ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

Recuperación de Imágenes Usando Atributos Difusos

Realizado por:

Jorge Eloy-García Vargas-Machuca

Dirigido por:

Mª Carmen Aranda Garrido

José Galindo Gómez

Málaga, Abril de 2003

INDICE GENERAL

Referencias

1. INTRODUCCIÓN

En esta primera sección, vamos a introducir muy brevemente un resumen de los aspectos principales de este proyecto. En primer lugar, mostraremos las motivaciones que nos han llevado a realizar este trabajo; en segundo lugar, explicaremos cuáles son los objetivos que se han establecido sobre él; por último, comentaremos las secciones en las que está dividida esta memoria, junto a una descripción somera de su contenido.

1.1. Motivación

En los últimos años ha habido un rápido incremento en el tamaño de las colecciones de imágenes digitales [RuHu99]. Cada día, equipos militares y civiles generan gigabytes de imágenes con contenidos muy diversos y variados, que pueden tener, en determinadas circunstancias, una importancia crítica. Sin embargo, no podemos hacer uso de esa información tan extensa sin organizarla, de manera que podamos acceder, buscar y recuperar imágenes de una forma eficiente y eficaz.

La **Recuperación de Imágenes** (*Image Retrieval*) ha sido un área de investigación muy activa desde los años 70, gracias al empuje de las dos comunidades de investigación mayoritarias: *Gestión de Bases de Datos* (*Database Management*) y *Visión por Computador* (*Computer Vision*). Estas dos comunidades estudian la recuperación de imágenes desde distintos ángulos: *basada en texto* (*text-based*) y *basada en el contenido* (*content-based*).

La recuperación de imágenes basada en texto comienza a finales de los años 70. Entonces, las imágenes se anotaban mediante texto, utilizándose *sistemas gestores de bases de datos* (*SGBD*) *basados en texto* para realizar el proceso de recuperación de imágenes. Además, avances como el modelado de datos o la evaluación de consultas permitieron la continuación de esta línea de investigación. Sin embargo, este esquema posee dos dificultades, especialmente cuando las colecciones de imágenes son grandes (decenas o cientos de miles): una es la gran cantidad de esfuerzo requerido en la anotación manual de las imágenes. La otra dificultad (más importante) resulta de la total relatividad que existe en la

percepción de las imágenes por los seres humanos. Una misma imagen es percibida de forma distinta por dos humanos diferentes.

A principio de los años 90 y, tratando de evitar estos dos problemas expuestos, se propone la recuperación de imágenes basada en el contenido (*content-based*). Con esta propuesta, las imágenes podían ser indexadas por su propio contenido visual, como el color, la textura, las relaciones espaciales o su forma general (*shape*). Desde entonces, se han creado numerosas técnicas y *sistemas de recuperación de imágenes* (de investigación y comerciales) a partir de esta línea de investigación.

La *extracción de características o rasgos* (*feature extraction*) es la primera fase del paradigma de recuperación de imágenes basadas en el contenido. Desde este punto de vista, las características pueden clasificarse en *características generales* y en *características específicas del dominio*. Las primeras incluyen el color, la textura, las relaciones espaciales y la forma, mientras que la última es dependiente de la aplicación en cuestión (como rostros humanos o huellas digitales) y, por tanto, requiere de un conocimiento particular del dominio analizado. En general, las *características específicas del dominio* son abordadas de manera más intensiva en el área de *Reconocimiento de Patrones* y, como se ha mencionado, requieren de un conocimiento exhaustivo del dominio en cuestión.

Por tanto, el objetivo es asociar cada imagen con lo que se denomina un *vector de características*, que estará formado por toda la información visual que se extraiga de cada una de estas imágenes.

Centrándonos en la representación de las imágenes a través de su forma (*shape*), hay que mencionar que en general se pretende que la representación sea invariante a *traslación*, *escalado* y *rotación*, es decir, que si la forma sufre alguna de las transformaciones anteriores, no cambie su representación. Este tipo de representaciones puede dividirse en dos grandes grupos: *representaciones basadas en el contorno y representaciones basadas en regiones*. El primero de ellos utiliza el contorno exterior de la forma para caracterizarla, mientras que el último utiliza la región entera. Existen numerosas representaciones basadas en el contorno, puesto que éste puede ser caracterizado de muchas formas. Por ejemplo, mediante los *descriptores de Fourier* del contorno, o mediante la utilización de *momentos invariantes*.

Tras esto, y para permitir la comparación de imágenes (segunda fase), generalmente se diseña una *medida de similitud* (o *distancia*) entre las representaciones. Si la distancia entre dos representaciones es pequeña indica que son parecidas. Por tanto, interesan aquellas imágenes de la base de datos que tengan la menor distancia posible con respecto a la imagen de entrada.

Estos conceptos teóricos quedan reflejados en la práctica mediante los denominados *Sistemas de Recuperación de Imágenes*, utilizados desde el punto de vista investigador y comercial. Por ejemplo, *QBIC* [NgSe96] es un sistema de recuperación de imágenes basado en el contenido. Tiene en cuenta el color, la textura (grosor, contraste, direccionalidad, ...) y la forma (área, circularidad, excentricidad, etc.) de cada imagen para caracterizarla y, posteriormente, compararla con otras imágenes. *RetrievalWare* [Dowe93] es también un sistema de recuperación de imágenes basado en el contenido. Su motor de búsqueda utiliza los rasgos color, forma, textura, brillo, etc. Otro sistema muy parecido a QBIC es *Virage* [BaFu97], que permite realizar consultas visuales basadas en el color, textura y estructura (información del contorno). Sin embargo, *Virage* va un paso más allá que QBIC, ya que permite realizar combinaciones de consultas a partir de cuatro consultas atómicas, pudiendo modificar el peso de cada una de ellas, permitiendo de esta manera que el usuario enfatice la/s característica/s que desee.

Realmente, existe un gran número de sistemas de este tipo que trata de combinar muchos descriptores o rasgos (*features*) de la imagen para caracterizarla de la forma más detallada y exacta posible, extrayendo de ella la máxima información posible, para poder compararla posteriormente con otras imágenes.

1.2. Objetivos

Se trata de desarrollar un sistema de recuperación de imágenes basado en el contenido. Como rasgo identificativo de las imágenes, únicamente se utilizará la forma (*shape*), más concretamente, una representación basada en el contorno. El proceso de caracterización de una imagen, de forma breve, será el siguiente. A partir de la imagen de entrada (con una única figura en su seno), se obtiene el contorno de la misma. Una vez calculado éste, se

deriva de él la función de curvatura. Se trata de una función que nos muestra como de accidentado o suave es el contorno exterior de la figura en cuestión. A través de esta función, podremos obtener los puntos más representativos de la imagen (*puntos característicos*) con toda su información, que describirán a partir de ese momento, a la imagen en la base de datos. Los puntos que se seleccionan son aquellos con alto valor de curvatura porque coinciden con esquinas (vértices) del objeto.

La 2ª fase de comparación entre imágenes no se va a realizar utilizando una *función de similitud*, sino realizando una comparación entre los puntos característicos de las imágenes a comparar. Es decir, consideraremos que dos imágenes son iguales si los objetos que contienen coinciden en la localización de sus puntos característicos. Para introducir cierta flexibilidad en esta comparación, se almacena la información en una *Base de Datos Difusa*. De esta manera almacenaremos, en forma de atributos difusos, determinada información sobre las imágenes de la base de datos. Los aspectos a considerar son los siguientes, es decir, la información que vamos a almacenar en la base de datos difusa consta de [ArGa98]:

- **Grado de Curvatura** (**Curvatura**) **de cada vértice**. Esto es calculado usando el valor absoluto de la curvatura en cada vértice y nos permite distinguir entre vértices muy puntiagudos y vértices poco puntiagudos.
- **Número de Puntos Característicos**. El número de vértices (esquinas) de un contorno es una característica básica del mismo.
- **Signo del valor de la curvatura de cada vértice**. Esto nos permite ver si el contorno sigue una trayectoria cóncava o convexa. En realidad, la importancia de esto está en la distribución de signos entre los vértices hallados.
- **Distancia entre vértices**. Esto nos permite distinguir entre figuras distintas pero que tengan igual número de vértices e igual distribución de signos en su curvatura (como entre un cuadrado y un rectángulo).

Resumiendo y desde un punto de vista a más alto nivel, se trata de realizar una aplicación que, dada una imagen, nos permita encontrar en una base de datos de imágenes aquella o aquellas que más se le parecen. Supondremos que cada imagen contiene un único objeto y que el contorno del mismo es suficiente para representarlo. Para la comparación

entre esas características de los contornos se efectuará una comparación difusa a través de lenguaje FSQL (*Fuzzy SQL*).

1.3. Organización de la Memoria

Este documento se organiza como sigue. El próximo capítulo mostrará las nociones y conceptos básicos sobre las imágenes y su tratamiento, centrándonos exclusivamente en aquello que será necesario para su caracterización. El Capítulo 3 mostrará una introducción a la lógica difusa, bases de datos difusas y el lenguaje FSQL. El Capítulo 4 profundizará sobre el proceso de recuperación de imágenes y detallará cada una de las fases que lo componen. El Capítulo 5 nos mostrará los detalles de diseño e implementación de la aplicación *FuzzyFinder*, desarrollada en este proyecto. El Capítulo 6 incluye los manuales de instalación y de usuario de dicha aplicación. El Capítulo 7 discutirá y valorará los resultados obtenidos en el proyecto. Por último, el Capítulo 8 mostrará las conclusiones finales y las líneas futuras que se proponen para continuar este trabajo. Además, incluimos la bibliografía utilizada para el desarrollo de todo el proyecto.

2. LAS IMÁGENES Y SU TRATAMIENTO

En esta sección vamos a profundizar sobre los temas de tratamiento de imágenes que vamos a utilizar durante el desarrollo del proyecto. En primer lugar, analizaremos los campos de aplicación del procesamiento de imágenes. Tras ello, describiremos la representación de la imagen digital utilizada. Después hablaremos de las etapas de las que se compone el procesamiento de imágenes. A continuación, introduciremos algunos conceptos que serán utilizados durante el proyecto (vecinos de un píxel y distancia entre píxeles). Tras esto, explicaremos en qué consiste la técnica de aumento de contraste, qué es y cómo se utiliza un histograma y analizaremos el proceso de umbralización, con el objetivo de segmentar la imagen, es decir, separar adecuadamente objeto de fondo. Por último, analizaremos a fondo la función de curvatura de un contorno, que nos va a permitir obtener los puntos más representativos de la imagen (puntos característicos).

2.1 Campos de Aplicación del Procesamiento de Imágenes

Las técnicas de procesamiento digital de imágenes [GoWo96] se emplean actualmente para resolver problemas muy diversos. Aunque a menudo parecen inconexos, estos problemas requieren normalmente métodos capaces de realzar la información de las imágenes para la interpretación y el análisis humano. En el campo de la medicina, por ejemplo, los procedimientos informatizados realzan el contraste o codifican los niveles de intensidad en colores para facilitar la interpretación de las imágenes de rayos X y de otras imágenes biomédicas. Los geógrafos emplean las mismas o similares técnicas para estudiar los patrones de polución a partir de imágenes aéreas o de satélites.

Los procedimientos de mejora de las imágenes y de restauración se emplean para procesar imágenes degradadas de objetos irrecuperables o bien resultados experimentales demasiado costosos para ser duplicados. En arqueología, los métodos de procesamiento de imágenes han servido para restaurar con éxito imágenes borrosas que eran los únicos registros existentes de piezas extrañas perdidas o dañadas después de haber sido fotografiadas. En la física y en campos afines, las técnicas de ordenador realzan de forma rutinaria imágenes de experimentos en áreas como los plasmas de alta energía y la

microscopía del electrón. De forma similar, los conceptos del tratamiento de imágenes se aplican con éxito en astronomía, biología, medicina nuclear, investigaciones judiciales, defensa y aplicaciones industriales.

2.2 Representación Digital de Imágenes

El término *imagen monocroma* [GoWo96] o simplemente *imagen* se refiere a una función bidimensional de intensidad de luz *f(x,y)*, donde *x* e *y* representan las coordenadas espaciales y el valor de *f* en un punto cualquiera *(x,y)* es proporcional al brillo (o *nivel de gris*) de la imagen en ese punto. La Figura 2.1 ilustra el convenio de ejes que se utilizará a lo largo de este documento.

