdColorStructureToFile* y *VdColorStructureFromFile*.

Y finalmente las dos funciones que llevan el peso del descriptor, *VdColorStructureCalculate* y *VdColorStructureSimilarity*.

VdColorStructureCalculate, es una función que recibe las imágenes correspondientes a los tres componentes RGB de una imagen original, y calcula los valores del histograma. Se trata de una función que puede dividirse en tres bloques principales, en el primero se cambiará del espacio de color original RGB al espacio de color HMMD, en el segundo se hará una cuantificación no uniforme del valor de los colores según define MPEG-7, y en el tercero, finalmente es en el que se recorre la imagen y se extrae el histograma.

Antes de hablar del cambio de espacio de color, vamos a definir el espacio de color HMMD. Este espacio, caracteriza los colores, con 4 valores:

- Hue: Valor que identifica el tono de color, es el mismo que el del espacio HSV
- Max: De alguna manera, indica cuanto negro contiene el color, dando una idea de la cantidad de oscuridad.
- Min: Igual que el valor Max pero con la cantidad de color blanco
- Diff: Da una idea del nivel de gris de la imagen y de la proximidad al color puro.

Para pasar del espacio RGB al HMMD, se utilizan unas pautas ya definidas que se muestran a continuación:

Max = máximo entre los valores de R, G y B
Min = mínimo entre los valores de R,G y B

El valor de hue depende de:

si Max = R y G ≥ B , entonces Hue=60x(G-B) ÷(Max-Min)

Si Max=R y G<B, entonces Hue=360+60x(G-B) ÷(Max-Min)

Si Max=G, Hue=60x(2.0+(B-R) ÷(Max-Min))

Si no es ninguno de los casos anteriores, Hue=60x(4.0+(B-R) ÷(Max-Min))

Una vez cambiado el espacio, el estándar nos obliga a realizar una cuantificación no uniforme para cada valor de color. Esta cuantificación la hemos implementado en una función especial, VdColorStructureCalculateBins, que recibe los valores de Diff, Hue y Sum y devuelve el nuevo número que identifica al color. Para conseguirlo, primero se divide el espacio HMMD en 5 subespacios, definidos por fronteras que se determinan a partir del valor de Diff. Los intervalos que delimitan los subespacios son los siguientes: [0,6), [6, 20), [20, 60), [60, 110) y [110, 255]. Seguidamente, cada color de cada subespacio se quantifica uniformemente según los valores de Hue y Sum, teniendo en cuenta los niveles de cuantificacion que nos define la tabla siguiente:

Subespacio	Número de niveles de cuantificación	
	256	
	Hue	Sum
0	1	32
1	4	8
2	16	4
3	16	4
4	16	4

Tabla 2: cuantificación no uniforme de 256

De esta manera conseguimos 32 niveles en el sub-espacio 0, 32 en el 1, y 64 niveles en cada uno de los otros tres espacios, es decir, un total de 256 niveles, que se corresponden con las 256 posiciones del vector *value* del descriptor. Esta cuantificación es la que realiza la función *VdColorStructureCalculateBins*.

Llegados a este punto, lo único que falta por hacer, es dividir la imagen en bloques de 8x8 píxeles, y mirar cuantos colores distintos hay dentro de cada sub-imagen. Una vez los conocemos, incrementamos su valor correspondiente en el vector resultado. En nuestro caso, para conseguirlo, utilizamos un vector auxiliar de 256 posiciones, recorremos cada uno de los bloques y colocamos un 1 en la posición correspondiente a los colores del vector auxiliar. Después este vector lo sumamos al vector resultado, y de esta manera lo vamos

incrementando y construyendo. En la imagen siguiente, se puede ver este proceso de manera esquemática:

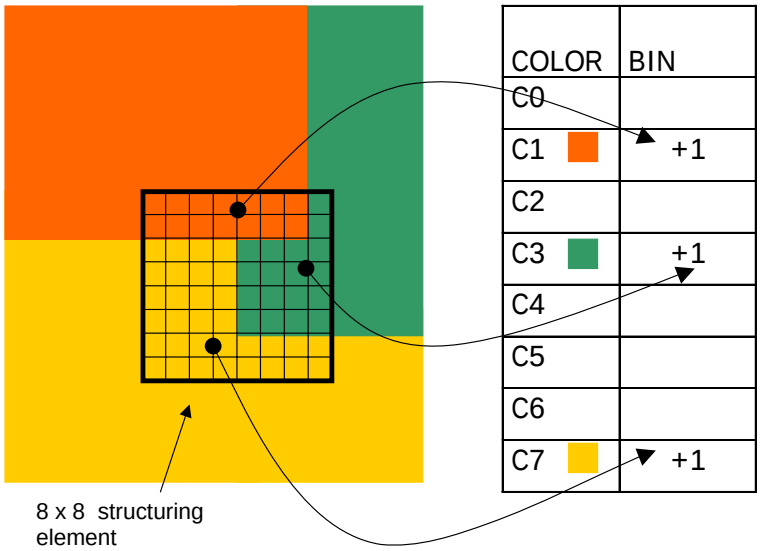


Fig.17 método de extracción del descriptor

Finalizado este proceso, tenemos el vector value del descriptor relleno con los valores correspondientes, los cuales dependerán del tamaño de la imagen, o lo que es lo mismo, del número de sub-imágenes de 8x8 píxeles. Es por este motivo que tenemos que cuantificar los valores entre 0 y 255. Esta cuantificación consta de dos pasos, una primera normalización de los valores entre 0 y 1, y seguidamente según este valor, se delimitan 6 intervalos y a cada uno de ellos se le asigna un número distinto de niveles. En la tabla siguiente se puede observar esta información:

intervalo	Niveles
[0 ; 0,000000001)	1
[0,000000001 ; 0,037)	25
[0,037 ; 0,08)	20
[0,08 ; 0,195)	35
[0,195 ; 0,32)	35
[0,32 ; 1]	140

Tabla 3: cuantificación no uniforme de los valores finales del descriptor

De esta manera conseguimos el resultado final de la función *VdColorStructureCalculate*.

Finalmente queda crear la función de similitud *VdColorStructureSimilarity*, a la cual le pasamos dos punteros a descriptores *Color Structure* y ella devuelve un número entre 0 y 1, 0 para similitud nula y 1 para igualdad. Para calcularlo, simplemente restamos los dos vectores *value*, y luego sumamos todos los valores del vector diferencia en valor absoluto. Una vez tenemos el resultado simplemente debemos normalizar entre 0 y 1, y restarle este valor a 1 para actuar tal y como dice el estándar.

Finalmente se guardan todas estas funciones en el archivo *VdColorStructure.c*.

Ahora, ya solo es necesario crear los ejecutables de extracción y similitud: *B_VD_COLORSTRUCTURE* y *B_VD_COLORSTRUCTURE_SIMILARITY* respectivamente. Al primero le entramos las componentes RGB de una imagen y un nombre para el descriptor, y con la ayuda de la función *VdColorStructureCalculate* calcula el descriptor de la imagen y lo almacena en el formato definido en la función *VdColorStructureToFile*, y al segundo, le entramos dos descriptores, en formato binario e imprime la similitud por pantalla.

5.4. HOMOGENEOUS TEXTURE

La textura, representa la regularidad de una imagen en lo que refiere a direccionalidad, tosquedad, regularidad del patrón, etc. El descriptor Homogeneous Texture, provee a sus usuarios la capacidad de realizar comparaciones imagen a imagen, realizando una descripción de la textura de estas imágenes a través del valor de la energía y la desviación energética de la señal, extraídas de la distribución frecuencial de esta.

El espacio frecuencial a partir del cual se extraen las características de textura de la imagen, se parte en regiones de 30º en dirección angular y en 5 regiones de octava en dirección radial. De esta manera se obtienen 30 canales como muestra la figura siguiente:

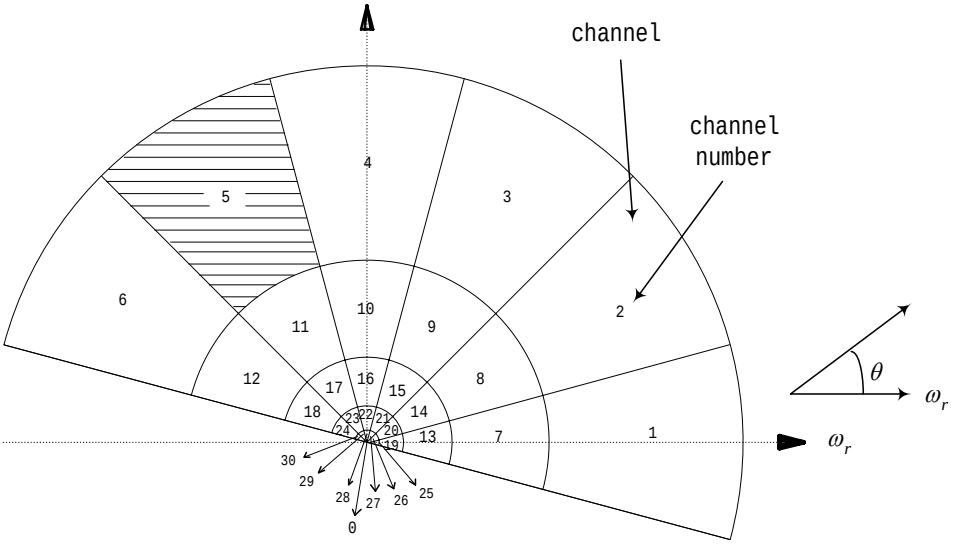


Fig.18 División del espacio frecuencial

En el espacio frecuencial normalizado (0≤w≤1). Las frecuencias centrales de los distintos canales se encuentran en los ángulos θ_r=30º*r, dónde r es el índice angular que varia de 0 a 5. El ancho angular de estos canales por tanto es de 30º. En dirección radial, las frecuencias

centrales serán $w_s=w_0*2^{-s}$, donde s es el índice radial y varia de 0 a 4, y w_0 es la frecuencia central mayor especificada como ¾.

Finalmente cada canal se caracteriza por un valor de 1 a 30 que corresponde al resultado de hacer 6xs+r+1 con sus índices correspondientes.

A partir de aquí, es necesario operar la imagen con los filtros de Gabor, obteniendo treinta filtrados distintos con la operación:

$$H_i(\omega,\theta) = G_{P_{s,r}}(\omega,\theta)\cdot|\omega|\cdot F(\omega,\theta) \text{ donde } i(=6\times s+r+1)$$

Donde H es el resultado de filtrar, G es el filtro de Gabor correspondiente al canal con índices s y r, y F es la transformada de Fourier de la imagen.

A partir de aquí, los valores de la energía y la desviación se obtienen haciendo:

$$e_i = \log_{10}[1+p_i]$$

$$d_i = \log_{10}[1+q_i]$$

La función de Gabor que se aplica para realizar la operación anterior es:

$$G_{P\,s,r}(\omega,\theta) = \exp\left[\frac{-(\omega-\omega_s)^2}{2\sigma_{\rho_s}^2}\right]\cdot\exp\left[\frac{-(\theta-\theta_r)^2}{2\sigma_{\theta_r}^2}\right]$$

Los valores dependientes del canal donde nos encontramos, se presentan en las tablas que hay a continuación.