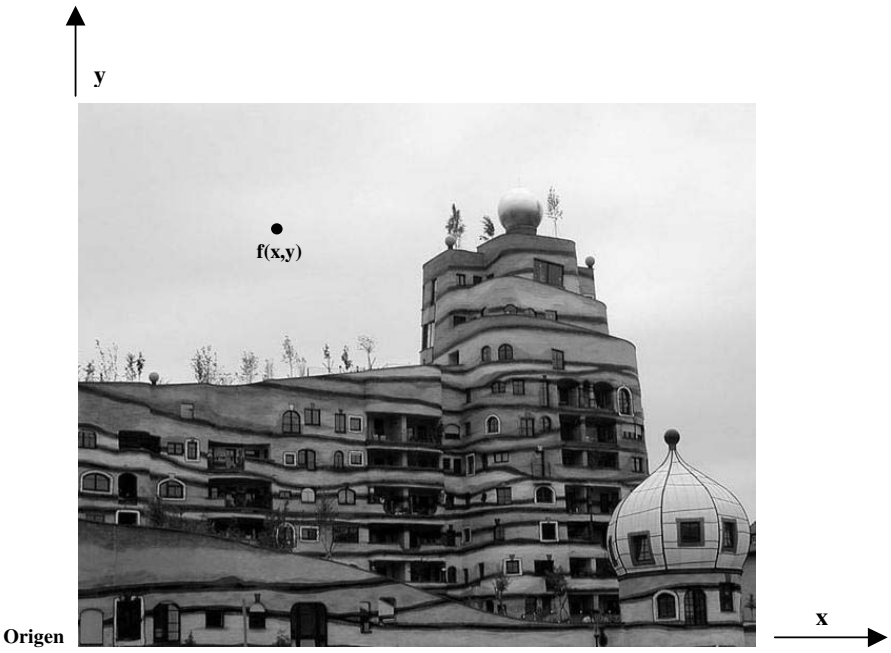


Figura 2.1: Convenio de ejes utilizado para la representación de imágenes en este proyecto. Foto: Hundertwasser

Una *imagen digital* es una imagen *f(x,y)* que se ha discretizado tanto en las coordenadas espaciales como en el brillo. Una imagen digital puede considerarse como una matriz cuyos índices de fila y columna identifican un punto de la imagen y el valor del correspondiente elemento de la matriz indica el nivel de gris en ese punto. Los elementos de una distribución digital de este tipo se denominan *elementos de la imagen*, o más comúnmente píxeles, abreviatura de su denominación inglesa "*picture elements*".

El modelo de color utilizado para representar el brillo de cada píxel en este proyecto es el *modelo RGB*, en el cual, cada color aparece con sus componentes espectrales primarias de rojo, verde y azul. Este modelo está basado en un sistema de coordenadas cartesianas. El subespacio de color de interés es el cubo mostrado en la Figura 2.2, en la que los valores RGB están en tres vértices; el *cyan*, magenta y amarillo en otros tres vértices; el negro, en el origen; y el blanco, en el vértice opuesto al origen. En este modelo la escala de grises se extiende del negro al blanco a lo largo de una diagonal del cubo, y los colores son puntos del cubo o de su interior, definidos por vectores que se extienden desde el origen. Por conveniencia, se supone que todos los valores de color han sido normalizados, de forma que el cubo de la Figura 2.2 es el cubo unidad. Es decir, todos los valores de R, G y B se supone que están en el intervalo [0,1].

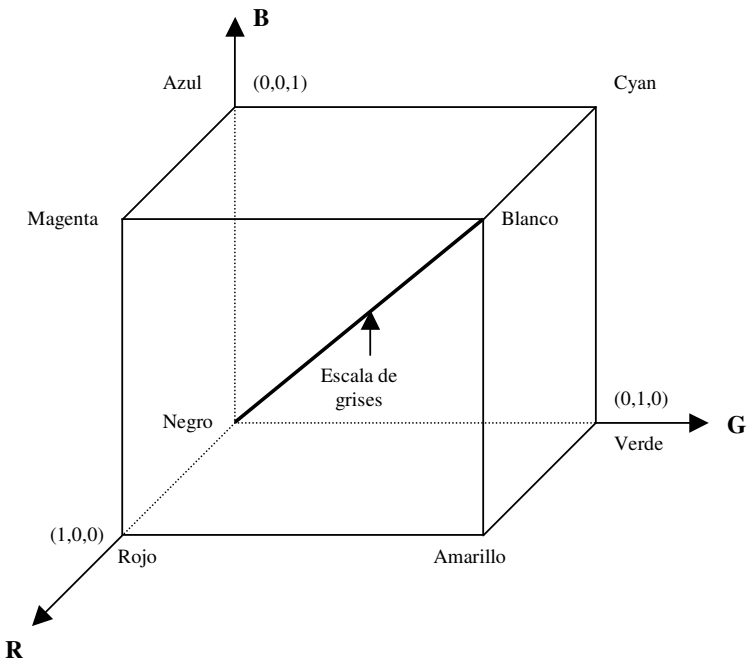


Figura 2.2: Cubo de valores RGB. Los puntos sobre la diagonal principal corresponden a valores de gris, desde el negro, en el origen, hasta el blanco, en el punto (1,1,1).

2.3 Etapas Fundamentales del Procesamiento de Imágenes

El tratamiento digital de imágenes [GoWo96] comprende un amplio rango de hardware, software y recursos teóricos. En esta sección se van a presentar las etapas fundamentales que conlleva el procesamiento de una imagen.

La Figura 2.3 muestra que el objetivo global es producir un resultado a partir de un determinado problema por medio del procesamiento de imágenes.

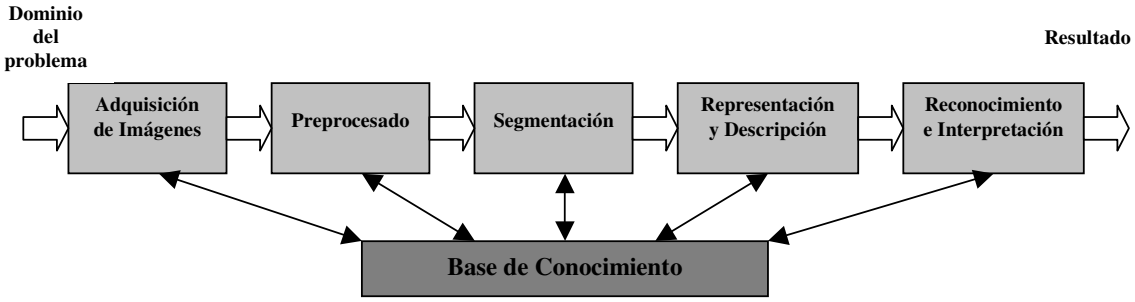


Figura 2.3: Etapas fundamentales del procesamiento digital de imágenes

La primera etapa del proceso es la *adquisición de imágenes*, es decir, la adquisición de una imagen digital. Para ello se necesita un sensor de imágenes y la posibilidad de digitalizar la señal producida por el sensor.

Una vez que se ha obtenido la imagen digital, la siguiente etapa trata del *preprocesamiento* de esa imagen. La función básica del preprocesamiento es la de mejorar la imagen de forma que se aumenten las posibilidades de éxito en los procesos posteriores. Un preprocesamiento típico consiste en utilizar técnicas para mejorar el contraste o eliminar ruidos.

La siguiente etapa es la *segmentación*. Definida de una forma general, la *segmentación* consiste en partir una imagen de entrada en sus partes constituyentes u objetos. En general, la segmentación autónoma es una de las labores más difíciles del tratamiento digital de imágenes. Por una parte, un procedimiento de segmentación demasiado tosco dilata

la solución satisfactoria de un problema de procesamiento de imágenes. Por otra, un algoritmo de segmentación débil o errático casi siempre garantiza que tarde o temprano habrá un fallo.

A la salida del proceso de segmentación, habitualmente tendremos o bien el contorno de una región o bien todos los puntos de una región determinada. En cada caso es necesario convertir los datos a una forma adecuada para el procesamiento por computador. La primera decisión que hay que tomar es si los datos se han de representar como un contorno o como una región completa. La representación como un contorno es la adecuada cuando el interés radica en las características de la forma exterior, como esquinas e inflexiones. La representación regional es adecuada cuando el interés se centra en propiedades internas, como por ejemplo la textura. Sin embargo, en algunas aplicaciones ambas representaciones coexisten.

La elección de una representación es sólo una parte de la solución para transformar los datos a una forma adecuada para ser posteriormente tratados por computador. También debe especificarse un método para describir los datos de forma que se resalten los rasgos de interés. La *descripción*, también denominada *selección de rasgos*, consiste en extraer rasgos con alguna información cuantitativa de interés o que sean fundamentales para diferenciar una clase de objetos de otra.

La última etapa de la Figura 2.3 incluye el reconocimiento e interpretación. El *reconocimiento* es el proceso que asigna una etiqueta a un objeto basándose en la información proporcionada por sus descriptores. La *interpretación* implica asignar significado a un conjunto de objetos reconocidos.

Hasta ahora no se ha dicho nada sobre la necesidad del conocimiento previo o sobre la interacción entre la base de conocimiento y los módulos de procesamiento de la Figura 2.3. El conocimiento sobre un dominio del problema está codificado en un sistema de procesamiento de imágenes como una base de datos de conocimiento, que puede ser muy compleja.

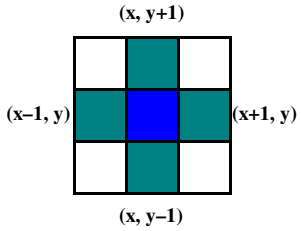
2.4 Algunas Relaciones Básicas entre Píxeles

En esta sección se considerarán algunas relaciones básicas [GoWo96] pero importantes entre los píxeles de una imagen digital. Como se mencionó anteriormente, una imagen se nota por *f(x,y)*. Cuando se haga referencia a un píxel en particular, se emplearán letras minúsculas, como *p* y *q*.

2.4.1 Vecinos de un Píxel

Un píxel *p* de coordenadas *(x,y)* tiene cuatro vecinos horizontales y verticales cuyas coordenadas vienen dadas por:

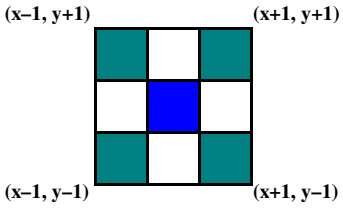
$$(x + 1, y), (x - 1, y), (x, y + 1), (x, y - 1)$$



Este conjunto de píxeles, denominado los *4-vecinos* de *p*, se representa por $N_4(p)$. Cada píxel está a una unidad de distancia (véase siguiente apartado) de *(x,y)*, y algunos de los vecinos de *p* caen fuera de la imagen digital si *(x,y)* está en el borde de la imagen.

Los cuatro vecinos en *diagonal* de *p* tienen las coordenadas:

$$(x + 1, y + 1), (x + 1, y - 1), (x - 1, y + 1), (x - 1, y - 1)$$



y se representan por $N_D(p)$. Estos puntos, junto a los 4-vecinos, se denominan los *8-vecinos* de *p*, y se representan por $N_8(p)$. Al igual que antes, algunos puntos de $N_D(p)$ y $N_8(p)$ caen fuera de la imagen si *(x,y)* está en el borde de la misma.

2.4.2 Medida de Distancia

Para los píxeles p, q y z, de coordenadas (x,y), (s,t) y (u,v) respectivamente, D es una *función distancia* o una *métrica* si:

a) $D(p,q) \geq 0$ ($D(p,q) = 0$ si y sólo si $p = q$),
b) $D(p,q) = D(q,p)$, y
c) $D(p,z) \leq D(p,q) + D(q,z)$.

La distancia que vamos a utilizar (en la obtención del contorno) es la denominada distancia D_8 (o también *distancia de tablero de ajedrez*) entre p y q, que se define como:

$$D_8(p,q) = \text{máx}(|x - s|, |y - t|)$$

Los píxeles a una distancia D_8 de (x,y) menor o igual que un cierto valor r forman un cuadrado centrado en (x,y). Por ejemplo, los píxeles a una distancia $D_8 \leq 2$ de (x,y) (el punto central) forman los siguientes contornos de distancia constante:

2 2 2 2 2
2 1 1 1 2
2 1 0 1 2
2 1 1 1 2
2 2 2 2 2

Por tanto, los píxeles con una distancia $D_8 = 1$ son los 8-vecinos de (x,y).

2.4.3 Conectividad

La conectividad entre píxeles es un concepto importante empleado para establecer los límites de los objetos y los componentes de áreas en una imagen. Para determinar si dos píxeles están conectados deben determinarse si son adyacentes en algún sentido (como ser 4-vecinos) y si sus niveles de gris cumplen un criterio especificado de similitud (como ser iguales). Por ejemplo, en una imagen binaria con valores 0 y 1, dos píxeles pueden ser 4-vecinos pero no estarán conectados a menos que tengan el mismo valor.

Sea V el conjunto de valores de nivel de gris empleados para definir la conectividad; por ejemplo, en una imagen binaria se tendrá $V = \{1\}$ para la conectividad entre píxeles con valor 1. En una imagen con escala de grises, para la conectividad entre píxeles con un rango de valores de intensidad de, por ejemplo, 32 a 64, se tiene $V = \{32, 33,..., 63, 64\}$.

Se consideran tres tipos de conectividad:

a) *4-conectividad*: Dos píxeles p y q con valores dentro de V están 4-conectados si q pertenece a $N_4(p)$.

b) *8-conectividad*: Dos píxeles p y q con valores dentro de V están 8-conectados si q pertenece a $N_8(p)$.

c) *m-conectividad* (*conectividad mixta*): Dos píxeles p y q con valores dentro de V están m-conectados si:

   i) q pertenece a $N_4(p)$, o bien

   ii) q pertenece a $N_D(p)$ y, además el conjunto $N_4(p) \cap N_4(q)$ es vacío.

Por consiguiente, un píxel p se considera *adyacente* a un píxel q si ambos están conectados.

Sea R un subconjunto de píxeles de una imagen M, $R = \{M(i_1, j_1), ..., M(i_k, j_k)\}$. R es *conexo-4* si la clausura reflexiva transitiva en R de la relación "ser 4-vecino" es la relación total en R. Es decir, si para todo par de elementos de R $M(i_p, j_p)$, $M(i_q, j_q)$ existe una cadena de elementos de R tales que el primero es $M(i_p, j_p)$, el último es $M(i_q, j_q)$, y cada elemento está *4-conectado* (relación 4-vecinos) al anterior. Análogamente se define la *8-conexión*.

2.5 Aumento de Contraste

Las imágenes con poco contraste [GoWo96] pueden ser debidas a diversas causas, como iluminación deficiente, falta de rango dinámico en el sensor o incluso incorrecta selección de la apertura de la lente durante la captación de la imagen. Esta técnica, como otras muchas existentes en el procesamiento de imágenes, son técnicas denominadas de *procesamiento punto a punto*, ya que se basan únicamente en la intensidad de píxeles individuales. En lo que

sigue indicaremos como r y s la intensidad de los píxeles antes y después del procesamiento, respectivamente.

La idea subyacente en las técnicas de aumento del contraste consiste en incrementar el rango dinámico de los niveles de gris de la imagen que se está procesando. La Figura 2.4 muestra una transformación típica empleada para la mejora del contraste. La ubicación de los puntos (r_1, s_1) y (r_2, s_2) controla la forma de la función de transformación. Por ejemplo, si $r_1 = s_1$ y $r_2 = s_2$, la transformación es una función lineal que no produce cambios en los niveles de gris. Si $r_1 = r_2$, $s_1 = 0$ y $s_2 = L - 1$, la transformación se convierte en una función umbral (véase Sección 2.7) que crea una imagen binaria. Los valores intermedios de (r_1, s_1) y (r_2, s_2) producen varios grados de dispersión de los niveles de gris de la imagen de salida, afectando de esta forma al contraste.

En general, se supone que $r_1 \leq r_2$ y $s_1 \leq s_2$, de forma que la función sea de valor único y monótonamente creciente. Esta condición mantiene el orden de nivel de gris, y de esta forma evita la creación de extrañas distribuciones de intensidad en la imagen procesada. En la Figura 4.4(a) podemos ver una imagen con muy bajo contraste y en la Figura 4.4(b) veremos la imagen aumentada de contraste mediante este procesamiento punto por punto de la imagen.

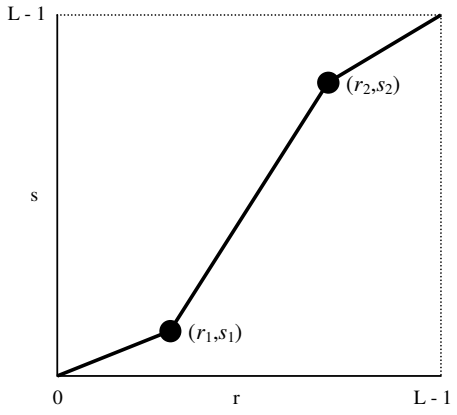


Figura 2.4: Perfil de la función de transformación

2.6 Histograma de una Imagen

El histograma de una imagen digital con niveles de gris en el rango [0,L - 1] es una función discreta $p(r_k) = n_k / n$, donde r_k es el k-ésimo nivel de gris, n_k es el número de píxeles de la imagen con ese nivel de gris, n es el número total de píxeles de la imagen y $k = 0, 1, ..., L - 1$.

De forma general se puede decir que $p(r_k)$ da una idea del valor de la probabilidad de que aparezca el nivel de gris r_k. La representación gráfica de esta función para todos los valores de k proporciona una descripción global de la apariencia de una imagen. Por ejemplo, la Figura 2.5 muestra los histogramas de cuatro tipos básicos de imágenes. El de la Figura 2.5(a) muestra que los niveles de gris están concentrados hacia el extremo oscuro del rango de la escala de gris. Así este histograma corresponde a una imagen con una apariencia global oscura. Sucede justo lo contrario en el caso de la Figura 2.5(b). El histograma mostrado en la Figura 2.5(d) tiene un perfil estrecho, lo que significa que el rango dinámico es pequeño y que por tanto la imagen tiene bajo contraste. Como todos los niveles de gris caen en la zona central de la escala de grises, la imagen aparecería como gris turbio. Finalmente, la Figura 2.5(c) muestra un histograma con una dispersión considerable, que corresponde a una imagen de alto contraste.

Aunque las propiedades que acabamos de mencionar sólo son descripciones globales que no indican nada específico sobre el contenido de la imagen, el perfil del histograma de una imagen proporciona sin duda una información muy útil sobre la posibilidad de mejora de la misma. De hecho, la utilización conjunta del histograma y la técnica de aumento de contraste (sección anterior) nos permite facilitar el proceso posterior de segmentación.

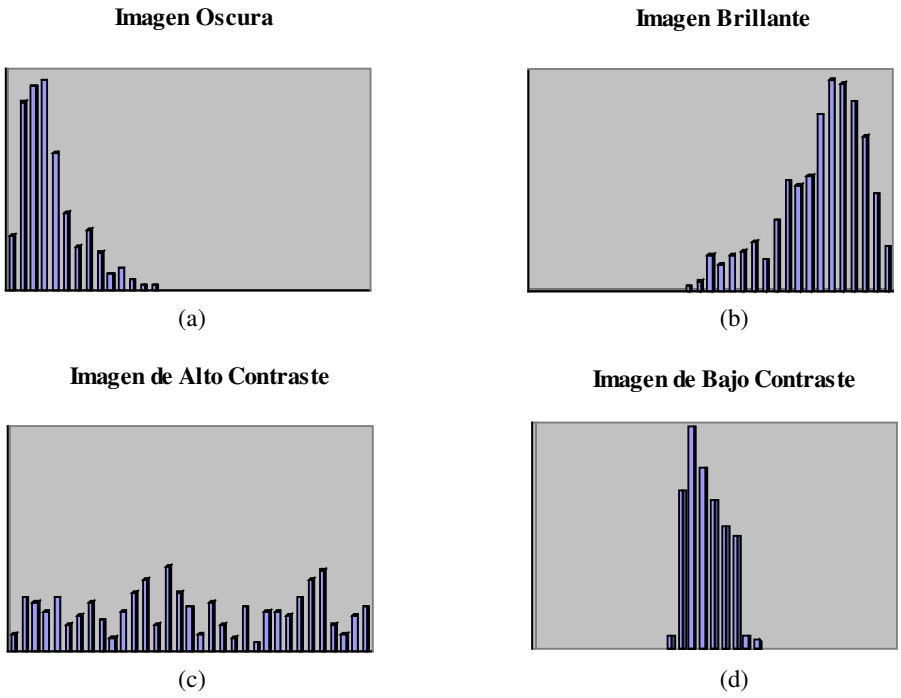


Figura 2.5: Histogramas correspondientes a cuatro tipos básicos de imágenes

2.7 Umbralización

La umbralización es uno de los métodos más importantes en el proceso de la segmentación de imágenes, que nos permite convertir una imagen con niveles de grises en una imagen binaria (por ejemplo, blanco y negro).

Supóngase que el histograma de nivel de gris de una imagen *f(x,y)*, que está compuesta de objetos luminosos sobre un fondo oscuro, de tal forma que los píxeles del objeto y el fondo tienen los niveles de gris agrupados en dos nodos dominantes. Una forma evidente de extraer los objetos del fondo es elegir un umbral *T* que separe dichos nodos. Entonces, cualquier punto *(x,y)* para el que $f(x,y) > T$ se denomina punto objeto; en caso contrario, se denomina punto del fondo.

Este tipo de umbralización se denomina *umbralización global* ("*global thresholding"*), que puede ser mostrado también con la siguiente función por trozos [Scha89]:

$$f_0(x) = \begin{cases} A & si\ f_i(x) \le T \\ B & si\ f_i(x) > T \end{cases}$$

donde f_i y f_0 son las respectivas funciones de imagen de entrada y salida. El valor del umbral T y los valores de intensidad de la imagen de salida, A y B, deben ser elegidos a priori. La elección del valor T es fundamental en el resultado final de este proceso. En general, un estudio detallado del histograma de la imagen ayuda a tomar esta decisión. Pueden consultarse varios métodos en [Scha89].

Lógicamente, tras el proceso de umbralización, la imagen obtenida será una imagen binaria (únicamente dos niveles de gris). La utilidad para el proceso de segmentación estriba en que uno de los niveles de gris representa al objeto, mientras que el otro nivel de gris representa al fondo. Ciertamente, existen otros tipos de umbralizaciones más complejas, sin embargo, ésta es la más sencilla y rápida de todas.

2.8 Obtención del Contorno: *Algoritmo de la Tortuga*

Existen muchos algoritmos que nos permiten obtener los ejes (o aristas) de una imagen. En este proyecto, una vez que hemos decidido umbralizar la imagen de manera que separamos objeto y fondo, únicamente necesitamos obtener el contorno del objeto obtenido, es decir, el borde exterior de aquella parte de la imagen con el nivel de gris asignado al objeto. Por tanto, únicamente vamos a necesitar un algoritmo que sea capaz de seguir fielmente el contorno de dicho objeto.

Por eso, el algoritmo más efectivo y eficiente para este caso es el simple *Algoritmo de la Tortuga*, cuyo pseudo-código se muestra en la Figura 2.6, si bien hay que indicar que el utilizado en este proyecto tiene ciertas modificaciones que son comentadas más adelante.

Supongamos que el preprocesamiento nos ha proporcionado una imagen binaria *M* completamente fiable, en la cual hay un objeto *R* (brillo 1) sobre un fondo uniforme (brillo 0). Supongamos también que *R* es simplemente conexo-4, es decir, no tiene agujeros, y no es parte de otra *R'* conexa-8. El sencillo *Algoritmo de la Tortuga* (o del conejo) proporciona una aproximación de la frontera-4 de *R*, es decir, el conjunto de píxeles de *R* tales que alguno de sus 4-vecinos no es de *R*.

ALGORITMO DE LA TORTUGA (*Papert*, 1973)

1. Comenzando por una esquina, buscar el primer $M(i_0,j_0)=1$.
2. Frontera <- $((i_0,j_0))$
3. Dirección = 0
4. Repetir
4.1. Si el píxel procesado es un 1,
dirección <- (dirección + 1) mod 4
En otro caso, dirección <- (dirección – 1) mod 4
4.2. i <- i + deltai(dirección)
4.3. j <- j + deltaj(dirección)
4.4. frontera <- añadir((i,j), frontera)
5. Hasta que i = i_0 y j= j_0 y dirección = 0
6. deltai(0) = deltai(2) = 0; deltai(1) = –1; deltai(3) = 1;
7. deltaj(1) = deltaj(3) = 0; deltaj(0) = 1; deltaj(2) = –1;

Figura 2.6: Pseudo-código del Algoritmo de la Tortuga

Una tortuga parte de una esquina y se mueve hasta situarse sobre un 1; entonces gira 90º a la izquierda y da un paso. Si se sitúa de nuevo sobre un 1, realiza el mismo giro y avance; por el contrario, si se sitúa sobre un 0, gira 90º a la derecha y da un paso. El camino recorrido por la tortuga desde que encontró el primer 1 hasta que vuelve a esa misma posición con la misma dirección es una *aproximación* de la frontera de la región 4-conexa a la que pertenece el primer 1 encontrado.