Radial index (s)	0	1	2	3	4
Center frequency (ω_s)	$\frac{3}{4}$	$\frac{3}{8}$	$\frac{3}{16}$	$\frac{3}{32}$	$\frac{3}{64}$
σ_{ρ_s}	$\frac{1}{4\sqrt{2\ln 2}}$	$\frac{1}{8\sqrt{2\ln 2}}$	$\frac{1}{16\sqrt{2\ln 2}}$	$\frac{1}{32\sqrt{2\ln 2}}$	$\frac{1}{64\sqrt{2\ln 2}}$

Tabla 4: Valores dependientesde la dirección radial

Angular index (r)	0	1	2	3	4	5
Center frequency (θ_r)	0°	30°	60°	90°	120°	150°
Angular bandwidth	30°	30°	30°	30°	30°	30°
σ_{θ_r}	$\frac{30^\circ}{2\sqrt{2\ln 2}}$	$\frac{30^\circ}{2\sqrt{2\ln 2}}$	$\frac{30^\circ}{2\sqrt{2\ln 2}}$	$\frac{30^\circ}{2\sqrt{2\ln 2}}$	$\frac{30^\circ}{2\sqrt{2\ln 2}}$	$\frac{30^\circ}{2\sqrt{2\ln 2}}$

Tabla 5: Valores dependiendo de la dirección angular

La implementación de este descriptor se ha llevado a cabo de la siguiente manera:

Definimos la estructura denominada ***VdHomoTexture***, la cual se constituye de dos partes, dos vectores de treinta posiciones, en el primero están los valores correspondientes a la energía, obtenidos con el proceso explicado anteriormente, y se llama ***energy***, el segundo cuyo nombre es ***energy_desv*** se encuentran los valores de la desviación energética.

La definición de esta estructura junto con la cabecera de todas las funciones que integran el descriptor, se sitúan en el archivo de cabeceras *VdHomoTexture.h*.

Ahora, construimos las funciones comunes del descriptor, para alocar memoria *VdHomoTextureAlloc*, la función para liberar esta memoria previamente alocada *VdHomoTextureFree*, las funciones *VdHomoTextureGetEnergy* y *VdHomoTextureGetDesviation*, que reciben un entero entre 0 i 29 y devuelven el valor de la posición correspondiente del vector *energy* o *energy_desv* respectivamente, la función de copia *VdHomoTextureCopy*, y finalmente las funciones de cálculo de descriptor *VdHomoTextureCalculate*, cálculo de distancia entre dos descriptores *VdHomoTextureDistance* y cambio de coordenadas de rectangular a polar *CartesianToPolar* que vamos a comentar más extensamente.

Para *VdHomoTexture* , tampoco es necesaria la función *Set* de escritura de valores para ninguno de los datos del descriptor. Y las funciones *VdHomoTextureToFile* y *VdHomoTextureFromFile*, también nos han sido proporcionadas por el tutor Javier Ruiz Hidalgo.

La función de cálculo del descriptor, recibe una imagen y el nombre que le daremos al descriptor. Seguidamente, construye los vectores con los valores y posiciones adecuadas a los dos cuadros anteriores para facilitar el cálculo de los filtros de Gabor. El siguiente paso, es definir y alocar memoria para 5 imágenes del mismo tamaño que la de entrada:

Ima_F: que contendrá los valores reales de la transformada de fourier
Ima_F_i: que contendrá los valores imaginarios de dicha transformada
Ima_G: contendra los valores del filtro de gabor correspondiente para cada píxel

Ima_H: contendrá los valores reales de la operación de filtrado
Ima_H_i: contendrá los valores imaginarios del filtrado

Cabe decir que para poder realizar la transformada de fourier de una imagen, se recurre a funciones de la librería L_MATH de Soft-Image, en concreto una función llamada fft2, con lo cual la librería en cuestión se debe incluir en el archivo. El hecho de utilizar estas funciones, obliga a que las imágenes deban tener unas dimensiones múltiples de 2.

A partir de aquí el procedimiento a seguir es el mismo para cada uno de los filtros, que se identifican con la ayuda de los índices r y s: para cada uno de los píxeles de la imagen (hacemos un recorrido), se cambian sus coordenadas a polares y se le aplica la función de Gabor con los correspondientes valores de r y s, el resultado lo guardamos en la imagen ima_G, y lo utilizamos para operar con el valor del módulo y píxel correspondiente de la transformada de fourier, consiguiendo así el valor del filtrado del píxel determinado que irá almacenándose en la imagen ima_H. Una vez rellena esta última, hacemos dos recorridos dentro de ella, en el primero calculamos el valor de la energía que se situará en la posición del vector energy que marquen los índices r y s, y en el segundo hacemos lo mismo pero con la desviación energética y el vector energy_desv.

Una vez concluido el proceso se van incrementando los índices r y s para conseguir los 30 filtrados necesarios. Y finalmente se liberan las memorias ocupadas por las imágenes.

En este descriptor, debido a problemas de tiempo, la función de similitud, que según el estándar debía ceñirse a unas determinadas cuantificaciones, no se ha podido implementar, y en su lugar se ha creado una función *VdHomoTextureDistance*, que lo que hace, es simplemente partir de dos descriptores que se le introducen, calcular la

distancia entre los vectores *energy* de los dos descriptores y la distancia de los vectores *energy_desv*, sumarlas y dividirlas entre dos. La distancia entre vectores se consigue realizando un sumatorio con el valor absoluto de la diferencia de los valores correspondientes de cada vector. De esta manera si el resultado es 0.0, significa que los dos descriptores són iguales i cuanto más se aleje de este valor menos se parecen.

Finalmente en este descriptor, también se usa la función *CartesianToPolar* que ya se ha utilizado en el Region Shape, ya que los cálculos realizados, requieren el uso de las coordenadas polares y no cartesianas de los píxeles. Todas estas funciones en el archivo *VdHomoTexture.c*.

Ahora, creamos los binarios de extracción y distancia: *B_VD_HOMOTEXTURE_EXTRACT* y *B_VD_HOMOTEXTURE_DISTANCE* respectivamente. Al primero le entramos una imagen y un nombre para el descriptor, y con la ayuda de la función *VdHomoTextureCalculate* calcula el descriptor de la imagen, y al segundo, le entramos dos descriptores i imprime la distancia por pantalla.

Aquí también es necesario guardar el descriptor en formato binario al disco para poder usarlo en la función de distancia.

6. PRUEBAS

6.1.EDGE HISTOGRAM

Por tal de comprobar el correcto funcionamiento del descriptor Edge Histogram a continuación se exponen y se analizan las pruebas realizadas.

La primera prueba es comprobar que al procesar la extracción del descriptor con una imagen homogénea el resultado es de 0 en todas las posiciones de Bincount:

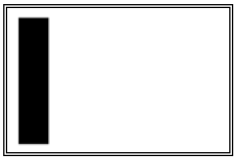
Imagen homogénea

Resultado:

```
Edge Histogram Descriptor:
BinCount[0] = 0
BinCount[1] = 0
BinCount[2] = 0
BinCount[3] = 0
BinCount[4] = 0
BinCount[5] = 0
BinCount[6] = 0
BinCount[7] = 0
…..
```

Este resultado obtenido es debido a que una imagen al ser homogénea no dispone de ningún tipo de transición ni de contorno que pueda ser detectado.

A continuación veamos que pasa al probar con una imagen con un rectángulo en la parte izquierda:



Resultado:

```
Edge Histogram Descriptor:
BinCount[0] = 1 BinCount[20] = 1 BinCount[40] = 1 BinCount[60] = 1
BinCount[1] = 0 BinCount[21] = 0 BinCount[41] = 0 BinCount[61] = 0
BinCount[2] = 2 BinCount[22] = 1 BinCount[42] = 1 BinCount[62] = 2
BinCount[3] = 2 BinCount[23] = 1 BinCount[43] = 1 BinCount[63] = 2
BinCount[4] = 1 BinCount[24] = 1 BinCount[44] = 1 BinCount[64] = 1
```

0—4 (0,0)	5—9 (0,1)	10—14(0,2)	15—19(0,3)
20—24(1,0)	25—29(1,1)	30—34(1,2)	35—39(1,3)
40—44(2,0)	45—49(2,1)	50—54(2,2)	55—59(2,3)
60—64(3,0)	65—69(3,1)	70—74(3,2)	75—79(3,3)

Intervalos de BinCount / número de bloque

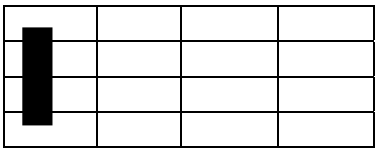


Imagen dividida en bloques

Como podemos observar los valores obtenidos son correctos, ya que marcan los contornos de los cinco tipos en la zona de la izquierda de la imagen. Si analizamos los valores de cada una de las posiciones más detalladamente, podemos ver que en los bloques (0,0) y (3,0) aparece un 1 en bordes verticales y no direccionales, un 0 en bordes horizontales, y un 2 en bordes de 45 y 135º. Lo que nos quiere decir que al filtrar la imagen se han encontrado muchos más puntos en las direcciones de 45 y 135, que en la dirección vertical y la dirección horizontal. Veamos porque:

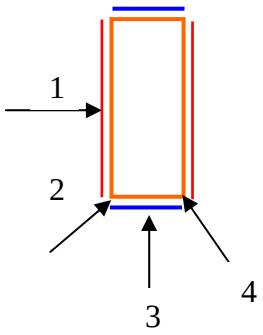


Fig. 19 Direcciones de detección de contornos.

El esquema de la izquierda quiere mostrar en que direcciones busca las transiciones el sistema para cada contorno. La flecha número 1 es la correspondiente al contorno vertical, ya que si se desea encontrar una transición, es decir un cambio de color, de contornos verticales se debe recorrer en sentido horizontal. Esto paso para todos los casos la dirección de recorrido es justo la dirección perpendicular a la que buscamos.

Para contornos horizontales seria la número 3, para 45º la número 4 y para 135º la 2.

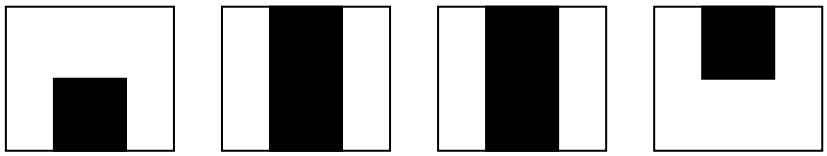
De este modo este esquema nos facilita la tarea de entender que cantidad de píxeles son detectados en cada caso. En el caso del contorno vertical son los de color rojo, en el horizontal son los de color azul y para los otros dos tipos son los de color naranja.

Por lo tanto también por este parte podemos asegurar que el resultado es correcto, ya que como vemos los píxeles de bordes horizontales son minoritarios frente a los verticales, al ser el rectángulo más alto que ancho. Además es fácil justificar porque hay muchos más píxeles en direcciones diagonales, porque como vemos en el esquema estas direcciones acaparan todos los píxeles del contorno de la figura.

Cabe mencionar que aunque el valor BinCount sea igual a 0, no significa que no hayan sido encontrado píxeles de contornos de ese tipo, sino que se han encontrado un número relativamente pequeño y al cuantificar este valor es considerado 0.

Finalmente aclarar el porque de que en los bloques del extremo superior e inferior izquierdo de la imagen los valores varían respecto de los bloques del centro. La variación precisamente se encuentra en una disminución de un punto en los contornos de 45º y 135º, ¿a que es debido?

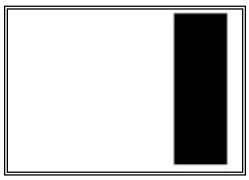
Si se observa los bloques por separado se puede entender que en los bloques del interior los contornos horizontales desaparecen, ya que la figura ocupa el espacio de arriba a abajo del bloque. Con lo cual al desparecer los contornos horizontales, disminuyen los de direcciones diagonales. Provocando que los bordes en sentido diagonal se igualen a los de sentido vertical.



Bloque (0,0) Bloque (1,0) Bloque (2,0) Bloque (3,0)

Fig. 20 Desglosamiento por bloques de la imagen

A continuación otro ejemplo similar, pero en este caso la figura se encuentra en el lado contrario, el derecho.



Resultado:

Edge Histogram Descriptor:
BinCount[10] = 1 BinCount[30] = 1 BinCount[50] = 1 BinCount[70] = 0
BinCount[11] = 0 BinCount[31] = 0 BinCount[51] = 0 BinCount[71] = 0
BinCount[12] = 1 BinCount[32] = 1 BinCount[52] = 1 BinCount[72] = 1
BinCount[13] = 1 BinCount[33] = 1 BinCount[53] = 1 BinCount[73] = 1
BinCount[14] = 1 BinCount[34] = 1 BinCount[54] = 1 BinCount[74] = 1
BinCount[15] = 1 BinCount[35] = 1 BinCount[55] = 1 BinCount[75] = 1
BinCount[16] = 1 BinCount[36] = 0 BinCount[56] = 0 BinCount[76] = 0
BinCount[17] = 1 BinCount[37] = 1 BinCount[57] = 1 BinCount[77] = 1
BinCount[18] = 1 BinCount[38] = 1 BinCount[58] = 1 BinCount[78] = 1
BinCount[19] = 1 BinCount[39] = 1 BinCount[59] = 1 BinCount[79] = 1

0—4 (0,0)	5—9 (0,1)	10—14(0,2)	15—19(0,3)
20—24(1,0)	25—29(1,1)	30—34(1,2)	35—39(1,3)
40—44(2,0)	45—49(2,1)	50—54(2,2)	55—59(2,3)
60—64(3,0)	65—69(3,1)	70—74(3,2)	75—79(3,3)

Intervalos de BinCounts / número de bloque

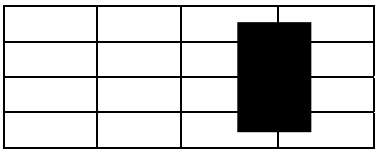


Imagen dividida en bloques

Con esta imagen se observa que sucede algo parecido. Los Bincounts con valores significativos pertenecen a los bloques de más a la derecha, que es precisamente donde se encuentra el objeto. En este caso lo que ocurre es que al ser el rectángulo más ancho, ocupa dos bloques de la imagen, en los cuales se detectan los bordes.

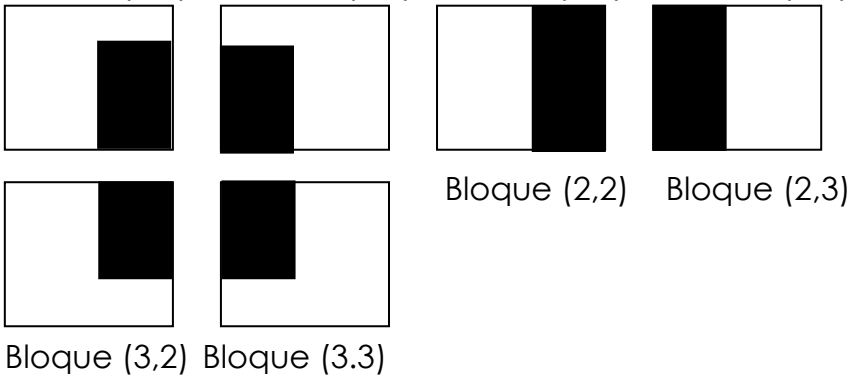
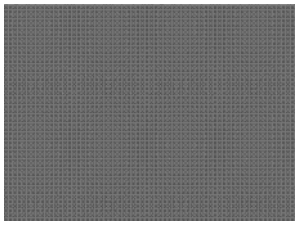


Fig. 21 Desglosamiento por bloques de la imagen

Como consecuencia el número de Bincount referido a contornos verticales y diagonales disminuye, ya que en cada bloque se detecta solo contornos verticales de un lado del rectángulo mientras que en el caso anterior se detectaban los dos lados.

Finalmente exponer un ejemplo como contraste al de la imagen homogénea.

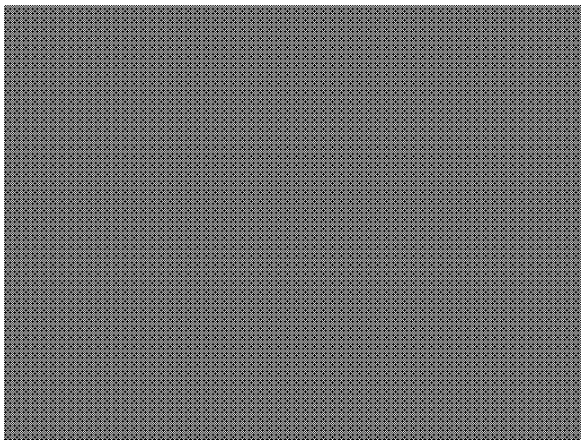


Edge Histogram Descriptor:
BinCount[0] = 6 BinCount[20] = 6
BinCount[1] = 6 BinCount[21] = 6
BinCount[2] = 8 BinCount[22] = 8
BinCount[3] = 8 BinCount[23] = 8
BinCount[4] = 8 BinCount[24] = 8
…..

Resultado:

Esta imagen esta completamente repleta de un estampado pequeño en forma de flor, por lo que el resultado obtenido son números muy elevados de contornos.

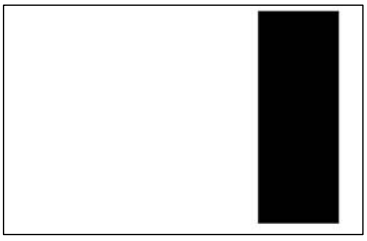
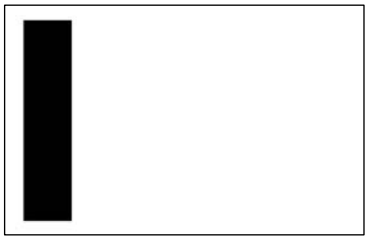
Una vez entendida la salida del descriptor, el siguiente paso es comprobar que el binario de similitud también proporciona valores coherentes. Las pruebas realizadas son las siguientes:



La similitud entre una imagen con estampado pequeño y una homogénea es 0.1


La similitud en Akiyo y Foreman es de 0.696875


La similitud entre estas dos imágenes consecutivas de un video de Akiyo es 0.996875

La similitud entre las dos imágenes analizadas anteriormente es 0.918750

El primer ejemplo nos muestra la similitud entre dos imágenes extremas, una que no tiene ningún contorno con otra que esta completamente llena de contornos, por lo tanto el resultado es el esperado, un nivel muy bajo de similitud. Recordar que los valores proporcionados siempre están comprendidos entre 0 y 1, siendo 1 y el máximo y 0 el mínimo de semejanza entre las dos imágenes.

En el segundo caso se realiza la similitud entre dos imágenes típicas del procesado de imagen, las cuales tiene distribución de contornos diferente pero no son tan opuestas como las anteriores. Un 0.696875 es un valor aceptable.

El tercer ejemplo expone la similitud entre dos imágenes casi idénticas que corresponden a dos frames seguidos de una secuencia, con lo que la única diferencia más o menos visible es la posición de la boca de la presentadora. De este modo el resultado cumple las expectativas esperadas, ya que se corresponde con un valor muy cercano a 1, 0.996875.

Finalmente el tercer ejemplo, el cual utiliza las dos imágenes ya analizadas, nos expone un valor bastante alto pero menor a los anteriores.

Si podría pensar que los valores resultantes son demasiado altos, pero si lo analizamos detenidamente tiene su motivo razonable. Cuando normalizamos el valor de los píxeles que forman los contornos, lo hacemos dividiendo entre un valor muy grande. Como normalmente al procesar imágenes comunes los valores que se obtienen son valores bajos, existe un rango de valores altos que raramente se llegan a obtener. Por consiguiente el resultado final de los cálculos nos da valores bastante grandes.

6.2. REGION SHAPE

Las pruebas realizadas con el descriptor Region Shape, con el fin de la verificación de su correcto funcionamiento, son las que se muestran a continuación:

Sabiendo que el procedimiento que sigue el descriptor es el de obtener un coeficiente que indica la proporción existente de la imagen base en la imagen principal, se debe observar un concepto antes de analizar los resultados.

Como se puede visualizar en la figura 22 las imágenes base ART nos marcan en sentido horizontal la cantidad de transiciones de la imagen, mientras que en el sentido vertical nos muestra la intensidad de

las transiciones, es decir, a medida que bajamos, las transiciones son mas bruscas, se unen color mas opuestos.

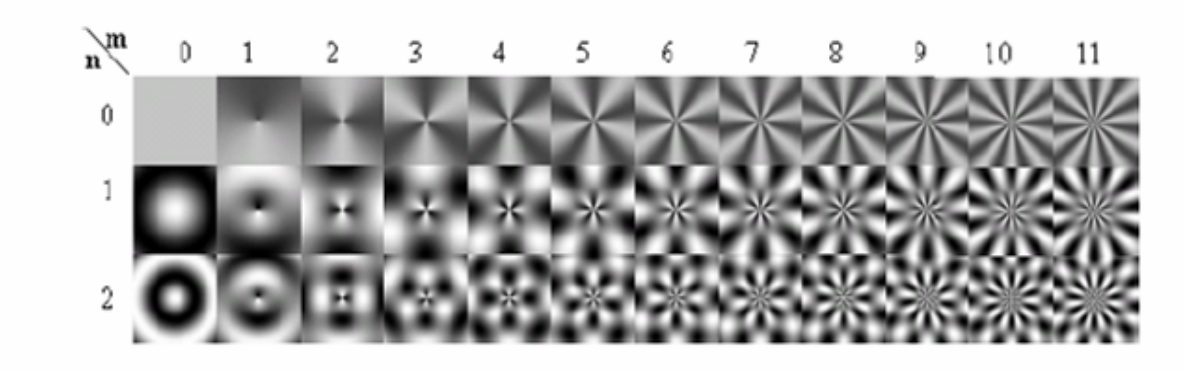


Fig. 22 Imágenes base ART

Veamos los resultados obtenidos:

Imagen de entrada

Coeficientes obtenidos

0	15	15	9	12	15	8	9	3	10	1	10
15	15	14	2	14	15	12	2	7	10	0	8
8	9	10	7	1	7	8	8	8	8	5	3

Tal y como muestra la matriz de coeficientes, los valores más altos se centran en la primera fila de valores. Esto es debido a que la imagen de entrada no contiene muchas transiciones, por lo que sus componentes más elevados son los que marcan poca variación en la imagen, es decir más zonas homogéneas. Como por ejemplo las imágenes base de más a la izquierda.

También cabe prestar atención a los valores relativamente grandes en el final de la tercera fila, debido a que las pocas transiciones que posee la imagen son entre negro y blanco, colores muy contrastados.

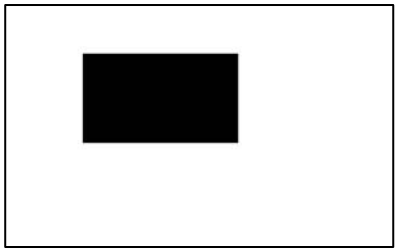
0	15	11	15	6	12	6	12	4	11	3	9
15	15	14	13	6	9	7	11	5	11	5	8
3	9	14	13	8	5	5	3	3	0	3	4

0	15	14	13	3	12	5	12	3	9	0	8
15	15	15	8	8	8	7	11	7	2	0	3
14	12	8	13	10	3	2	4	7	7	2	2

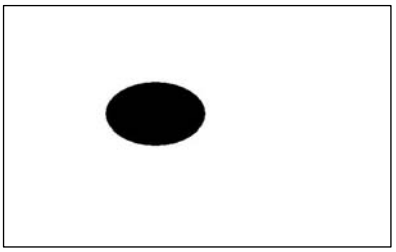
Observar como al procesar la imagen de Foreman los valores de la tabla de resultados están más distribuidos que en la imagen homogénea.

En la imagen de Foreman se puede apreciar el hecho de que las transiciones son menos pronunciadas, ya que se componen de diferentes tonos de gris. Esto queda reflejado en el resultado, con valores más pequeños en la última fila, la cual se encarga de denotar los contornos más pronunciados.

Respecto la imagen homogénea, observar como los valores más elevados se concentran en la esquina superior izquierda de la tabla de resultados, y van disminuyendo a medida que nos dirigimos a la parte inferior derecha. Esto es debido, como se ha explicado con anterioridad, a que las zonas homogéneas se reflejan en los resultados de la zona superior izquierda de la tabla.