Las diferencias entre este algoritmo y el utilizado realmente en el proyecto son:

- Se consigue un *contorno exacto* a partir de la imagen umbralizada (binaria), no aproximado.
- Utilizamos la relación *8-vecinos* en vez de la relación 4-vecinos, que utiliza este algoritmo inicial.
- Este algoritmo sin modificar no *permite la existencia de huecos* (o agujeros) en el interior de los objetos. El algoritmo modificado sí lo permite, aunque obteniendo únicamente el contorno exterior de la forma.

2.9 Representación de Formas

En esta sección vamos a tratar el problema de representar la forma de una curva plana [MoMa92] que ha sido extraída de una imagen de entrada (contorno de la forma).

El problema de extraer la forma de una imagen (el problema de la *segmentación*) es, en general, un problema separado y no debería ser considerado como parte del problema de la representación de la forma, ya que el primero puede ser tratado eficazmente haciendo uso del conocimiento de la imagen y escena en cuestión.

Un método general de representación de imágenes útil en visión por computador debe hacer un reconocimiento exacto y seguro de un posible objeto. Por tanto, dicha representación debería satisfacer necesariamente una serie de criterios. A continuación, se muestra una lista con dichos criterios.

2.9.1 Criterios de Bondad de la Representación

- **Invariabilidad**: Si dos curvas tienen la misma forma, deberían también tener la misma representación.
- **Unicidad**: Si dos curvas no tienen la misma forma, deberían tener diferentes representaciones.
- **Estabilidad**: Si dos curvas tienen una pequeña diferencia en su forma, sus representaciones deberían también tener una pequeña diferencia, y si dos representaciones tienen una pequeña diferencia, las curvas que representan también deberían tener una pequeña diferencia de forma.

La importancia del *criterio de invariabilidad* radica en que garantiza que todas las curvas con la misma forma tendrán la misma representación. Por lo tanto, será posible concluir que dos curvas tienen diferente forma observando si tienen diferentes representaciones. Sin el criterio de invariabilidad, dos curvas con la misma forma podrían tener diferentes representaciones.

El *criterio de la unicidad* es importante, ya que garantiza que dos curvas con diferentes formas tendrán diferentes representaciones. Por consiguiente, será posible afirmar que dos curvas tienen la misma forma observando si tienen la misma representación. Sin el criterio de unicidad, dos curvas con diferentes formas podrían tener la misma representación.

La trascendencia del *criterio de estabilidad* radica en que garantiza que un pequeño cambio en la forma de una curva no provocará un gran cambio en su representación, y una pequeña diferencia entre dos representaciones no indicará una gran diferencia de forma entre las curvas que ellas representan.

En consecuencia, cuando dos representaciones son parecidas, las curvas que representan son parecidas en la forma, y cuando dos representaciones no se parecen, las curvas que representan no son parecidas en la forma. Cuando este criterio se satisface, la representación puede considerarse estable respecto al ruido.

Para la representación de una forma, es útil cumplir con una serie de propiedades adicionales para que sea adecuado en tareas prácticas de reconocimiento de formas en visión por computador. A continuación se muestra un listado de dichos criterios.

- **Eficiencia**: la representación debería ser eficiente para calcular y almacenar. Esto es importante, ya que podría ser necesario para un sistema de reconocimiento de objetos, con la finalidad de realizar un reconocimiento en tiempo real. Por *eficiencia* hemos de entender que la complejidad computacional debería ser un polinomio de pequeño orden en tiempo y espacio (y en el número de procesadores, si utilizamos una arquitectura computacional) en función del tamaño de la curva de entrada.
- **Facilidad de implementación**: si tenemos dos o más representaciones compitiendo, es ventajoso elegir la representación cuya implementación requiera el menor gasto de tiempo en programación y depuración.
- **Computación de propiedades de forma**: podría ser útil determinar las propiedades de una forma de una curva a partir de su representación. Por ejemplo, si una curva tiene una forma simétrica, sería deseable poder determinar esto desde su representación, (*criterio de simetría*). Además, si la forma de una curva, o de parte de

ella, es igual a la forma de una parte de la otra curva, podría ser útil determinar esa relación usando sus representaciones, (*criterio de parte-todo*).

2.9.2 Métodos de Representación de Formas

Los métodos de representación de formas para curvas planas existentes en visión por computador *no satisfacen todos los criterios* mencionados en la sección anterior. Existe un gran número de métodos, sin embargo, se observa que cada uno podría ser bastante adecuado para tareas especiales de reconocimiento y de representación de formas. Algunos ejemplos de estos métodos son la *Transformada de Hough*, que se ha usado para detectar líneas, círculos, y formas arbitrarias, *aproximaciones de curvas* (*regresión por mínimos cuadrados* y *regresión robusta*) representaciones que utilizan *segmentos de curvatura constante*, representaciones con *B-Splines* o *Descriptores de Fourier*.

Otra representación de las curvas planas muy utilizada es la *función de curvatura*. Se trata de calcular el grado de curvatura en cada punto del contorno. Esto puede realizarse de distintos modos. Uno de ellos es el propuesto por Mokhtarian y Mackworth, mediante el cual se obtiene una función discreta del mismo tamaño que el contorno. Aquellos puntos con alta curvatura se corresponden con esquinas en la forma, mientras que los puntos con curvatura nula (o prácticamente nula) deberán coincidir con los puntos intermedios de líneas rectas.

Sea $C = \{p_i = (x_i, y_i), i = 1, ..., n\}$ la secuencia de n puntos que describen el contorno de una curva cerrada C [PeFe93]. El proceso de discretización introduce ruido en los valores de las coordenadas y no todos los métodos se comportan de manera adecuada ante la presencia de dicho ruido. Mokhtarian y Mackworth propusieron calcular el vector de curvatura de la curva plana que representa el contorno de la forma, mediante la convolución de dicha curva con una función gaussiana (con desviación típica σ) de la misma dimensión que las curvas $x(t)$ e $y(t)$. De esta manera se controla el ruido de cuantización (debido a que el ángulo entre píxeles vecinos sólo puede variar en incrementos de 45º, ya que el vector de entrada es digital) y también reducir la sensibilidad de la curvatura al ruido presente en la frontera.

La expresión que calcula la curvatura es la siguiente, teniendo en cuenta que $\dot{z}$ y $\ddot{z}$ representan la primera y segunda derivada, utilizando la gaussiana para suavizar la curva:

$$k = \frac{\dot{x}\ddot{y} - \ddot{y}\ddot{x}}{(\dot{x}^2 + \dot{y}^2)^{3/2}} \qquad (2.1)$$

Por tanto, la extracción de los puntos de mayor curvatura de este vector, nos permitirán caracterizar a esa imagen, ya que podremos obtener una serie de propiedades a través de ellos (su curvatura, la distancia entre esos puntos, su signo, etc.).

La representación es prácticamente invariante a rotación, a cambios de escala uniforme y a la traslación de la curva. Esto y algunas otras propiedades la hacen adecuada para reconocer una curva ruidosa de una forma arbitraria a cualquier escala u orientación.

2.9.3. Conclusiones

A continuación, presentamos una evaluación de la representación de la forma utilizando la función de curvatura de acuerdo con los criterios expuestos anteriormente:

- **Criterio de la invariabilidad**

Recuérdese que por invariabilidad entendemos que la representación de la forma de una curva no debería cambiar cuando aplicamos a la curva transformaciones que preserven la forma (*rotación*, *escala uniforme* y *traslación*).

- La *traslación* de la curva no produce cambios en las representaciones de la curva utilizadas.
- El *cambio de escala uniforme* hace que la representación sufra cambios también. Sin embargo, un posterior tratamiento adecuado de la información obtenida de la representación puede preservar esta invariabilidad.
- La *rotación* causa únicamente un desplazamiento horizontal de la representación. Sin embargo, un posterior tratamiento adecuado de la

información puede determinar el desplazamiento entre las dos representaciones a comparar.

- **Unicidad**

El criterio de unicidad requiere que dos curvas con diferentes formas tengan diferentes representaciones. Esto es necesario para poder ser capaces de reconocer dos o más curvas con la misma forma a través de la observación de sus representaciones.

Como demuestran Mokhtarian y Mackworth en [MoMa92] (*Teorema 6*), una curva plana puede ser reconstruida a partir de cualquiera de sus representaciones y, por tanto, se satisface el criterio de unicidad.

- **Estabilidad**

El criterio de estabilidad se cumple, como demuestran Mokhtarian y Mackworth en [MoMa92] (*Teorema 3*). Además, las observaciones realizadas nos muestran que un pequeño cambio en la desviación estándar de la gaussiana (σ) da como resultado un pequeño cambio en la localización de los puntos de la curva con curvatura nula.

Por tanto, los experimentos realizados muestran que las representaciones son estables respecto al ruido *uniforme* y *no uniforme* en las curvas que representan, por lo que, se satisface el criterio de estabilidad.

El resto de criterios (*eficiencia*, *facilidad de implementación* y *cálculo de propiedades de la forma*) son satisfechos por esta representación de la forma.

3. BASES DE DATOS DIFUSAS y LENGUAJE FSQL

En este capítulo vamos, en primer lugar, a realizar una introducción a los términos más importantes de la lógica difusa (*fuzzy logic*), destacando los aspectos que son utilizados en este proyecto. Seguidamente, hablaremos sobre las bases de datos difusas como extensión a las bases de datos tradicionales, indicando las mejoras y los aspectos más destacados de ellas. Por último, nos centraremos en el lenguaje *FSQL*, diseñado e implementado por J. Galindo, que trata de extender el lenguaje de consultas *SQL* para que permita realizar consultas difusas mediante la creación previa de tablas con atributos difusos, que serán utilizados en este proyecto y que constituirán el eje central de las comparaciones entre imágenes.

Hay que mencionar que casi toda la información que aquí se muestra está obtenida de la tesis doctoral de José Galindo Gómez [Gali99]. Además, el primero de los apartados se ha obtenido fundamentalmente del capítulo primero del Proyecto Fin de Carrera de Ingeniería Técnica Industrial en Electrónica de la Universidad de Málaga titulado "*Software para Control Difuso de todo tipo de Sistemas*" [Esco03] desarrollado por Calixto Escobar Rodríguez y dirigido por José Galindo Gómez.

3.1 Lógica Difusa (Fuzzy Logic): Teoría de Conjuntos Difusos

En esta sección se trata de exponer las nociones fundamentales, operaciones, conceptos, métodos y principios subyacentes de la lógica difusa que le permitan situarse en el marco de una lógica multivaluada *(Many-Valued Logic)*, donde no sólo una expresión puede ser *cierta* o *falsa*, sino que permite cuantificar en qué medida lo es.

Antes de comenzar describiendo brevemente la lógica difusa y su origen preguntémonos ¿qué significa *fuzzy*?, término sobre el cual se sustenta una forma de expresar las leyes, modos y formas del conocimiento científico (lógica).

Originalmente el término *fuzzy* procede de *fuzz*, que sirve para denominar la pelusa que recubre el cuerpo de los polluelos al poco de salir del huevo. Este término en inglés significa "confuso, borroso, no definido o desenfocado". La traducción de esta palabra al castellano es difuso o borroso, aunque *fuzzy*, en los ámbitos académico y tecnológico, está aceptado tal cual, de forma similar a como lo es *"bit"*.

La lógica difusa nació cuando el Profesor Lotfi A. Zadeh publicó un artículo titulado *"Fuzzy Sets"* (Conjuntos Difusos) [Zade65]. En este artículo el Dr. Zadeh presentó unos conjuntos sin límites precisos los cuales, según él, juegan un importante papel en el reconocimiento de formas, interpretación de significados, y especialmente en la abstracción, la esencia del proceso de razonamiento del ser humano.