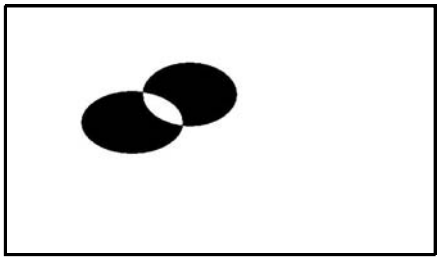
0	15	15	15	15	15	12	10	7	8	4	10
15	15	15	15	15	15	15	12	11	10	5	11
15	9	15	15	11	15	14	6	10	12	7	7

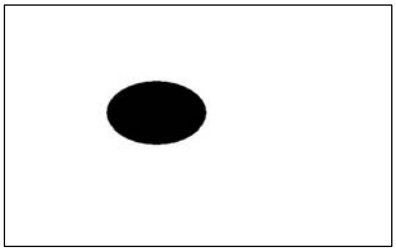


0	15	15	6	14	15	9	9	1	7	3	9
15	15	14	8	15	15	13	4	6	9	3	9
14	12	15	14	9	13	9	7	11	12	6	4

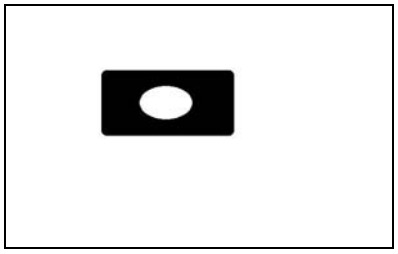
En estos dos ejemplos se puede observar los resultados obtenidos al procesar un objeto rectangular y uno elíptico.

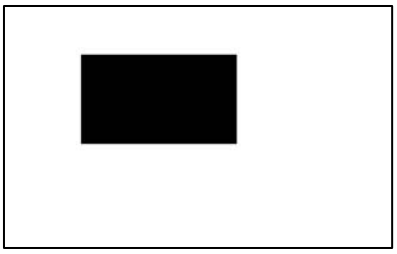
La similitud obtenida entre diferentes imágenes ha sido la siguiente:





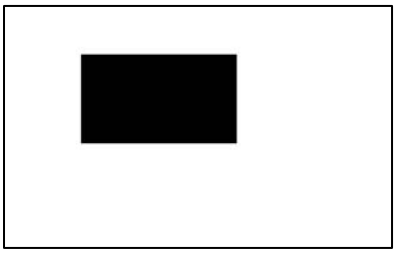
La similitud entre estas dos imágenes con objetos elípticos es de 0.908299

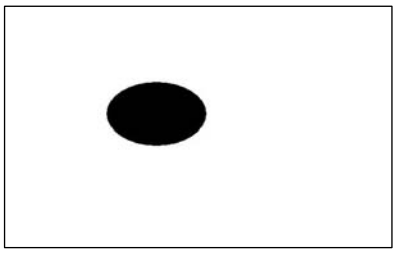




La similitud entre estas dos imágenes con objetos rectangulares es de 0.883386

La similitud entre estas dos imágenes de Foreman es de 0.968963





La similitud entre un rectángulo y una elipse es de 0.827056

Analizando los resultados se verifica el concepto que se explica en la definición del descriptor. Region Shape considera menos similares dos figuras parecidas, pero con la diferencia que una de ellas posee un agujero, como es el caso del segundo ejemplo de los rectángulos. Y mas similares dos imágenes donde se tiene el objeto parecido pero en otro ángulo, como en el caso del tercer ejemplo de la cara de foreman.

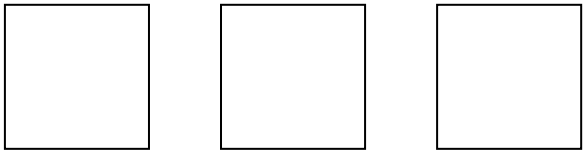
Además también se puede observar como la similitud es más alta entre dos objetos de la misma forma, como el caso de los dos rectángulos, que de diferentes formas, como en último ejemplo.

6.3. COLOR ESTRUCTURE

Para comprobar el correcto funcionamiento del Color Structure, lo más sencillo es entrarle tres imágenes homogéneas como parámetros a la función de extracción, de manera que, simulamos una imagen en color también homogénea. El resultado en este caso debe ser un vector en el

que todas las posiciones tengan valor 0, exceptuando la correspondiente al color en cuestión que tiene que valer 255, ya que el color aparece en todas las sub-imágenes en que hemos dividido la original para calcular el descriptor.

Primero de todo probamos introduciéndole tres imágenes blancas, de manera que en realidad es como si la imagen original fuera una imagen blanca:



R G B

El resultado que obtenemos es:

ColorStructure Descriptor:
numOfValues = 256

El vector ***value*** es un vector que tiene todos los valores igual a 0, excepto el 31, que vale 255, y que si observamos en la cuantificación no uniforme, efectivamente equivale al blanco.

Del mismo modo podemos probar con una imagen negra, es decir una imagen con los tres componentes a 0. Para eso le entramos tres imágenes negras y el resultado es idéntico al anterior pero con la característica de que la posición del vector que ocupa el valor distinto a 0 e igual a 255 es la 0.

R G B

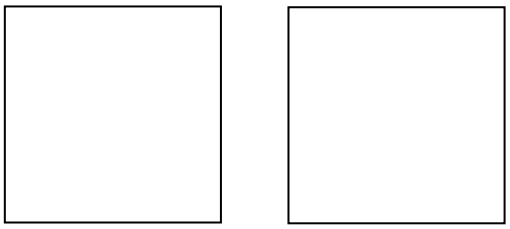
Del mismo modo, podemos crear configuraciones distintas de imágenes homogéneas de manera que vamos obteniendo resultados como los anteriores pero donde la posición del vector que vale 255 va moviéndose. En el anexo están las pruebas del blanco, del negro y de la combinación blanco negro blanco.

R G B

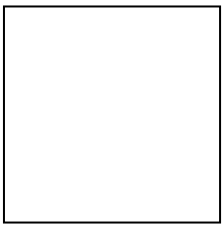
También en el anexo, esta lo que se obtiene al introducir los componentes RGB de la imagen Akiyo.

El siguiente paso es comprobar el funcionamiento del binario de similitud.

En este caso, a continuación se muestran tres pares de imágenes y los resultados que se obtienen en cada caso de similitud:



Obtenemos una similitud de 1.000000 que es bueno puesto que se trata de la misma imagen.



En este segundo caso el valor de la similitud es 0.992218

La similitud correspondiente es ahora 0.996109

6.4. HOMOGENEOUS TEXTURE

Para probar el Homogeneous Texture, primero probamos a introducirle una imagen negra. Según las previsiones, el hecho de que la transformada de una imagen negra valga 0 siempre, tiene que hacer que el resultado del descriptor también tenga este valor para cualquiera de los dominios frecuenciales del filtro.

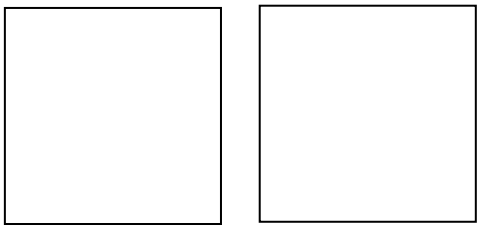
A continuación se puede ver la imagen y el resultado obtenido:

Efectivamente todos los valores son nulos.

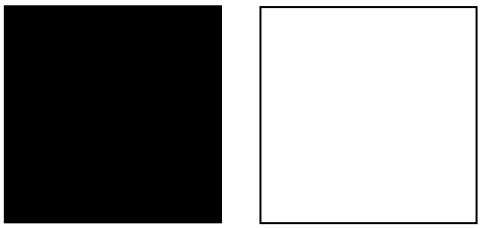
```
Homogeneous Texture Descriptor:
Energy[0] = 0.000000
Energy[1] = 0.000000
Energy[2] = 0.000000
…
Energy[28] = 0.000000
Energy[29] = 0.000000
Energy Desviation[0] = 0.000000
Energy Desviation[1] = 0.000000
Energy Desviation[2] = 0.000000
…
Energy Desviation[28] = 0.000000
Energy Desviation[29] = 0.000000
```

En el anexo, se presentan también los resultados de aplicar el descriptor a una imagen blanca y a la imagen Akiyo.

Para el binario de diferencia, procedemos de manera igual que con el descriptor anterior. A continuación se muestran unas parejas de imágenes y los distintos valores de distancia entre sus respectivos descriptores:



El resultado de la comparación es que efectivamente la distancia es 0.0.



En este caso el valor de la distancia es de 8.092915

Y en este último caso la distancia es mucho mayor y vale 56.727135

7. HERRAMIENTAS UTILIZADAS

Para llevar a cabo dichos descriptores es necesario utilizar los siguientes paquetes de software:

GNU/LINUX

GNU/LINUX: es el sistema operativo formado por la unión del núcleo Linux junto con las librerías y herramientas del proyecto GNU y muchos otros proyectos o grupos de software libre. GNU es el proyecto local donde esta ubicado el núcleo o kernel denominado Linux. [9]

El proyecto se realiza bajo el entorno Linux, debido a las importantes ventajas que proporciona frente a su competidor más cercano, el sistema Windows de Microsoft. El sistema operativo Linux es un software libre formado por código abierto que además de ser gratuito, permite a cualquiera modificar su código fuente para mejorarlo. Hoy en día se ha convertido en una herramienta potente de trabajo utilizada por gobiernos, empresas y universidades como una opción prioritaria si se trata de implementar proyectos de tecnología. Utilizado por la evidente superioridad en el ámbito de la seguridad, ya que actualmente es inmune a la gran mayoría de virus informáticos, en velocidad, integración adaptabilidad, etc…

SOFT_IMAGE:

Soft_Image es una plataforma de procesado de imagen desarrollada por los miembros del departamento de TSC de la UPC, y la contribución de las tesis de diversos estudiantes.

Dicho software esta al alcance de cualquier miembro del grupo de imagen y la autorización para trabajar con el y utilizarlo como una herramienta de procesado de imágenes y videos, es totalmente libre.

Pero si lo que se pretende es modificar sus contenidos con el objetivo de ampliarlos o mejorarlos los procedimientos son los siguientes. Existen dos versiones de Soft_Image. La primera llamada Alpha que es la que contiene un acceso más flexible y es precisamente donde los profesores, doctorantes y proyectistas tienen permiso para modificar las fuentes. Y una segunda denominada Beta que es más restrictiva y solo una persona tiene la faena de revisar los códigos nuevos introducidos en la versión Alpha, para añadirlo a la versión Beta.

Soft_Image esta formado por un conjunto de directorios donde se almacenan los diferentes archivos que son necesarios para la realización del proyecto. Además de integrar documentos de ayuda para facilitar su manipulación. Donde puedes encontrar desde la organización de la estructura principal para saber en que lugar exacto buscar un fichero, hasta un glosario de explicaciones para entender con mas facilidad los códigos introducidos por otras personas con anterioridad.

MAKE

Make es la aplicación que permite a Soft_Image compilar los archivos una vez creados.

Compilar bajo Unix es posible gracias a un archivo llamado MAKEFILE. Este archivo situado en la raíz del directorio de Soft_Image, contiene instrucciones indicando el orden en el cual las fuentes deben ser compiladas. Makefile hace uso de dos archivos más: Slave_Makefiles y machine.mk. Este último contiene las opciones de configuración necesarias para permitir la compilación.

Durante la compilación Makefile es interpretado por el programa make. Por lo que antes de iniciar el proceso de compilación se deben de tener en cuenta tres variables incluidas dentro de dicho archivo:

- UI_BIN_LIST la cual representa la lista de ejecutables de Soft_Image. Esta variable es usada para especificar la lista de carpetas binarias incluyendo las fuentes que el usuario quiere compilar. Para cada fuente de la lista se crea un archivo binario.
- UI_LIB_LIST la cual representa la lista de librerías de Soft_Image. Esta variable es usada para especificar la lista de carpetas librería incluyendo las fuentes que el usuario quiere compilar. Para cada fuente de la lista se crea un archivo librería.
- UI_EXT_LIST la cual representa la lista externa de Soft_Image. La lista externa contiene el conjunto de módulos que no pueden ser compilados con Makefile, suministrada por nuestra plataforma. Tales situaciones pueden suceder si se pretende incluir fuentes que han estado desarrolladas por alguien exterior al grupo de procesamiento de imagen.

En el caso de este proyecto se debe incluir en la de UI_BIN_LIST los nombres de los ejecutables de extracción y similitud creados para cada descriptor.

El orden de los pasos para compilar las fuentes de una aplicación son los siguientes:

1- Gracias a UI_LIB_LIST and UI_BIN_LIST son generados los prototipos desde los archivos fuente. La aplicación make genera los archivos prototipo solo para esas fuentes que han sido cambiadas desde la última compilación.

2- El siguiente paso es la generación de librería: las fuentes incluidas en las carpetas especificadas por UI_LIB_LIST son compiladas y situadas en el directorio Soft_ Image/lib. La localización exacta de las librerías depende de los parámetros pasados para hacer la aplicación make, solo las fuentes que han sido modificadas desde la ultima compilación son compiladas.

3- Finalmente la aplicación make coge cada carpeta de UI_BIN_LIST, compila las fuentes asociadas a esas carpetas, guarda los resultados de los archivos compilados en la carpeta "lib" y genera los correspondientes binarios, que se guardan en la carpeta "bin".

Make comprende cinco tipos de compilación posible:

- Optimizada

Esta opción se utiliza si se desea compilar las fuentes de una forma optimizada y rápida. Basta con escribir "make" en el directorio principal de Soft_Image.

Las librerías generadas son almacenadas en el directorio lib/linux y los binarios en bin/linux.

- Debug

La opción debug se ejecuta con la comanda "make –e DBG=1" y se utiliza si se requiere compilar librerías y binarios con información extra. El procedimiento se basa en poner en los archivos fuentes las líneas de código siguientes:

```
#ifdef _DEBUG_

Comanda de control de error.

#endif
```

Todas las líneas de código que se encuentren dentro de ifdef _DEBUG_ son tenidas en cuenta solo en la opción de compilación DEBUG. Lo que hace que el proceso se ralentice en comparación de la opción optimizada.

Los archivos generados son guardados en los directorios bin/linux_dbg y lib/linux_dbg.

Para poder utilizar las aplicaciones Valgrind y ddd es necesario compilar el código con esta opción.

- Profile

  Ejecutando la comanda "make –e PROF=1" se activa la opción de compilación de tipo profile. Profile es utilizado si se quiere compilar los archivos, incluyendo información de nivel en el código. La información de nivel es un tipo especial de información que el compilador añade a tu código a fin de poder extraer una estadística de ejecución de la aplicación, la cual incluye las veces que una función ha sido llamada, el tiempo de ejecución de cada función, etc.

  Los archivos generados son guardados en los directorios lib/linux_prof y bin/linux_prof.

- Clean

  Si se pretende vaciar los directorios de prototipos, librerías y binarios generados en otras compilaciones, debe ser ejecutada la opción de compilación clean, utilizando el comando "make clean".

De las cinco opciones de compilación anteriores las utilizadas para la realización de este proyecto han sido la Optimizada y Debug. La primera para compilar más rápidamente una vez que todo esta correcto y la segunda para testear fallos y poder utilizar valgrind y ddd.

DDD (Data Display Debugger)

DDD es un compilador que permite ejecutar paso a paso el código del programa realizado. Permite de este modo ver que realiza verdaderamente cada comanda, observar el valor de las variables e incluso visualizar imágenes utilizadas en el programa para poder comprobar si los diversos procesados que se les aplica son correctos.

Para su uso es necesario compilar el programa en opción debug(make –e DBG=1) y ejecutar en la línea de comandos la siguiente sentencia:

"ddd ./B_nombre_del_binario parámetros_de_entrada"

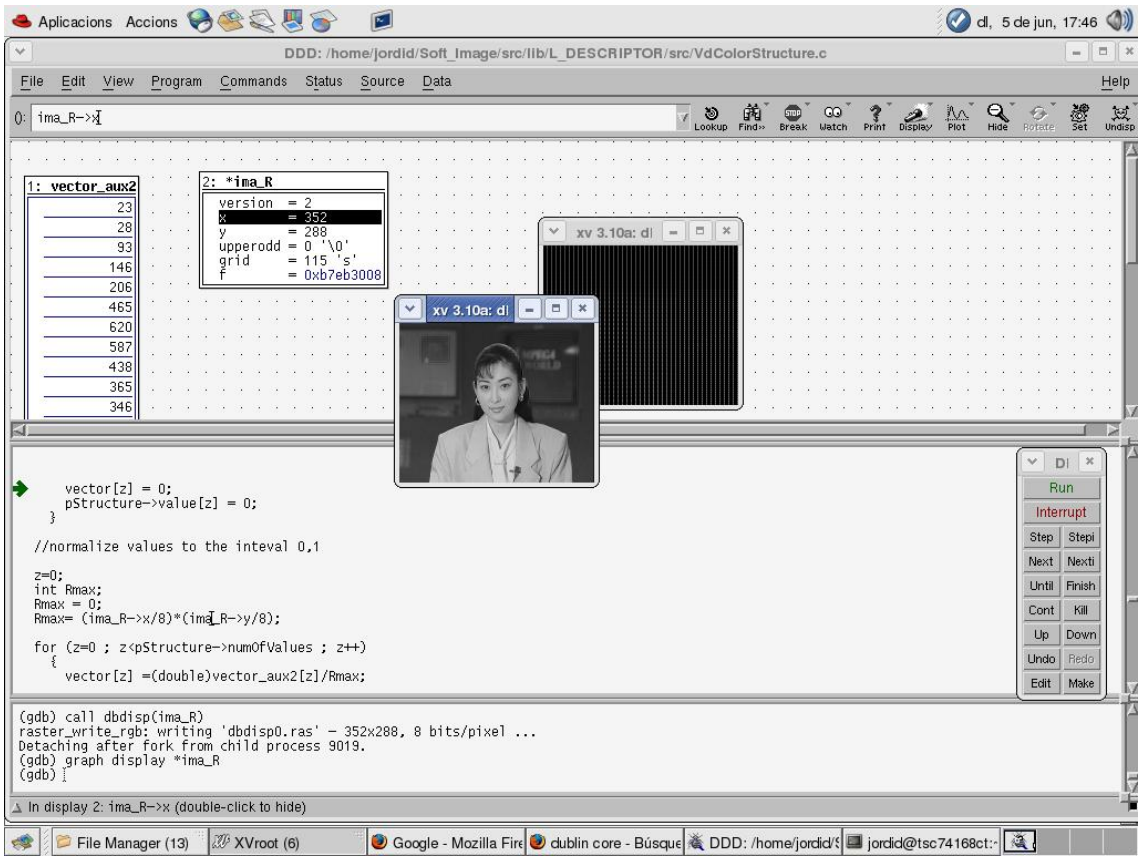


Fig.23 Apariencia de ddd debuger

VALGRIND

Programa de control y detección de errores de memoria.

Valgrind es una herramienta que ayuda a encontrar problemas de manipulación de memoria en cualquier programa. Cuando un programa es ejecutado por Valgrind todas las lecturas y escrituras de memoria son chequeadas, interceptando las llamadas a las funciones malloc (reserva memoria), free (libera memoria), delete y new. Esto precisamente es lo que le permite a Valgrind detectar errores de memoria o acciones de lecturas y escrituras incorrectas.

Tras su instalación y antes de poderlo utilizar debemos primero compilar el programa realizado en la opción debug(make –e DBG=1). Posteriormente se debe ir al directorio donde se han creado los ficheros binarios (bin/linux_dbg) y ejecutar el comando:

"valgrind–tool=memcheck ./B_nombre_del_binario parámetros_de_entrada"

Seguidamente podremos visualizar por pantalla los diversos errores de memoria cometidos.

Esta herramienta ha sido de mucha utilidad para comprobar la eficiencia y la correcta programación de nuestros descriptores en lo que a memoria se refiere. Hemos conseguido que valgrind no detecte ningún error en nuestros descriptores.

XV (XView)

Es una herramienta que te permite visualizar cualquier tipo de imágenes en casi todos los formatos y medidas. El uso de XView en este proyecto ha sido para la conversión de imágenes en formato mapa de bits (.bmp) a Sunraster (.ras). Como Soft_Image mayoritariamente trabaja con dicho formato, que al no ser muy común dificulta la búsqueda de imágenes, xview ha sido de mucha ayuda.

EL LENGUAJE C

El lenguaje C es un lenguaje de programación transportable que se utiliza para desarrollar software [10].

C se creó con el objetivo de tener un lenguaje más comprensible y flexible de creación de programas que el lenguaje ensamblador, pero que siguiese conservando la proximidad con la máquina. En estos momentos ha llegado a convertirse en uno de los lenguajes más utilizados, aunque más orientado a la implementación de sistemas operativos como Unix y Linux lo cuales están realizados en su mayoría en C.

Cuando se compara C con los demás lenguajes de programación, se pueden distinguir unas claras ventajas. Su eficiencia, la cual le permite hacer implementaciones optimas que gracias a sus propiedades de bajo nivel proporcionan velocidades más altas al proceso. Su portabilidad, que le permite ser compilado en casi todos los sistemas conocidos. Y finalmente su flexibilidad, la cual le permite elaborar programas modulares y utilizar bibliotecas y código existente.

Los descriptores se crean bajo el lenguaje C debido a que en la plataforma de Soft_Image los archivos se implementan en dicho lenguaje.

8. CONCLUSIONES

Una vez finalizado el proyecto, hemos extraído las siguientes conclusiones:

Creemos que dicho proyecto nos ha ayudado mucho a familiarizarnos con el sistema operativo Linux, del cual teníamos poco conocimiento. Tras trabajar con él durante estos meses, hemos aprendido las nociones básicas y comprendido algunos de los puntos a favor que presenta frente a Microsoft Windows.

Lo mismo sucede con el lenguaje C, gracias a la tarea de implementar los diferentes descriptores hemos ampliado las nociones de dicho lenguaje en cuestión de utilización de funciones, gestión de reserva y liberación de memoria, uso de punteros, empleo de parámetros por referencia, etc...

Ha sido muy positivo y gratificante el hecho de disponer de programas especializados en detección de errores, como valgrind, o de compilación paso a paso, como ddd, con los que el proceso de comprobación y verificación de los descriptores ha sido una tarea menos ardua y costosa.

En cuanto a la consecución de los objetivos iniciales del proyecto, en general las expectativas se han cumplido, ya que hemos conseguido implementar cuatro descriptores. Sin embargo esta consecución no ha sido completa, debido básicamente a la falta de tiempo. Por ejemplo, solo uno de estos descriptores ha podido usarse en el proyecto final de carrera de Mireia Luna, el Edge Histogram.

Otra de las posibles mejoras que podríamos llevar a cabo sobre nuestros descriptores, seria por ejemplo, amplificar la capacidad del Color Structure para calcular histogramas con menos colores (128 y 64).

O cuantificar la energía del Homogeneous Texture, para poder así implementar la función de similitud en lugar de la de distancia.

Además, podríamos incluso, intentar adaptar los descriptores que solo funcionan con imágenes en escala de grises a imágenes en color como por ejemplo Edge Histogram o Homogeneous Texture.

9. BIBLIOGRAFIA

[1] Liliana Patricia Santacruz Valencia "Estándares aplicados a la educación", Madrid 2000.

[2] Martínez, José M. **"MPEG-7 Overview",** www.chiariglione.org/mpeg/standards/mpeg-7/mpeg-7.htm

[3] Peig Olivé, Enric, "**Interoperabilidad de Metadatos en Sistemas distribuidos",** Tesis Doctoral, Director: Jaime Delgado Merçé, Universidad: Pompeu Fabra, Barcelona, Octubre del 2003.

[4] Rosa, Antonio de la i Senso, Jose A., "**La dualidad texto-imagen en SVG (Scalable Vector Graphics): nuevas posibilidades para la descripción de información gráfica,** En: El profesional de la información, 2003, septiembre-octubre, v. 12, n. 5, pp. 377-398.