En la lógica clásica sólo es posible tratar información que sea totalmente cierta o totalmente falsa; no le es posible manipular aquella información imprecisa o incompleta inherente a un problema y como información que es, contiene datos que permitirían una mejor resolución del mismo. Con ello se podría decir que la lógica difusa es una extensión de los sistemas clásicos, como el propio Zadeh indica en [Zade92]. La lógica difusa es la lógica que soporta modos de razonamiento aproximados en lugar de exactos. Su importancia radica en que muchos modos de razonamiento humano, en especial el razonamiento según el sentido común, son aproximados por naturaleza.

Esta lógica es una lógica multivaluada y sus características principales, presentadas por Zadeh en la referencia antes mencionada, son:

- En la lógica difusa, el razonamiento exacto es considerado como un caso particular del razonamiento aproximado.
- Cualquier sistema lógico puede ser trasladado a términos de lógica difusa.
- En lógica difusa, el conocimiento es interpretado como un conjunto de restricciones flexibles, es decir, difusas, sobre un conjunto de variables.
- La inferencia (proceso mediante el cual se llega a un resultado, se obtienen consecuencias o se deduce una cosa de otra) es considerada como un proceso de propagación de dichas restricciones.
- En lógica difusa, todo problema es un problema de grados.

La lógica difusa se ha convertido en un tema muy común en control de máquinas como el resultado de hacerlas más "capaces" y "responsables". Se podría decir que la lógica difusa permite a los ordenadores trabajar no sólo con métodos cuantitativos sino también cualitativos, se trata pues de un intento de aplicar una forma más humana de pensar en la programación de computadoras.

En esta sección, vamos a introducir algunas nociones elementales sobre la teoría de conjuntos difusos. En este resumen nos detendremos en los aspectos semánticos y de representación relacionados con esta potente herramienta. En la literatura podemos encontrar una gran cantidad de trabajos sobre esta teoría, como en [YaOv87] donde se puede encontrar una recopilación de algunos de los artículos más interesantes publicados sobre el tema por L.A. Zadeh. En [DuPr80, DuPr88, Zimm91] es posible encontrar recopilados los aspectos más importantes que constituyen la teoría de conjuntos difusos así como la teoría de la posibilidad. Una más moderna síntesis de los conjuntos difusos y sus aplicaciones puede verse en [KrGe94, McTh94, MoVa93] y, sobre todo, en [PeGo98].

3.1.1 Conjuntos Difusos (*Fuzzy Sets*)

Los conjuntos difusos son una generalización de los (sub)conjuntos clásicos en el sentido de que los amplían pues permiten la descripción de nociones "vagas" e "imprecisas". Relajan la restricción de los conjuntos clásicos de pertenencia o no-pertenencia absoluta al mismo. Es necesario hacer notar que muchos de estos conceptos con naturaleza "imprecisa", si no todos, responden a criterios subjetivos. Esto es, la definición de esa imprecisión depende en mayor o menor medida de la persona que la expresa. Dicha generalización nos lleva a que:

- La pertenencia de un elemento a un conjunto pasa a ser un concepto *"difuso o borroso"*. Para algunos elementos puede no estar clara su pertenencia o no al conjunto.

- Dicha pertenencia puede ser cuantificada por un grado. Dicho grado se denomina habitualmente como *grado de pertenencia* de dicho elemento al conjunto y toma un valor en el intervalo $[0,1] \in \Re$ por convenio.

Nótese la gran potencialidad que presenta este último punto al *permitir expresar de forma cuantitativa algo cualitativo* (difuso) mediante el grado de pertenencia. De forma más formal podemos definir un conjunto difuso como sigue:

Definición 3.1 *Un **Conjunto Difuso** A sobre un universo de discurso Ω (intervalo finito o infinito dentro del cual el conjunto difuso puede tomar un valor) es un conjunto de pares*

$$A = \{\mu_A(x) / x : x \in \Omega, \mu_A(x) \in [0,1] \in \Re\}$$

*donde $\mu_A(x)$ se denomina **grado de pertenencia** del elemento x al conjunto difuso A. Este grado oscila entre los extremos **0** y **1** del dominio de los nº reales:*

$\mu_A(x) = 0$ *indica que x no pertenece en absoluto al conjunto difuso A.*
$\mu_A(x) = 1$ *indica que x pertenece totalmente al conjunto difuso A.*

*A veces, en vez de dar una lista exhaustiva de todos los pares que forman el conjunto (valores discretos), se da una definición para la función $\mu_A(x)$, llamada función característica o **función de pertenencia.***

□

Si la función de pertenencia sólo produce valores del conjunto {0,1}, entonces el conjunto que genera no es difuso, sino *crisp* (concreto, preciso).

De la definición de conjunto difuso se derivan los siguientes conceptos:

- **Universo de Discurso:** Como se ha mencionado anteriormente, existen dos posibilidades a la hora de establecer el intervalo de valores válido para el conjunto difuso y son:

  - A través de un universo de discurso *finito o discreto:*

$$\Omega = \{x_1, x_2, \ldots, x_n\}$$

un conjunto difuso A se puede representar como:

$$A = \mu_1 / x_1 + \mu_2 / x_2 + \ldots + \mu_n / x_n \qquad (3.1)$$

donde μ_i representa el grado de pertenencia del elemento x_i, con i =1,2,..,n. Habitualmente los elementos con grado cero no se listan. Aquí el operador "+" no hace el papel de la suma aritmética sino que tiene el sentido de agregación y " / " no es el operador de división sino que tiene el significado de asociación de ambos valores.

- Expresar el conjunto difuso a través de su función de pertenencia en un universo de discurso *infinito*. Así, un conjunto difuso A sobre Ω puede representarse como:

$$A = \int \mu_A(x) / x \qquad (3.2)$$

- **Etiqueta Lingüística:** Es aquella palabra en lenguaje natural, que expresa o *identifica a un conjunto difuso*, que puede estar formalmente definido o no. Así la función de pertenencia de un conjunto difuso A, $\mu_A(x)$, expresa el grado en que *x* verifica la categoría especificada por A.

  Con esta definición, podemos asegurar que en nuestra vida cotidiana utilizamos multitud de etiquetas lingüísticas para expresar conceptos abstractos: "joven", "viejo", "frío", "caliente", "barato", "caro", "limpio", "sucio"...

  Además, la definición intuitiva de esas etiquetas, no sólo puede variar de un individuo a otro y del momento particular, sino que también varía del contexto en el que se aplique. Por ejemplo, con toda seguridad no medirán la misma altura una persona "alta" y un edificio "alto".

La representación de conjuntos difusos puede ser variada y depende fundamentalmente de la naturaleza del universo de discurso (establece el contexto) sobre el que definamos el conjunto difuso.

Ejemplo 3.1. Si expresamos el concepto cualitativo *joven* mediante un conjunto difuso, donde el eje de abcisas (eje x) representa el universo de discurso *edad* y el eje de ordenadas (eje y) representa los grados de pertenencia en el intervalo [0,1], el conjunto difuso que representa dicho concepto podría expresarse en la forma siguiente, considerando un universo discreto:

$$Joven = \{1/0,..., 1/20, 1/25, 0.9/26, 0.8/27, 0.7/28, 0.6/29, 0.5/30,..., 0.1/34\}$$

La *edad* (en años enteros) sería el universo de discurso de *joven*. La etiqueta lingüística *joven* identificaría a este conjunto difuso representado por una función de pertenencia si consideramos un universo de discurso no discreto, de otros como *adulto*, *viejo*, etc. y así. Véase Figura 3.1.

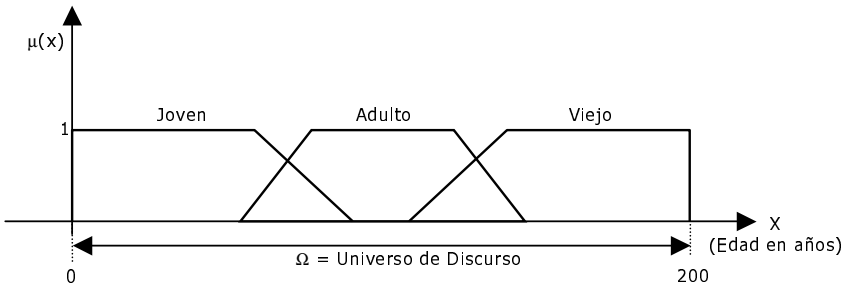


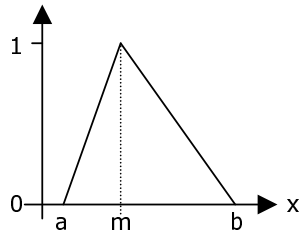
Figura 3.1: Gráfico que ilustra tres Etiquetas Lingüísticas (Ejemplo 3.1).

3.1.2 Tipos de Funciones de Pertenencia

Según la forma de la función de pertenencia, tendremos distintas clases de conjuntos difusos, que se pueden clasificar en dos grupos, las formadas por líneas rectas *lineales* y las que presentan formas *curvas*. Aquí únicamente vamos a mostrar dos de las funciones *lineales*, aunque solamente se usará en nuestra aplicación la función *triangular*, con la restricción de que *a* y *b* tienen la misma separación respecto al valor central, y dicha separación se denomina *margen*.

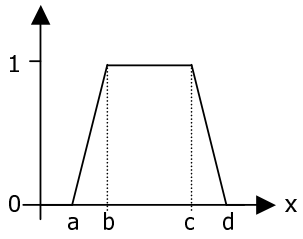
1. *Triangular*: Definido por sus límites inferior *a* y superior *b*, y el valor modal *m*, tal que *a<m<b*.

$$A(x)=\begin{cases}0 & si\ \ x\le a\\ (x-a)/(m-a) & si\ \ x\in(a,m]\\ (b-x)/(b-m) & si\ \ x\in(m,b)\\ 1 & si\ \ x\ge b\end{cases}$$



2. *Función Trapezoidal*: Definida por sus límites inferior *a* y superior *d*, y los límites de su *núcleo* (ver Definición 3.8), *b* y *c*, inferior y superior respectivamente.

$$T(x)=\begin{cases}0 & si\ \ (x\le a)\ \text{o}\ \ (x\ge d)\\ (x-a)/(b-a) & si\ \ x\in(a,b]\\ 1 & si\ \ x\in(b,c)\\ (d-x)/(d-c) & si\ \ x\in[c,d)\end{cases}$$



3.1.3 Conceptos sobre Conjuntos Difusos

Sobre conjuntos difusos se definen una serie de conceptos que nos permiten tratar y comparar conjuntos difusos:

- **Igualdad *(Equality)*** de conjuntos difusos sobre un mismo universo de discurso:

  Definición 3.2 *Dos conjuntos difusos A y B sobre Ω se dicen iguales si cumplen:*

$$A = B \Leftrightarrow \forall x \in \Omega,\ \mu_A(x) = \mu_B(x)$$

□

- **Inclusión *(Inclusion)*** de un conjunto difuso en otro:

  Definición 3.3 *Dados dos conjuntos difusos A y B sobre Ω, decimos que A está incluido en B si cumplen:*

$$A \subseteq B \Leftrightarrow \forall x \in \Omega,\ \mu_A(x) \leq \mu_B(x)$$

□

- **Soporte** (***Support***) de un conjunto difuso:

  Definición 3.4 *El soporte (support) de un conjunto difuso A definido sobre* Ω *es un subconjunto de dicho universo que satisface:*

$$\text{supp}(A) = \{x \in \Omega,\ \mu_A(x) > 0\}$$

□

- **α-corte** de un conjunto difuso:

  Definición 3.5 *Denotándolo por* A_α*, es un subconjunto no difuso (clásico) de elementos de* Ω*, cuya función de pertenencia toma un valor mayor o igual que algún valor concreto* α *de dicho universo de discurso que satisface:*

$$A_\alpha = \{x : x \in \Omega,\ \mu_A(x) \geq \alpha, \alpha \in [0,1]\}$$

□

El *Teorema de Representación* descrito a continuación permite representar cualquier conjunto difuso A mediante la unión de sus α-cortes. Véase Figura 3.2.