[5] Ruiz Hidalgo, Javier, **"Introducción al estándar MPEG-7",** Image processing Group, TSC Departament, Noviembre 2001.

[6] Sakihama Miyashiro, Luis Miguel i Gonzales Jauregui, Antonio, **"Una revisión a los nuevos estandares multimedia",** Sistemas de Televisión, Universidad Nacional de Ingeniería.

[7] Vasquez Paulus**,** Cristian**, "METADATOS: Introducción e historia"**, Julio 2005.

[8] Vivancos Vicente, Pedro José, **"El estàndard MPEG-7"** InforMAS revista de Ingenieria Informática del CIIRM, Ilustre Colegio de ingenieros en informática de la Región de Murcia, Murcia , Julio del 2005.

[9] "Las ventajas de Linux y el Software Libre" www.andalux.com

[10] "**Lenguaje de programación C**" www.wikipedia.org

[11] "Dublín Core metadata initiative" www.dublincore.org

[12] Julien Ricard, David Coeurjolly, y Atilla Baskurt "**Generalizations of Angular Radial Transform for 2D and 3D Shape Retrieval".**

10. ANEXOS

Resultado del Color Structure de una imagen blanca:

ColorStructure Descriptor:

numOfValues = 256

value[0] = 0
value[1] = 0
value[2] = 0
value[3] = 0
value[4] = 0
value[5] = 0
value[6] = 0
value[7] = 0
value[8] = 0
value[9] = 0
value[10] = 0
value[11] = 0
value[12] = 0
value[13] = 0
value[14] = 0
value[15] = 0
value[16] = 0
value[17] = 0
value[18] = 0
value[19] = 0
value[20] = 0
value[21] = 0
value[22] = 0
value[23] = 0
value[24] = 0
value[25] = 0
value[26] = 0
value[27] = 0
value[28] = 0
value[29] = 0
value[30] = 0
value[31] = 255
value[32] = 0
value[33] = 0
value[34] = 0
value[35] = 0
value[36] = 0
value[37] = 0
value[38] = 0
value[39] = 0
value[40] = 0
value[41] = 0
value[42] = 0
value[43] = 0
value[44] = 0
value[45] = 0
value[46] = 0
value[47] = 0
value[48] = 0
value[49] = 0
value[50] = 0
value[51] = 0
value[52] = 0
value[53] = 0
value[54] = 0
value[55] = 0
value[56] = 0
value[57] = 0
value[58] = 0
value[59] = 0
value[60] = 0
value[61] = 0
value[62] = 0
value[63] = 0
value[64] = 0
value[65] = 0
value[66] = 0
value[67] = 0
value[68] = 0
value[69] = 0
value[70] = 0
value[71] = 0
value[72] = 0
value[73] = 0
value[74] = 0
value[75] = 0
value[76] = 0
value[77] = 0
value[78] = 0
value[79] = 0
value[80] = 0
value[81] = 0
value[82] = 0
value[83] = 0
value[84] = 0
value[85] = 0
value[86] = 0
value[87] = 0
value[88] = 0
value[89] = 0
value[90] = 0
value[91] = 0
value[92] = 0
value[93] = 0
value[94] = 0
value[95] = 0
value[96] = 0
value[97] = 0
value[98] = 0
value[99] = 0
value[100] = 0
value[101] = 0
value[102] = 0
value[103] = 0
value[104] = 0
value[105] = 0
value[106] = 0
value[107] = 0
value[108] = 0
value[109] = 0
value[110] = 0
value[111] = 0
value[112] = 0
value[113] = 0
value[114] = 0
value[115] = 0
value[116] = 0
value[117] = 0
value[118] = 0
value[119] = 0
value[120] = 0
value[121] = 0
value[122] = 0
value[123] = 0
value[124] = 0
value[125] = 0
value[126] = 0
value[127] = 0
value[128] = 0
value[129] = 0
value[130] = 0
value[131] = 0
value[132] = 0
value[133] = 0
value[134] = 0
value[135] = 0
value[136] = 0
value[137] = 0
value[138] = 0
value[139] = 0
value[140] = 0
value[141] = 0
value[142] = 0
value[143] = 0
value[144] = 0
value[145] = 0
value[146] = 0
value[147] = 0
value[148] = 0
value[149] = 0
value[150] = 0
value[151] = 0

value[152] = 0	value[178] = 0	value[204] = 0	value[230] = 0
value[153] = 0	value[179] = 0	value[205] = 0	value[231] = 0
value[154] = 0	value[180] = 0	value[206] = 0	value[232] = 0
value[155] = 0	value[181] = 0	value[207] = 0	value[233] = 0
value[156] = 0	value[182] = 0	value[208] = 0	value[234] = 0
value[157] = 0	value[183] = 0	value[209] = 0	value[235] = 0
value[158] = 0	value[184] = 0	value[210] = 0	value[236] = 0
value[159] = 0	value[185] = 0	value[211] = 0	value[237] = 0
value[160] = 0	value[186] = 0	value[212] = 0	value[238] = 0
value[161] = 0	value[187] = 0	value[213] = 0	value[239] = 0
value[162] = 0	value[188] = 0	value[214] = 0	value[240] = 0
value[163] = 0	value[189] = 0	value[215] = 0	value[241] = 0
value[164] = 0	value[190] = 0	value[216] = 0	value[242] = 0
value[165] = 0	value[191] = 0	value[217] = 0	value[243] = 0
value[166] = 0	value[192] = 0	value[218] = 0	value[244] = 0
value[167] = 0	value[193] = 0	value[219] = 0	value[245] = 0
value[168] = 0	value[194] = 0	value[220] = 0	value[246] = 0
value[169] = 0	value[195] = 0	value[221] = 0	value[247] = 0
value[170] = 0	value[196] = 0	value[222] = 0	value[248] = 0
value[171] = 0	value[197] = 0	value[223] = 0	value[249] = 0
value[172] = 0	value[198] = 0	value[224] = 0	value[250] = 0
value[173] = 0	value[199] = 0	value[225] = 0	value[251] = 0
value[174] = 0	value[200] = 0	value[226] = 0	value[252] = 0
value[175] = 0	value[201] = 0	value[227] = 0	value[253] = 0
value[176] = 0	value[202] = 0	value[228] = 0	value[254] = 0
value[177] = 0	value[203] = 0	value[229] = 0	value[255] = 0

Resultado de la imagen negra en el Color Structure:

ColorStructure Descriptor:

numOfValues = 256

value[0] = 255	value[10] = 0	value[20] = 0	value[30] = 0
value[1] = 0	value[11] = 0	value[21] = 0	value[31] = 0
value[2] = 0	value[12] = 0	value[22] = 0	value[32] = 0
value[3] = 0	value[13] = 0	value[23] = 0	value[33] = 0
value[4] = 0	value[14] = 0	value[24] = 0	value[34] = 0
value[5] = 0	value[15] = 0	value[25] = 0	value[35] = 0
value[6] = 0	value[16] = 0	value[26] = 0	value[36] = 0
value[7] = 0	value[17] = 0	value[27] = 0	value[37] = 0
value[8] = 0	value[18] = 0	value[28] = 0	value[38] = 0
value[9] = 0	value[19] = 0	value[29] = 0	value[39] = 0

value[40] = 0
value[41] = 0
value[42] = 0
value[43] = 0
value[44] = 0
value[45] = 0
value[46] = 0
value[47] = 0
value[48] = 0
value[49] = 0
value[50] = 0
value[51] = 0
value[52] = 0
value[53] = 0
value[54] = 0
value[55] = 0
value[56] = 0
value[57] = 0
value[58] = 0
value[59] = 0
value[60] = 0
value[61] = 0
value[62] = 0
value[63] = 0
value[64] = 0
value[65] = 0
value[66] = 0
value[67] = 0
value[68] = 0
value[69] = 0
value[70] = 0
value[71] = 0
value[72] = 0
value[73] = 0
value[74] = 0
value[75] = 0
value[76] = 0
value[77] = 0
value[78] = 0
value[79] = 0
value[80] = 0
value[81] = 0
value[82] = 0
value[83] = 0
value[84] = 0
value[85] = 0
value[86] = 0
value[87] = 0
value[88] = 0
value[89] = 0
value[90] = 0
value[91] = 0
value[92] = 0
value[93] = 0
value[94] = 0
value[95] = 0
value[96] = 0
value[97] = 0
value[98] = 0
value[99] = 0
value[100] = 0
value[101] = 0
value[102] = 0
value[103] = 0
value[104] = 0
value[105] = 0
value[106] = 0
value[107] = 0
value[108] = 0
value[109] = 0
value[110] = 0
value[111] = 0
value[112] = 0
value[113] = 0
value[114] = 0
value[115] = 0
value[116] = 0
value[117] = 0
value[118] = 0
value[119] = 0
value[120] = 0
value[121] = 0
value[122] = 0
value[123] = 0
value[124] = 0
value[125] = 0
value[126] = 0
value[127] = 0
value[128] = 0
value[129] = 0
value[130] = 0
value[131] = 0
value[132] = 0
value[133] = 0
value[134] = 0
value[135] = 0
value[136] = 0
value[137] = 0
value[138] = 0
value[139] = 0
value[140] = 0
value[141] = 0
value[142] = 0
value[143] = 0
value[144] = 0
value[145] = 0
value[146] = 0
value[147] = 0
value[148] = 0
value[149] = 0
value[150] = 0
value[151] = 0
value[152] = 0
value[153] = 0
value[154] = 0
value[155] = 0
value[156] = 0
value[157] = 0
value[158] = 0
value[159] = 0
value[160] = 0
value[161] = 0
value[162] = 0
value[163] = 0
value[164] = 0
value[165] = 0
value[166] = 0
value[167] = 0
value[168] = 0
value[169] = 0
value[170] = 0
value[171] = 0
value[172] = 0
value[173] = 0
value[174] = 0
value[175] = 0
value[176] = 0
value[177] = 0
value[178] = 0
value[179] = 0
value[180] = 0
value[181] = 0
value[182] = 0
value[183] = 0
value[184] = 0
value[185] = 0
value[186] = 0
value[187] = 0
value[188] = 0
value[189] = 0
value[190] = 0
value[191] = 0
value[192] = 0
value[193] = 0
value[194] = 0
value[195] = 0
value[196] = 0
value[197] = 0
value[198] = 0
value[199] = 0
value[200] = 0
value[201] = 0
value[202] = 0
value[203] = 0
value[204] = 0
value[205] = 0
value[206] = 0
value[207] = 0
value[208] = 0
value[209] = 0
value[210] = 0
value[211] = 0
value[212] = 0
value[213] = 0
value[214] = 0
value[215] = 0
value[216] = 0
value[217] = 0
value[218] = 0
value[219] = 0
value[220] = 0
value[221] = 0
value[222] = 0
value[223] = 0
value[224] = 0
value[225] = 0
value[226] = 0
value[227] = 0

value[228] = 0	value[235] = 0	value[242] = 0	value[249] = 0
value[229] = 0	value[236] = 0	value[243] = 0	value[250] = 0
value[230] = 0	value[237] = 0	value[244] = 0	value[251] = 0
value[231] = 0	value[238] = 0	value[245] = 0	value[252] = 0
value[232] = 0	value[239] = 0	value[246] = 0	value[253] = 0
value[233] = 0	value[240] = 0	value[247] = 0	value[254] = 0
value[234] = 0	value[241] = 0	value[248] = 0	value[255] = 0

Resultado del Color Structure a la imagen "blanco_negro_blanco"

ColorStructure Descriptor:

numOfValues = 256

value[0] = 0	value[30] = 0	value[60] = 0	value[90] = 0
value[1] = 0	value[31] = 0	value[61] = 0	value[91] = 0
value[2] = 0	value[32] = 0	value[62] = 0	value[92] = 0
value[3] = 0	value[33] = 0	value[63] = 0	value[93] = 0
value[4] = 0	value[34] = 0	value[64] = 0	value[94] = 0
value[5] = 0	value[35] = 0	value[65] = 0	value[95] = 0
value[6] = 0	value[36] = 0	value[66] = 0	value[96] = 0
value[7] = 0	value[37] = 0	value[67] = 0	value[97] = 0
value[8] = 0	value[38] = 0	value[68] = 0	value[98] = 0
value[9] = 0	value[39] = 0	value[69] = 0	value[99] = 0
value[10] = 0	value[40] = 0	value[70] = 0	value[100] = 0
value[11] = 0	value[41] = 0	value[71] = 0	value[101] = 0
value[12] = 0	value[42] = 0	value[72] = 0	value[102] = 0
value[13] = 0	value[43] = 0	value[73] = 0	value[103] = 0
value[14] = 0	value[44] = 0	value[74] = 0	value[104] = 0
value[15] = 0	value[45] = 0	value[75] = 0	value[105] = 0
value[16] = 0	value[46] = 0	value[76] = 0	value[106] = 0
value[17] = 0	value[47] = 0	value[77] = 0	value[107] = 0
value[18] = 0	value[48] = 0	value[78] = 0	value[108] = 0
value[19] = 0	value[49] = 0	value[79] = 0	value[109] = 0
value[20] = 0	value[50] = 0	value[80] = 0	value[110] = 0
value[21] = 0	value[51] = 0	value[81] = 0	value[111] = 0
value[22] = 0	value[52] = 0	value[82] = 0	value[112] = 0
value[23] = 0	value[53] = 0	value[83] = 0	value[113] = 0
value[24] = 0	value[54] = 0	value[84] = 0	value[114] = 0
value[25] = 0	value[55] = 0	value[85] = 0	value[115] = 0
value[26] = 0	value[56] = 0	value[86] = 0	value[116] = 0
value[27] = 0	value[57] = 0	value[87] = 0	value[117] = 0
value[28] = 0	value[58] = 0	value[88] = 0	value[118] = 0
value[29] = 0	value[59] = 0	value[89] = 0	value[119] = 0

value[120] = 0	value[154] = 0	value[188] = 0	value[222] = 0
value[121] = 0	value[155] = 0	value[189] = 0	value[223] = 0
value[122] = 0	value[156] = 0	value[190] = 0	value[224] = 0
value[123] = 0	value[157] = 0	value[191] = 0	value[225] = 0
value[124] = 0	value[158] = 0	value[192] = 0	value[226] = 0
value[125] = 0	value[159] = 0	value[193] = 0	value[227] = 0
value[126] = 0	value[160] = 0	value[194] = 0	value[228] = 0
value[127] = 0	value[161] = 0	value[195] = 0	value[229] = 0
value[128] = 0	value[162] = 0	value[196] = 0	value[230] = 0
value[129] = 0	value[163] = 0	value[197] = 0	value[231] = 0
value[130] = 0	value[164] = 0	value[198] = 0	value[232] = 0
value[131] = 0	value[165] = 0	value[199] = 0	value[233] = 0
value[132] = 0	value[166] = 0	value[200] = 0	value[234] = 0
value[133] = 0	value[167] = 0	value[201] = 0	value[235] = 0
value[134] = 0	value[168] = 0	value[202] = 0	value[236] = 0
value[135] = 0	value[169] = 0	value[203] = 0	value[237] = 0
value[136] = 0	value[170] = 0	value[204] = 0	value[238] = 0
value[137] = 0	value[171] = 0	value[205] = 0	value[239] = 0
value[138] = 0	value[172] = 0	value[206] = 0	value[240] = 0
value[139] = 0	value[173] = 0	value[207] = 255	value[241] = 0
value[140] = 0	value[174] = 0	value[208] = 0	value[242] = 0
value[141] = 0	value[175] = 0	value[209] = 0	value[243] = 0
value[142] = 0	value[176] = 0	value[210] = 0	value[244] = 0
value[143] = 0	value[177] = 0	value[211] = 0	value[245] = 0
value[144] = 0	value[178] = 0	value[212] = 0	value[246] = 0
value[145] = 0	value[179] = 0	value[213] = 0	value[247] = 0
value[146] = 0	value[180] = 0	value[214] = 0	value[248] = 0
value[147] = 0	value[181] = 0	value[215] = 0	value[249] = 0
value[148] = 0	value[182] = 0	value[216] = 0	value[250] = 0
value[149] = 0	value[183] = 0	value[217] = 0	value[251] = 0
value[150] = 0	value[184] = 0	value[218] = 0	value[252] = 0
value[151] = 0	value[185] = 0	value[219] = 0	value[253] = 0
value[152] = 0	value[186] = 0	value[220] = 0	value[254] = 0
value[153] = 0	value[187] = 0	value[221] = 0	value[255] = 0

Resultado del descriptor Color Structure para la imagen de Akiyo:

ColorStructure Descriptor:

numOfValues = 256

value[0] = 0	value[6] = 92	value[12] = 38	value[18] = 3
value[1] = 3	value[7] = 91	value[13] = 9	value[19] = 0
value[2] = 4	value[8] = 65	value[14] = 8	value[20] = 0
value[3] = 39	value[9] = 54	value[15] = 4	value[21] = 0
value[4] = 30	value[10] = 48	value[16] = 4	value[22] = 0
value[5] = 85	value[11] = 47	value[17] = 3	value[23] = 0

value[24] = 0
value[25] = 0
value[26] = 0
value[27] = 0
value[28] = 0
value[29] = 0
value[30] = 0
value[31] = 0
value[32] = 21
value[33] = 96
value[34] = 128
value[35] = 72
value[36] = 54
value[37] = 64
value[38] = 29
value[39] = 6
value[40] = 3
value[41] = 0
value[42] = 0
value[43] = 0
value[44] = 0
value[45] = 0
value[46] = 0
value[47] = 0
value[48] = 0
value[49] = 0
value[50] = 0
value[51] = 0
value[52] = 0
value[53] = 0
value[54] = 0
value[55] = 0
value[56] = 0
value[57] = 0
value[58] = 0
value[59] = 0
value[60] = 0
value[61] = 0
value[62] = 0
value[63] = 0
value[64] = 0
value[65] = 49
value[66] = 88
value[67] = 97
value[68] = 36
value[69] = 3
value[70] = 0
value[71] = 0
value[72] = 0
value[73] = 0
value[74] = 28
value[75] = 43
value[76] = 4
value[77] = 0
value[78] = 0
value[79] = 0
value[80] = 3
value[81] = 3
value[82] = 0
value[83] = 0
value[84] = 0
value[85] = 0
value[86] = 0
value[87] = 0
value[88] = 0
value[89] = 0
value[90] = 0
value[91] = 0
value[92] = 0
value[93] = 0
value[94] = 0
value[95] = 0
value[96] = 0
value[97] = 0
value[98] = 0
value[99] = 0
value[100] = 0
value[101] = 0
value[102] = 0
value[103] = 0
value[104] = 0
value[105] = 0
value[106] = 0
value[107] = 0
value[108] = 0
value[109] = 0
value[110] = 0
value[111] = 0
value[112] = 0
value[113] = 0
value[114] = 0
value[115] = 0
value[116] = 0
value[117] = 0
value[118] = 0
value[119] = 0
value[120] = 0
value[121] = 0
value[122] = 0
value[123] = 0
value[124] = 0
value[125] = 0
value[126] = 0
value[127] = 0
value[128] = 0
value[129] = 4
value[130] = 51
value[131] = 46
value[132] = 0
value[133] = 0
value[134] = 0
value[135] = 0
value[136] = 3
value[137] = 6
value[138] = 16
value[139] = 42
value[140] = 6
value[141] = 0
value[142] = 0
value[143] = 0
value[144] = 0
value[145] = 0
value[146] = 0
value[147] = 0
value[148] = 0
value[149] = 0
value[150] = 0
value[151] = 0
value[152] = 0
value[153] = 0
value[154] = 0
value[155] = 0
value[156] = 0
value[157] = 0
value[158] = 0
value[159] = 0
value[160] = 0
value[161] = 0
value[162] = 0
value[163] = 0
value[164] = 0
value[165] = 0
value[166] = 0
value[167] = 0
value[168] = 0
value[169] = 0
value[170] = 0
value[171] = 0
value[172] = 0
value[173] = 0
value[174] = 0
value[175] = 0
value[176] = 0
value[177] = 0
value[178] = 0
value[179] = 0
value[180] = 0
value[181] = 0
value[182] = 0
value[183] = 0
value[184] = 0
value[185] = 0
value[186] = 0
value[187] = 0
value[188] = 0
value[189] = 0
value[190] = 0
value[191] = 0
value[192] = 0
value[193] = 0
value[194] = 0
value[195] = 0
value[196] = 0
value[197] = 0
value[198] = 0
value[199] = 0
value[200] = 0
value[201] = 4
value[202] = 26
value[203] = 67
value[204] = 56
value[205] = 0
value[206] = 0
value[207] = 0
value[208] = 0
value[209] = 0
value[210] = 0
value[211] = 0

value[212] = 0
value[213] = 0
value[214] = 0
value[215] = 0
value[216] = 0
value[217] = 0
value[218] = 0
value[219] = 0
value[220] = 0
value[221] = 0
value[222] = 0
value[223] = 0
value[224] = 0
value[225] = 0
value[226] = 0
value[227] = 0
value[228] = 0
value[229] = 0
value[230] = 0
value[231] = 0
value[232] = 0
value[233] = 0
value[234] = 0
value[235] = 0
value[236] = 0
value[237] = 0
value[238] = 0
value[239] = 0
value[240] = 0
value[241] = 0
value[242] = 0
value[243] = 0
value[244] = 0
value[245] = 0
value[246] = 0
value[247] = 0
value[248] = 0
value[249] = 0
value[250] = 0
value[251] = 0
value[252] = 0
value[253] = 0
value[254] = 0
value[255] = 0

Resultado del descriptor HomoTexture de una imagen blanca:

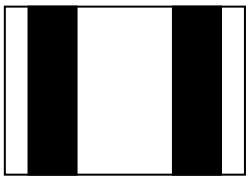
Homogeneous Texture Descriptor:
Energy[0] = 4.795362
Energy[1] = 0.000006
Energy[2] = 0.000000
Energy[3] = 0.000000
Energy[4] = 0.000000
Energy[5] = 0.000000
Energy[6] = 0.000000
Energy[7] = 0.000000
Energy[8] = 0.000000
Energy[9] = 0.000000
Energy[10] = 0.000000
Energy[11] = 0.000000
Energy[12] = 0.000000
Energy[13] = 0.000000
Energy[14] = 0.000000
Energy[15] = 0.000000
Energy[16] = 0.000000
Energy[17] = 0.000000
Energy[18] = 0.000000
Energy[19] = 0.000000
Energy[20] = 0.000000
Energy[21] = 0.000000
Energy[22] = 0.000000
Energy[23] = 0.000000
Energy[24] = 0.000000
Energy[25] = 0.000000
Energy[26] = 0.000000
Energy[27] = 0.000000
Energy[28] = 0.000000
Energy[29] = 0.000000
Energy Desviation[0] = 6.594696
Energy Desviation[1] = 0.000397
Energy Desviation[2] = 0.000000
Energy Desviation[3] = 0.000000
Energy Desviation[4] = 0.000000
Energy Desviation[5] = 0.000000
Energy Desviation[6] = 0.000000
Energy Desviation[7] = 0.000000
Energy Desviation[8] = 0.000000
Energy Desviation[9] = 0.000000
Energy Desviation[10] = 0.000000
Energy Desviation[11] = 0.000000
Energy Desviation[12] = 0.000000
Energy Desviation[13] = 0.000000
Energy Desviation[14] = 0.000000
Energy Desviation[15] = 0.000000
Energy Desviation[16] = 0.000000
Energy Desviation[17] = 0.000000
Energy Desviation[18] = 0.000000
Energy Desviation[19] = 0.000000
Energy Desviation[20] = 0.000000
Energy Desviation[21] = 0.000000
Energy Desviation[22] = 0.000000
Energy Desviation[23] = 0.000000
Energy Desviation[24] = 0.000000
Energy Desviation[25] = 0.000000
Energy Desviation[26] = 0.000000
Energy Desviation[27] = 0.000000
Energy Desviation[28] = 0.000000
Energy Desviation[29] = 0.000000