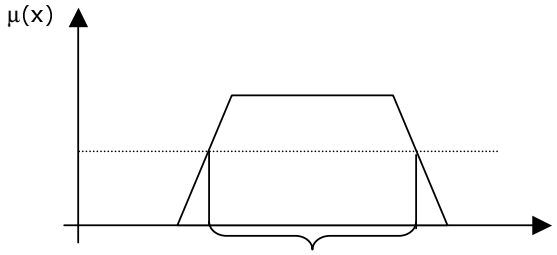


Figura 3.2: α-corte en un Trapecio

- **Teorema de Representación:**

Definición 3.6 *Todo subconjunto difuso A puede ser obtenido a partir de la unión de sus α-cortes:*

$$A = \bigcup_{\alpha \in [0,1]} A_\alpha$$

□

- **Conjunto Difuso Convexo o Cóncavo:**

Definición 3.7 *Haciendo uso del Teorema de Representación se establece el concepto de conjunto difuso como aquel en que todos sus α-cortes son convexos:*

$$\forall x, y \in \Omega, \forall \lambda \in [0,1]: \mu_A(\lambda \cdot x + (1-\lambda) \cdot y) \geq \min(\mu_A(x), \mu_A(y))$$

□

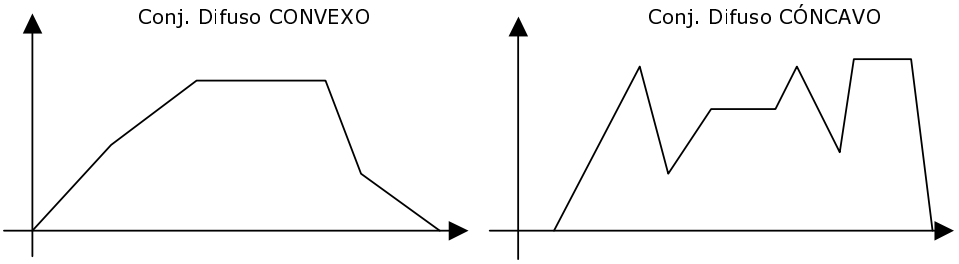


Figura 3.3: Ejemplos de Conjuntos Difusos Convexos y Cóncavos

Por último, indicar que si dos conjuntos difusos son convexos, también es convexa su intersección.

- **Núcleo *(Kernel, Core)*:**

Definición 3.8 *El núcleo de un conjunto difuso A, definido sobre Ω, es un subconjunto de dicho universo que satisface:*

$$\text{Kern}(A) = \{x \in \Omega,\ \mu_A(x) = 1\}$$

□

- **Altura *(Height)*:**

  Definición 3.9 *La altura de un conjunto difuso A, definido sobre Ω, se define como:*

$$\mathrm{Hgt}(A) = \sup_{x \in \Omega} \mu_A(x)$$

□

- Conjunto difuso **Normalizado:**

  Definición 3.10 *Un Conjunto Difuso es normalizado sí y sólo sí:*

$$\exists x \in \Omega, \mu_A(x) = \mathrm{Hgt}(A) = 1$$

□

3.1.4 Operaciones sobre Conjuntos Difusos

El hecho de que la teoría de conjuntos difusos generalice la teoría de conjuntos clásica da lugar a que los conjuntos difusos admitan las operaciones de unión, intersección y complemento. En [Petr96] podemos encontrar estas y otras operaciones, como la *concentración* (elevar al cuadrado la función de pertenencia), la *dilatación* (efectuar la raíz cuadrada de la función de pertenencia) y la *intensificación* que pueden utilizarse cuando se usan modificadores lingüísticos ("*linguistic hedges*") como "muy" o "poco".

3.1.4.1 Unión e Intersección

Definición 3.11 *Si A y B son dos conjuntos difusos sobre un universo de discurso Ω, la función de pertenencia de la* ***unión*** *de ambos conjuntos,* $A \cup B$*, viene dada por*

$$\mu(A \cup B)(x) = f(\mu_A(x), \mu_B(x)), x \in \Omega$$

donde f es una ***s-norma*** (***t-conorma***) [ScSk83].

□

Definición 3.12 *Si A y B son dos conjuntos difusos sobre un universo de discurso Ω, la función de pertenencia de la* ***intersección*** *de ambos conjuntos,* $A \cap B$*, viene dada por*

$$\mu(A \cap B)(x) = g(\mu_A(x), \mu_B(x)), x \in \Omega$$

donde g es una ***t-norma*** [ScSk83].

□

Tanto las *s-normas* como *t-normas* establecen *modelos genéricos* para las operaciones de *unión* y *intersección* respectivamente, las cuales deben cumplir ciertas propiedades básicas (conmutativa, asociativa, monotonicidad y condiciones frontera).

Definición 3.13 *Norma Triangular, t-norma: Operación binaria t:* $[0,1]^2 \rightarrow [0,1]$ *que cumple las siguientes propiedades:*

- Conmutativa: $x\ t\ y = y\ t\ x$
- Asociativa: $x\ t\ (y\ t\ z) = (x\ t\ y)\ t\ z$
- Monotonicidad: *Si* $x \leq y$ y $w \leq z$ *entonces* $x\ t\ w \leq y\ t\ z$
- Condiciones Frontera: $x\ t\ 0 = 0,\ x\ t\ 1 = x$

□

Definición 3.14 *Conorma Triangular, s-norma o t-conorma: Operación binaria s:* $[0,1]^2 \rightarrow [0,1]$ *que cumple las siguientes propiedades:*

- Conmutativa: $x\ s\ y = y\ s\ x$
- Asociativa: $x\ s\ (y\ s\ z) = (x\ s\ y)\ s\ z$
- Monotonicidad: *Si* $x \leq y,\ y\ w \leq z$ *entonces* $x\ s\ w \leq y\ s\ z$
- Condiciones Frontera: $x\ s\ 0 = x,\ x\ s\ 1 = 1$

□

T-Norma del Mínimo: La función *mín* ($\wedge$) es una t-norma, que corresponde a la operación de *intersección* en conjuntos clásicos cuyos grados de pertenencia están en {0,1}. Por eso, esta función es la extensión natural de la intersección en conjuntos difusos.

T-Conorma o S-Norma del Máximo: La función *máx* ($\vee$) es una s-norma, que corresponde a la operación de *unión* en conjuntos clásicos cuyos grados de pertenencia están en {0,1}. Por eso, esta función es la extensión natural de la unión en conjuntos difusos.

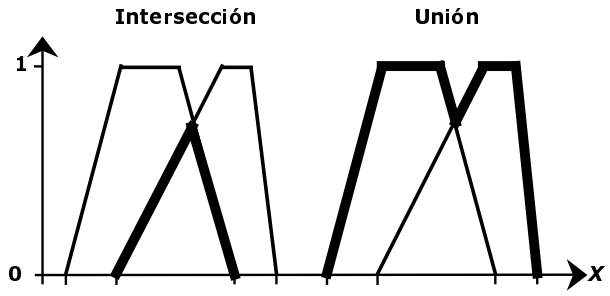


Figura 3.4: Intersección (Mínimo) y Unión (Máximo) de dos conjuntos difusos trapezoidales

Existe un amplio conjunto de operadores, denominados *T-Normas* (*Normas Triangulares*) y *T-Conormas S* (*Co-Normas Tringulares*), que pueden ser utilizados como conectivos para modelar la intersección y la unión respectivamente, como los presentados en [DuPr80, Yage80, PeGo98]. Los más importantes son:

- Operadores *Idempotentes:* El *Máximo* y el *Mínimo* para la unión y la intersección respectivamente. Satisfacen, además de la *idempotencia*, la propiedad distributiva aplicada sobre ambos y son estrictamente crecientes. Estos operadores son los más utilizados porque conservan gran cantidad de las propiedades de los operadores booleanos. En [Benz76] puede encontrarse una justificación de la elección de los operadores *Max* y *Min* para definir los anteriores conceptos.

- Operadores *Arquimedianos:* Emplean la *Suma Probabilística*, ($x+y-x*y$), y el *producto*, ($x*y$), para la unión y la intersección, respectivamente. Estos operadores no satisfacen la propiedad distributiva ni son idempotentes.

- Operadores *Acotados (Bounded):* Los operadores dados por *min(1, x+y)* y *max(0, x+y-1)*, representan la unión y la intersección respectivamente. Estos operadores no satisfacen la idempotencia, la propiedad distributiva ni la propiedad de absorción. Por el contrario, satisfacen las propiedades conmutativa, asociativa y de identidad.

3.1.4.2 Complemento o Negación

La noción de complemento se puede construir a partir del concepto de negación fuerte de [Tril79]:

Definición 3.15 *Una función C de [0,1] en [0,1] es una* ***negación fuerte*** *si satisface las siguientes condiciones:*

A. C(0)=1
B. C(C(a))=a (involución)
C. C es estrictamente decreciente
D. C es continua

□

Aunque existen varios tipos de operadores que satisfacen tales propiedades o versiones relajadas de las mismas, nosotros, para el complemento, emplearemos principalmente la versión proporcionada por Zadeh en [Zade65b], en el cual:

$$C(x) = 1 - x$$

Por tanto, para un conjunto difuso A sobre un universo de discurso Ω, la función de pertenencia del *complemento* de A, $\neg A$, viene dada por:

$$\mu_{\neg A}(x) = 1 - \mu_A(x), x \in \Omega$$

3.1.5 Números Difusos

El concepto de número difuso fue introducido por primera vez en [Zade75] con el propósito de analizar y manipular valores numéricos aproximados, como por ejemplo "próximo a 0", "casi 5", etc. El concepto ha sido refinado sucesivamente y en este trabajo entenderemos por número difuso lo siguiente [DuPr85]:

Definición 3.16 *Sea A un subconjunto difuso de* Ω *y* $\mu_A(x)$ *su función de pertenencia cumpliendo:*

1. $\forall x, y \in \Omega, \forall \mu_A(t) \geq \min(\mu_A(x), \mu_A(y))$, *es decir, que es convexo.*
2. $\mu_A(x)$ *es semicontinua superiormente.*
3. *El soporte de A es un conjunto acotado.*

entonces diremos que A es un ***número difuso*** (véase Figura 3.5).

□

Algunos autores incluyen en la definición la necesidad de que el subconjunto difuso esté normalizado.

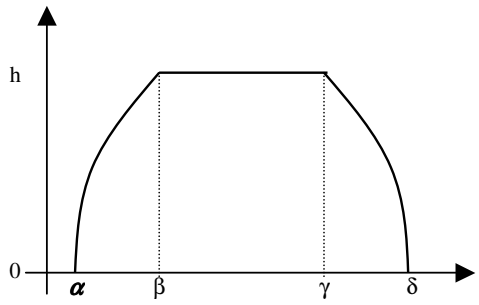


Figura 3.5: Número difuso general

La forma general de la función de pertenencia de un número difuso A, es la que sigue a continuación:

$$\mu_A(x) = \begin{cases} r_A(x) & \text{si } x \in [\alpha, \beta) \\ h & \text{si } x \in [\beta, \gamma] \\ s_A(x) & \text{si } x \in (\gamma, \delta] \\ 0 & \text{en otro caso} \end{cases} \qquad (3.3)$$

donde r_A, s_A:$\Omega \rightarrow [0,1]$, r_M no decreciente, s_M no creciente y

$$r_A(\beta) = h = s_A(\gamma) \qquad (3.4)$$

con $h \in (0,1]$ y $\alpha, \beta, \gamma, \delta \in \Omega$.

Al número *h* se le denomina *altura* del número difuso, al intervalo $[\beta,\gamma]$ *intervalo modal* y a los números β-α y δ-γ *holguras izquierda y derecha* respectivamente.

Un caso particular de números difusos que se obtiene cuando consideramos a las funciones r_A y s_A como funciones lineales. En este caso la función de pertenencia adopta la forma:

$$\mu_A(x) = \begin{cases} h + \dfrac{(x-\beta)h}{\beta-\alpha} & \text{si } x \in [\alpha, \beta) \\ h & \text{si } x \in [\beta, \gamma] \\ h - \dfrac{(x-\gamma)h}{\delta-\gamma} & \text{si } x \in (\gamma, \delta] \\ 0 & \text{en otro caso} \end{cases} \qquad (3.5)$$

A un número difuso de este tipo lo llamaremos *triangular o* ***trapezoidal***. Si trabajamos con números difusos normalizados, entonces $h = 1$; en este caso podremos caracterizar un número difuso trapezoidal normalizado A, mediante el empleo de los 4 parámetros que realmente son imprescindibles (véase Figura 3.6):

$$A \equiv (\alpha, \beta, \gamma, \delta)$$

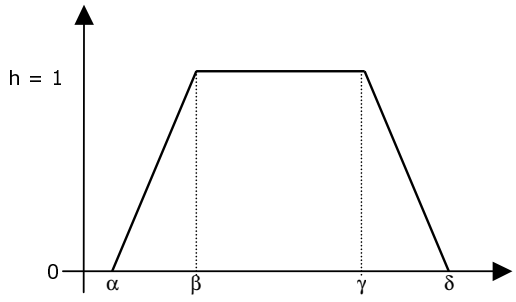


Figura 3.6: Número difuso trapezoidal normalizado

3.1.5.1 El Principio de Extensión (*Extension Principle*)

Una de las nociones más importantes en teoría de conjuntos difusos es el *principio de extensión*, propuesto en [Zade75]. Proporciona un método general que permite extender conceptos matemáticos no difusos para el tratamiento de cantidades difusas. Se usa para transformar cantidades difusas, que tengan iguales o distintos universos, según una función de transformación en esos universos. Véase una representación gráfica en la Figura 3.7.