Resultado del descriptor Homogeneous Texture de una imagen Akiyo:

Homogeneous Texture Descriptor:
Energy[0] = 9.672271
Energy[1] = 10.297669
Energy[2] = 10.050285
Energy[3] = 5.158335
Energy[4] = 0.000013
Energy[5] = 0.000000
Energy[6] = 2.263987
Energy[7] = 0.959197
Energy[8] = 0.000185
Energy[9] = 0.000000
Energy[10] = 0.000000
Energy[11] = 0.000000
Energy[12] = 0.000000
Energy[13] = 0.000000
Energy[14] = 0.000000
Energy[15] = 0.000000
Energy[16] = 0.000000
Energy[17] = 0.000000
Energy[18] = 0.000000
Energy[19] = 0.000000
Energy[20] = 0.000000
Energy[21] = 0.000000
Energy[22] = 0.000000
Energy[23] = 0.000000
Energy[24] = 0.000000
Energy[25] = 0.000000
Energy[26] = 0.000000
Energy[27] = 0.000000
Energy[28] = 0.000000
Energy[29] = 0.000000

Energy Desviation[0] = 12.078811
Energy Desviation[1] = 12.704210
Energy Desviation[2] = 12.456824
Energy Desviation[3] = 7.564872
Energy Desviation[4] = 0.003216
Energy Desviation[5] = 0.000000
Energy Desviation[6] = 4.668166
Energy Desviation[7] = 3.315410
Energy Desviation[8] = 0.044715
Energy Desviation[9] = 0.000000
Energy Desviation[10] = 0.000000
Energy Desviation[11] = 0.000000
Energy Desviation[12] = 0.000000
Energy Desviation[13] = 0.000000
Energy Desviation[14] = 0.000000
Energy Desviation[15] = 0.000000
Energy Desviation[16] = 0.000000
Energy Desviation[17] = 0.000000
Energy Desviation[18] = 0.000000
Energy Desviation[19] = 0.000000
Energy Desviation[20] = 0.000000
Energy Desviation[21] = 0.000000
Energy Desviation[22] = 0.000000
Energy Desviation[23] = 0.000000
Energy Desviation[24] = 0.000000
Energy Desviation[25] = 0.000000
Energy Desviation[26] = 0.000000
Energy Desviation[27] = 0.000000
Energy Desviation[28] = 0.000000
Energy Desviation[29] = 0.000000

Resultado del descriptor Homo Texture de la imagen:

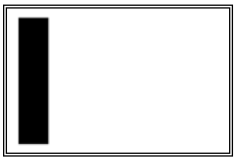


Homogeneous Texture Descriptor:

Energy[0] = 9.547877
Energy[1] = 9.877400
Energy[2] = 9.024649
Energy[3] = 3.840892
Energy[4] = 0.000000
Energy[5] = 0.000000
Energy[6] = 1.977641
Energy[7] = 1.065083
Energy[8] = 0.000149
Energy[9] = 0.000000
Energy[10] = 0.000000
Energy[11] = 0.000000
Energy[12] = 0.000000
Energy[13] = 0.000000
Energy[14] = 0.000000
Energy[15] = 0.000000
Energy[16] = 0.000000
Energy[17] = 0.000000
Energy[18] = 0.000000
Energy[19] = 0.000000
Energy[20] = 0.000000
Energy[21] = 0.000000
Energy[22] = 0.000000
Energy[23] = 0.000000
Energy[24] = 0.000000
Energy[25] = 0.000000
Energy[26] = 0.000000
Energy[27] = 0.000000
Energy[28] = 0.000000
Energy[29] = 0.000000

Energy Desviation[0] = 11.954418
Energy Desviation[1] = 12.283940
Energy Desviation[2] = 11.431189
Energy Desviation[3] = 6.247369
Energy Desviation[4] = 0.000103
Energy Desviation[5] = 0.000000
Energy Desviation[6] = 4.379603
Energy Desviation[7] = 3.432690
Energy Desviation[8] = 0.036465
Energy Desviation[9] = 0.000000
Energy Desviation[10] = 0.000000
Energy Desviation[11] = 0.000000
Energy Desviation[12] = 0.000000
Energy Desviation[13] = 0.000000
Energy Desviation[14] = 0.000000
Energy Desviation[15] = 0.000000
Energy Desviation[16] = 0.000000
Energy Desviation[17] = 0.000000
Energy Desviation[18] = 0.000000
Energy Desviation[19] = 0.000000
Energy Desviation[20] = 0.000000
Energy Desviation[21] = 0.000000
Energy Desviation[22] = 0.000000
Energy Desviation[23] = 0.000000
Energy Desviation[24] = 0.000000
Energy Desviation[25] = 0.000000
Energy Desviation[26] = 0.000000
Energy Desviation[27] = 0.000000
Energy Desviation[28] = 0.000000
Energy Desviation[29] = 0.000000

Resultado del descriptor Edge Histogram de la imagen:

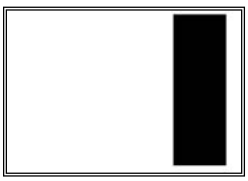


Edge Histogram Descriptor:

BinCount[0] = 1
BinCount[1] = 0
BinCount[2] = 2
BinCount[3] = 2
BinCount[4] = 1
BinCount[5] = 0
BinCount[6] = 0
BinCount[7] = 0

BinCount[8] = 0
BinCount[9] = 0
BinCount[10] = 0
BinCount[11] = 0
BinCount[12] = 0
BinCount[13] = 0
BinCount[14] = 0
BinCount[15] = 0
BinCount[16] = 0
BinCount[17] = 0
BinCount[18] = 0
BinCount[19] = 0
BinCount[20] = 1
BinCount[21] = 0
BinCount[22] = 1
BinCount[23] = 1
BinCount[24] = 1
BinCount[25] = 0
BinCount[26] = 0
BinCount[27] = 0
BinCount[28] = 0
BinCount[29] = 0
BinCount[30] = 0
BinCount[31] = 0
BinCount[32] = 0
BinCount[33] = 0
BinCount[34] = 0
BinCount[35] = 0
BinCount[36] = 0
BinCount[37] = 0
BinCount[38] = 0
BinCount[39] = 0
BinCount[40] = 1
BinCount[41] = 0
BinCount[42] = 1
BinCount[43] = 1
BinCount[44] = 1
BinCount[45] = 0
BinCount[46] = 0
BinCount[47] = 0
BinCount[48] = 0
BinCount[49] = 0
BinCount[50] = 0
BinCount[51] = 0
BinCount[52] = 0
BinCount[53] = 0
BinCount[54] = 0
BinCount[55] = 0
BinCount[56] = 0
BinCount[57] = 0
BinCount[58] = 0
BinCount[59] = 0
BinCount[60] = 1
BinCount[61] = 0
BinCount[62] = 2
BinCount[63] = 2
BinCount[64] = 1
BinCount[65] = 0
BinCount[66] = 0
BinCount[67] = 0
BinCount[68] = 0
BinCount[69] = 0
BinCount[70] = 0
BinCount[71] = 0
BinCount[72] = 0
BinCount[73] = 0
BinCount[74] = 0
BinCount[75] = 0
BinCount[76] = 0
BinCount[77] = 0
BinCount[78] = 0
BinCount[79] = 0

Resultado del descriptor Edge Histogram de la imagen:

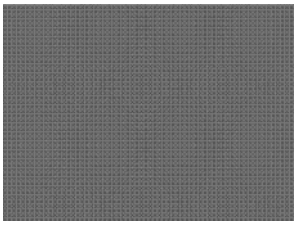


Edge Histogram Descriptor:

BinCount[0] = 0
BinCount[1] = 0
BinCount[2] = 0
BinCount[3] = 0
BinCount[4] = 0
BinCount[5] = 0
BinCount[6] = 0
BinCount[7] = 0
BinCount[8] = 0
BinCount[9] = 0
BinCount[10] = 1
BinCount[11] = 0
BinCount[12] = 1
BinCount[13] = 1
BinCount[14] = 1
BinCount[15] = 1
BinCount[16] = 1
BinCount[17] = 1
BinCount[18] = 1
BinCount[19] = 1
BinCount[20] = 0
BinCount[21] = 0
BinCount[22] = 0
BinCount[23] = 0
BinCount[24] = 0
BinCount[25] = 0
BinCount[26] = 0
BinCount[27] = 0
BinCount[28] = 0
BinCount[29] = 0
BinCount[30] = 1
BinCount[31] = 0
BinCount[32] = 1
BinCount[33] = 1
BinCount[34] = 1
BinCount[35] = 1
BinCount[36] = 0
BinCount[37] = 1
BinCount[38] = 1
BinCount[39] = 1
BinCount[40] = 0
BinCount[41] = 0
BinCount[42] = 0
BinCount[43] = 0
BinCount[44] = 0
BinCount[45] = 0
BinCount[46] = 0
BinCount[47] = 0
BinCount[48] = 0
BinCount[49] = 0
BinCount[50] = 1
BinCount[51] = 0
BinCount[52] = 1
BinCount[53] = 1
BinCount[54] = 1
BinCount[55] = 1
BinCount[56] = 0
BinCount[57] = 1
BinCount[58] = 1
BinCount[59] = 1
BinCount[60] = 0
BinCount[61] = 0
BinCount[62] = 0
BinCount[63] = 0
BinCount[64] = 0
BinCount[65] = 0
BinCount[66] = 0
BinCount[67] = 0
BinCount[68] = 0

BinCount[69] = 0
BinCount[70] = 0
BinCount[71] = 0
BinCount[72] = 1
BinCount[73] = 1
BinCount[74] = 1
BinCount[75] = 0
BinCount[76] = 1
BinCount[77] = 1
BinCount[78] = 1
BinCount[79] = 1

Resultado del descriptor Edge Histogram de la imagen:



Edge Histogram Descriptor:
BinCount[0] = 6
BinCount[1] = 6
BinCount[2] = 8
BinCount[3] = 8
BinCount[4] = 8
BinCount[5] = 6
BinCount[6] = 6
BinCount[7] = 8
BinCount[8] = 8
BinCount[9] = 8
BinCount[10] = 6
BinCount[11] = 6
BinCount[12] = 8
BinCount[13] = 8
BinCount[14] = 8
BinCount[15] = 6
BinCount[16] = 6
BinCount[17] = 8
BinCount[18] = 8
BinCount[19] = 8
BinCount[20] = 6
BinCount[21] = 6
BinCount[22] = 8
BinCount[23] = 8
BinCount[24] = 8
BinCount[25] = 6
BinCount[26] = 6
BinCount[27] = 8
BinCount[28] = 8
BinCount[29] = 8
BinCount[30] = 6
BinCount[31] = 6
BinCount[32] = 8
BinCount[33] = 8
BinCount[34] = 8
BinCount[35] = 6
BinCount[36] = 6
BinCount[37] = 8
BinCount[38] = 8
BinCount[39] = 8
BinCount[40] = 6
BinCount[41] = 6
BinCount[42] = 8
BinCount[43] = 8
BinCount[44] = 8
BinCount[45] = 6
BinCount[46] = 6
BinCount[47] = 8
BinCount[48] = 8
BinCount[49] = 8
BinCount[50] = 6
BinCount[51] = 6
BinCount[52] = 8
BinCount[53] = 8
BinCount[54] = 8

BinCount[55] = 6
BinCount[56] = 6
BinCount[57] = 8
BinCount[58] = 8
BinCount[59] = 8
BinCount[60] = 6
BinCount[61] = 6
BinCount[62] = 8
BinCount[63] = 8
BinCount[64] = 8
BinCount[65] = 6
BinCount[66] = 6
BinCount[67] = 8
BinCount[68] = 8
BinCount[69] = 8
BinCount[70] = 6
BinCount[71] = 6
BinCount[72] = 8
BinCount[73] = 8
BinCount[74] = 8
BinCount[75] = 6
BinCount[76] = 6
BinCount[77] = 8
BinCount[78] = 8
BinCount[79] = 8