Sea A una cantidad difusa definida sobre el universo de discurso X y *f* una función de transformación no difusa tal que *f:* X → Y. El propósito es extender *f* en A de forma que opere sobre X y devuelva una cantidad difusa B sobre Y. Dicho objetivo se obtiene por uso de la composición del *Sup-Min* como a continuación se comenta de forma generalizada en el caso del producto cartesiano de *n* universos de discursos.

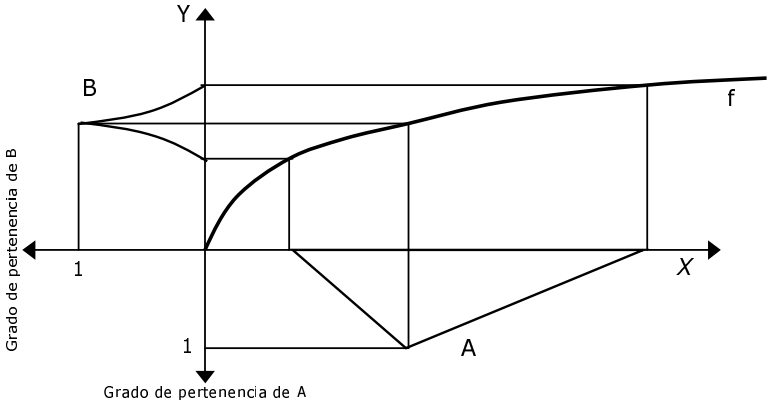


Figura 3.7: Representación gráfica del Principio de Extensión

Definición 3.17 *Sea* Ω *un producto cartesiano de universos tal que* $\Omega = \Omega_1 \times \Omega_2 \times \ldots \times \Omega_n$, *y* $A_1, A_2, \ldots, A_n$, *n conjuntos difusos de* $\Omega_1, \Omega_2, \ldots, \Omega_n$ *respectivamente, f una función desde* Ω *hasta el universo* Ω', *entonces un conjunto difuso B de* Ω' *viene definido por:*

$$B = \int_{\Omega'} \mu_B(y) / y$$

donde $y = f(A_1, A_2, \ldots, A_n)$ $(y \in \Omega')$, y

$$\mu_B(y) = \begin{cases} \underset{\substack{\Omega_1,\ldots,\Omega_n \\ f(\Omega_1,\ldots,\Omega_n)=\Omega'}}{Sup} \; Min(\mu_{A_1}(x_1),\ldots,\mu_{A_n}(x_n)) \; si \; f^{-1}(y) \neq 0 \\ 0 \qquad\qquad\qquad\qquad\qquad\qquad \text{en otro caso} \end{cases}$$

□

Ejemplo 3.2. Sean X e Y, ambos, el universo de los números naturales.

Función **sumar 4:** ***y = f(x)= x + 4:***

A = 0.1/2 + 0.4/3 + 1/4 + 0.6/5;

B = f(A) = 0.1/6 + 0.4/7 + 1/8 + 0.6/9;

Función **suma: $y = f(x_1,x_2) = x_1 + x_2$:**

A_1 = 0.1/2 + 0.4/3 + 1/4 + 0.6/5;

A_2 = 0.4/5 + 1/6;

B = f(A_1, A_2) = 0.1/7 + 0.4/8 + 0.4/9 + 1/10 + 0.6/11;

3.1.5.2 Aritmética Difusa

Gracias al *Principio de Extensión*, es posible extender las operaciones aritméticas clásicas al tratamiento de números difusos. De esta forma, las cuatro operaciones principales quedan extendidas en:

Suma Extendida: Dadas dos cantidades difusas A_1 y A_2, la función de pertenencia de la suma viene dada por la expresión:

$$\mu(A_1 + A_2)(y) = Sup\{Min(\mu_{A1}(y - x), \mu_{A2}(x)) / x \in \Omega\} \qquad (3.6)$$

De este modo, la suma queda expresada en términos de la operación del supremo. La suma extendida es una operación conmutativa, asociativa y no existe el concepto de número simétrico.

Diferencia Extendida: Dadas dos cantidades difusas A_1 y A_2, la función de pertenencia de la diferencia viene dada por la expresión:

$$\mu(A_1 - A_2)(y) = Sup\{Min(\mu_{A1}(y+x), \mu_{A2}(x)) / x \in \Omega\} \qquad (3.7)$$

De estas definiciones se puede obtener fácilmente que si A_1 tiene *n* términos y A_2 tiene *m* términos, el número de términos de $A_1 + A_2$ y de $A_1 - A_2$ es (*n*-1) + (*m*-1) + 1, o lo que es lo mismo: *n* + *m* - 1.

Producto Extendido: El producto de dos cantidades difusas $A_1 * A_2$ se obtiene:

$$\mu(A_1 \bullet A_2)(y) = \begin{cases} Sup\{Min\{\mu_{A_2}(z/y), \mu_{A_2}(y))/ y \in \Omega - \{0\}\}\} & si\ z \neq 0 \\ Max(\mu_{A_1}(0), \mu_{A_2}(0) & si\ z = 0 \end{cases} \qquad (3.8)$$

División Extendida: La división de dos cantidades difusas se define mediante:

$$\mu(A_1 \div A_2)(y) = Sup\{Min(\mu_{A1}(y.z), \mu_{A2}(y)) / y \in \Omega\} \qquad (3.9)$$

Basándose en una expresión particular del principio de incertidumbre adaptada al empleo de α-cortes y en un tipo de número difuso similar al descrito anteriormente, denominado número L-R, en [DuPr80] se describen fórmulas de cálculo rápido para las anteriores operaciones aritméticas.

3.1.6 Teoría de la Posibilidad

Esta teoría se basa en la idea de variables lingüísticas y cómo éstas están relacionadas con los conjuntos difusos [Zade78]. Así, se puede evaluar la *posibilidad* de que una determinada variable *X* sea o pertenezca a un determinado conjunto *A*, como el grado de pertenencia de los elementos de *X* en *A*.

Definición 3.18 *Sea un conjunto difuso A definido sobre Ω con su función de pertenencia $\mu_A(x)$ y una variable X sobre Ω (que desconocemos su valor). Entonces, la proposición "X es A" define una* ***Distribución de Posibilidad****, de forma que se dice que la "posibilidad" de que X = u vale $\mu_A(u)$, para todo valor u ∈ Ω.*

3.2 Bases de Datos Difusas

En esta sección vamos a realizar una introducción a las *Bases de Datos Difusas*, centrándonos en el modelo GEFRED que es el utilizado en este proyecto.

En primer lugar, realizaremos una introducción genérica a las bases de datos y, más concretamente, a las *bases de datos relacionales*. En el siguiente apartado veremos las distintas representaciones de la imprecisión en bases de datos, distinguiendo aquellos que no consideran la lógica difusa y los que sí lo hacen. En ese mismo apartado, inspeccionaremos brevemente los modelos de Bases de Datos Relaciones Difusas (BDRD) y, a continuación, mostraremos los principales aspectos de modelo de BDRD GEFRED, mostrando algunos detalles del Álgebra Relacional Difusa y del Cálculo Relacional Difuso.

3.2.1 Modelo Relacional de Bases de Datos

En esta sección intentaremos dar una visión general de lo que son las bases de datos en general y el *Modelo Relacional* en particular. No pretendemos dar una explicación exhaustiva de su definición ni de sus características pues ya existen multitud de libros sobre el tema con distintos enfoques.

Entenderemos por **Base de Datos** un repositorio o conjunto de datos almacenados, normalmente en dispositivos electrónicos de un ordenador, y que son gestionados (para lectura y/o escritura) por un programa llamado *Sistema Gestor de Bases de Datos* (SGBD).

Los datos de una base de datos no deben tener redundancias (un mismo dato no debe estar almacenado en distintos lugares) y pueden ser *compartidos* por varios usuarios del SGBD.

El SGBD maneja las solicitudes de acceso a la base de datos por parte de los usuarios, ocultando a éstos detalles sobre el hardware donde los datos están almacenados. Además, el SGBD se encarga de tareas como la privacidad de los datos y la eficiencia en su acceso.

La gestión de bases de datos se empezó a hacer popular en los años 1970s y 1980s y hoy día su uso está ampliamente extendido, usándose en multitud de pequeñas y medianas empresas y no solamente en las grandes.

En todas las bases de datos se almacena información sobre determinadas entidades u objetos y sobre las relaciones existentes entre algunas de estas entidades. Los modelos de bases de datos que más se generalizaron fueron:

1. **Modelo jerárquico**: Los datos se representan por una estructura en árbol, donde los datos inferiores dependen o están incluidos en los datos superiores. Su limitación radica en el hecho de que sólo permite representar directamente relaciones de uno a muchos.

2. **Modelo en red**: Los datos se pueden ver como en el modelo jerárquico, pero sin limitación en cuanto a su organización, de forma que los datos inferiores pueden estar también relacionados con varios de los superiores. Así, permite modelar mejor una correspondencia de muchos a muchos.

3. **Modelo relacional**: En este modelo todos los datos son vistos en forma de tabla o relación. Es el más extendido por su facilidad de comprensión y lo veremos con profundidad a continuación.

El **modelo relacional** de bases de datos fue propuesto por Dr. E.F. Codd de IBM en 1970, en un intento de simplificar la estructura de la base de datos, que llegaba a ser demasiado compleja en otros modelos. Pronto, el modelo relacional se popularizó, aunque muchos de los sistemas que se llamaban relacionales no lo eran realmente. Por eso, Codd publicó una lista de 12 reglas que deberían cumplir todos los SGBD Relacionales.

Definición 3.19: *Llamaremos **Base de Datos Relacional** (BDR) a aquella base de datos donde todos los datos visibles al usuario están organizados estrictamente como tablas (o relaciones) de datos y donde todas las operaciones de la base de datos trabajan sobre estas tablas y/o producen nuevas tablas. El SGBD Relacional (SGBDR) es un SGBD que trabaja sobre este tipo de bases de datos*.

□

Esta definición, junto con las características básicas de toda base de datos, nos lleva a tener que definir los siguientes apartados:

1. *Estructuras de datos empleadas*: Las tablas o relaciones.
2. *Integridad de los datos*: Claves primarias, candidatas y externas.
3. *Definición de los datos*: Lenguajes (*DDL*) básicamente para la creación, borrado y alteración de la estructura de las tablas y objetos de la base de datos relacional.
4. *Manipulación de los datos*: Lenguajes (*DML*) básicamente para la consulta, modificación e inserción de datos.

3.2.1.1 Estructuras de Datos: Tablas o Relaciones

La estructura de los datos viene caracterizada por el concepto de tabla o relación. Para precisar dicho concepto es necesaria la definición previa de algunos conceptos:

Definición 3.20: *Llamaremos **Atributo**, **Campo** o **Columna** (Attribute, Column) a cada una de las características importantes y variables que tiene cualquier tipo de entidad u objeto y que son almacenadas en una base de datos, para reconocer a dicha entidad u objeto. No nos importará que dicha entidad tenga otros atributos, pero sólo trataremos aquellos que realmente nos interesan y que, por tanto, deseamos almacenar en nuestra base de datos*.

□

Definición 3.21: *Llamaremos **Dominio** (Domain) al conjunto de todos los valores posibles que puede tomar un atributo. Por tanto, todos los atributos tendrán un dominio asociado*.

□

Una característica inicial que se le exigía a los valores del dominio era la *atomicidad*, en el sentido que no exista una descomposición de los mismos que aporte significado. No obstante esta premisa se ha relajado con la evolución del modelo, y actualmente se encuentran en la literatura muchas matizaciones de la misma. Un dominio puede llevar

asociado un conjunto de operadores específicos del mismo. Por ejemplo, el dominio de los números enteros tiene operadores como la suma y la resta.

Definición 3.22: *Llamaremos **Tupla** al conjunto de todos los valores concretos de todos los atributos de una determinada entidad u objeto.*

□

Definición 3.23: *Llamaremos **Valor de Dominio** a un valor concreto de una tupla. O sea, es el valor para un atributo concreto de una entidad concreta.*

□

Con estos conceptos podemos definir lo que es una relación:

Definición 3.24: *Llamaremos **Relación** o **Tabla** (Relation, Table) al conjunto de dos partes, cabecera y cuerpo*:

- *La* cabecera *consiste en un conjunto fijo de n pares atributo-dominio,*

$$\{(A_1 : D_1), (A_2 : D_2), ..., (A_n : D_n)\}$$

donde cada atributo A_j *se corresponde exactamente con el dominio subyacente* D_j *con j = 1, 2, ..., n, siendo estos dominios no necesariamente distintos.*

- *El* cuerpo *consta de un conjunto de tuplas, donde cada tupla consiste en un conjunto de n pares atributo-valor,*

$$\{(A_1 : v_{i1}), (A_2 : v_{i2}), ..., (A_n : v_{in})\}$$

con i = 1, 2, ..., m, siendo m el número de tuplas que contiene el conjunto. Para cada atributo A_j *existe un par atributo-valor* $(A_j : v_{ij})$ *donde* v_{ij} *es el Valor de Dominio del atributo* A_j *perteneciente al dominio* D_j *de la entidad o tupla i-ésima.*

*Los valores m y n se denominan **cardinalidad** y **grado** de la relación, respectivamente. Mientras que el grado se mantiene constante en el tiempo para una relación, la cardinalidad puede variar en una misma relación con el tiempo.*

Cada relación tiene un nombre que identifica a todo su contenido y que será usado para expresar las operaciones que se realicen sobre la relación.

□

Otras definiciones útiles para expresar las restricciones de integridad son:

Definición 3.25: *Llamaremos **Clave** o **Llave Primaria** (Primary Key) al conjunto de atributos que se usarán para identificar unívocamente cada tupla de una relación. Este conjunto debe ser mínimo, es decir, que no exista un subconjunto de él que pueda usarse con el mismo propósito. Pueden existir varios de estos conjuntos para una relación dada, pero solamente se seleccionará uno de estos como Clave Primaria, quedando los restantes como Claves Alternativas o Claves Candidatas.*

□

Definición 3.26: *Llamaremos **Clave** o **Llave Externa** (Foreign Key) al conjunto de atributos de una relación que son Clave primaria o candidata en otra relación distinta y que pueden usarse para enlazar o relacionar los datos de ambas relaciones.*

□

De su definición se obtienen las siguientes propiedades para las relaciones:

1. *No hay tuplas duplicadas*: esto implica que siempre deberá existir una clave primaria. La necesidad de que exista una clave primaria para cada relación estriba en que es la única forma de acceder de forma unívoca a cada tupla de la misma. A veces, la clave primaria puede ser el conjunto de todos los atributos que forman la relación.
2. *Los atributos no están ordenados*. Esta propiedad proviene del hecho de que la *cabecera* de una relación es un conjunto matemático.

3. *Las tuplas no se encuentran ordenadas*. También esta propiedad se apoya en que la definición del *cuerpo* de una relación se corresponde con un conjunto matemático.
4. *Los valores de los atributos son atómicos*, en el sentido en el que apuntábamos cuando introducíamos el concepto de dominio. Una relación que cumpla esta propiedad se dice que esta *normalizada*. Aunque a primera vista esta propiedad pueda suponer una fuerte restricción a la representación de imprecisión, se pueden hacer lecturas de la misma que posibiliten la representación de este tipo de información sin ocasionar una pérdida de validez del modelo.

3.2.1.2 Integridad de los Datos

Una base de datos consiste en una configuración de datos que se supone que representan una porción del mundo real. Ningún modelo de base de datos puede garantizar que esa representación se corresponda con la realidad en todo momento, esto supondrá, entre otras cosas, que la base de datos tendrá que poseer un conocimiento, no solo sobre los datos, sino también sobre su significado. Lo que sí puede y debe hacer un SGBD es impedir que se introduzca información que no pueda ser identificada, ni que se haga referencia en un lugar de la base de datos a información que no exista en la misma. Estos son los enunciados informales de las reglas de *Identidad* y *Referencial* respectivamente. En forma más precisa pueden ser formulados como sigue:

- **Regla de Identidad**. Ningún componente de la clave primaria de una relación base puede aceptar un valor *nulo*. Donde *nulo* significa que la información no se encuentra por alguna razón (como por ejemplo que la propiedad no sea aplicable o que el valor sea desconocido).

- **Regla de Integridad Referencial**. La base de datos no puede contener valores para la clave externa que no hallen correspondencia con los adoptados por la clave primaria de la relación a la que hacen referencia.

3.2.1.3 Definición de los datos

Los lenguajes de definición de datos o *DDL* (*Data Definition Language*) permiten la creación, modificación y eliminación de objetos de la base de datos relacional. Estos objetos pueden ser de diversos tipos según el SGBD y están desde los básicos como los son tablas, vistas o índices, hasta los más sofisticados como son *snapshots*, sinónimos, procedimientos y funciones almacenadas, paquetes de funciones y procedimientos, *clusters*, secuencias, roles, *triggers*...

Para esto existe un lenguaje ampliamente extendido en la actualidad: el lenguaje *SQL* (en su parte DDL), que puede ser consultado en [ElNa00].

3.2.1.4 Manipulación de los Datos: Álgebra y Cálculo Relacional

Los lenguajes de manipulación de datos o DML (*Data Manipulation Language*) permiten la consulta, modificación e inserción de datos. Para esto, el lenguaje SQL (en su parte DML), del que hablaremos más adelante, es también muy utilizado [ElNa00].

Las consultas forman la parte más importante de la manipulación de datos, pues en una base de datos hay información almacenada en distintas tablas y su objetivo es poder consultar toda esta información y la relación que exista entre ella. Para tal fin puede ser útil emplear varias relaciones en una misma consulta.

Codd diseñó dos niveles de lenguajes formales para la Manipulación de Datos: el *Álgebra Relacional* y el *Cálculo Relacional*. Estos dos lenguajes son la base de cualquier otro lenguaje y son explicados a continuación.

El **Álgebra** proporciona un conjunto de operadores mediante los cuales especificar la operación a realizar sobre las tablas, mientras que el **Cálculo** proporciona una sintaxis con la que expresar aquello que se desea obtener de las relaciones sin tener que especificar el mecanismo para obtenerlo. O sea, en el Álgebra hay que indicar *cómo* se recuperan los datos de nuestra consulta a través de sus operadores y en el Cálculo hay que indicar *qué* datos queremos recuperar, sin especificar cómo hacerlo.

Definiciones del Álgebra y del Cálculo Relacional, en sus dos versiones *Cálculo de tuplas* y *Cálculo de dominios*, pueden encontrarse en la bibliografía numerosas veces. En [Ullm82] Ullman expone como pasar una expresión en álgebra a otra en cálculo de tuplas, cómo pasar una expresión en cálculo de tuplas a otra en cálculo de dominios y cómo pasar una expresión en cálculo de dominios a otra en álgebra, demostrando la equivalencia total de estos lenguajes.

3.2.2 Representación de la Imprecisión en Bases de Datos

Existen varios modelos para representar la imprecisión en bases de datos. Cada uno tiene sus ventajas, inconvenientes y limitaciones respecto a los otros.

Además, el término "imprecisión" engloba varios significados que puede ser interesante distinguir: que la información que tenemos es incompleta, que no sabemos si es o no cierta (incertidumbre), que desconocemos totalmente esa información (*unknown*) o que dicha información no es aplicable a una determinada entidad (*undefined*). A veces, estos significados no son disyuntivos, sino que pueden unirse en una determinada información.

El objetivo global va más allá de simplemente representar la información imprecisa, sino que se requiere modificar también las operaciones que se efectúan sobre las relaciones (consultas difusas o flexibles, condiciones, inserción y modificación de datos difusos).

También se incluye la consulta flexible o difusa en bases de datos clásicas (sin más imprecisión que los *NULL* del modelo relacional según el SGBD empleado).

Podemos distinguir entre *modelos sin lógica difusa* y *modelos de BDRD*. Los primeros no serán abordados en este documento, puesto que el servidor FSQL que hemos utilizado se basa en el modelo BDRD GEFRED, que utiliza la lógica difusa.

3.2.2.1 Modelo GEFRED (Medina-Pons-Vila)

Es una síntesis ecléctica de otros modelos. Se basa en la definición de *Dominio Difuso Generalizado* (*D*) y *Relación Difusa Generalizada* (*R*), que incluyen los dominios y las relaciones clásicas respectivamente.

Dominio Difuso Generalizado: Es un modelo posibilístico, por lo que en los dominios admite distribuciones de posibilidad, pero también incluye el caso en el que el dominio subyacente no sea numérico sino escalares de cualquier tipo. Por ejemplo, una distribución de posibilidad sobre un atributo difuso con dominio subyacente no ordenado como "Color del Pelo": {1/Castaño, 0.7/Pelirrojo, 0.4/Rubio}.

Además, incluye los valores *Unknown*, *Undefined* y *NULL* con el mismo sentido que en el modelo de Umano-Fukami.

Relación Difusa Generalizada: Es una relación cuyos atributos tienen un Dominio Difuso Generalizado. A cada atributo A_j es posible asociarle un atributo de compatibilidad C_j donde almacenar un grado, que expresa el valor con el que se ha satisfecho la operación realizada sobre él, y se obtiene como consecuencia de los procesos de manipulación de los datos de esa relación.

La Relación Difusa Generalizada se compone de:

- *Cabeza H*: Nombre de cada uno de los *n* atributos, sus dominios y sus atributos de compatibilidad (opcional).
- *Cuerpo B*: Incluye los valores de *m* tuplas: i = 1, 2, ..., *m*.

$$R = \begin{cases} H = \{(A_1 : D_1 [, C_1]), ..., (A_n : D_n [, C_n]\} \\ B = \{(A_1 : \tilde{d}_{i1} [, c_{i1}]), ..., (A_n : \tilde{d}_{in} [, c_{in}]\} \end{cases}$$

Comparador Difuso Generalizado: GEFRED define un tipo de comparador general basado en cualquier comparador clásico existente (=, >, <, ...), pero no concreta la definición de cada uno. El único requisito que establece es que el comparador difuso debe respetar los

resultados de los comparadores clásicos cuando se comparan distribuciones de posibilidad que expresan valores *crisp* (como 1/x con $x \in$ X).

La *Igualdad Difusa* o *Aproximadamente Igual* es el comparador difuso más importante y más empleado. Suelen usarse las siguientes medidas:

- de **Posibilidad**: $Poss(A,B) = \sup_{x \in X} \{\min(A(x), B(x))\}$;
  → Posibilidad de que A y B sean iguales

- de **Necesidad**: $Nec(A,B) = \inf_{x \in X} \{\max(A(x), 1 - B(x))\}$;
  → Necesidad de que B sea igual a A (no a la inversa)

GEFRED no propone otros comparadores difusos, pero podemos encontrar la definición de otros comparadores (*Mayor difuso*, *Mayor o Igual difuso*, *Mucho Mayor difuso*) en su versión de posibilidad y necesidad, así como la comparación de distribuciones de posibilidad sobre dominios subyacentes no ordenado en [GaMe98, GaMe00].

En cuanto a la *indistinguibilidad/redundancia*, es decir, respecto al hecho de que en una relación no pueden existir tuplas repetidas, si éstas se diferencian sólo en atributos difusos, ¿cómo discernir si son o no la misma tupla?

El *modelo Prade-Testemale* [PrTe84] propone que dos distribuciones de posibilidad A y B son aproximadamente iguales (indistinguibles) si

$$\sup_{x \in X} |A(x) - B(x)| \leq \varepsilon$$

donde ε es un umbral que depende del dominio X que se considere.

Dos tuplas serán consideradas iguales (redundantes) si son aproximadamente iguales en todos sus componentes.

Este criterio no cumple la propiedad transitiva por lo que no se pueden crear clases de equivalencia. Dados dos valores redundantes ¿qué valor almacenamos en la base de datos? Lo más intuitivo es almacenar la unión de ambos. También pueden considerarse dos valores como iguales si uno incluye al otro, de forma que se elimina el valor que está incluido en el otro. Pero, ¿qué hacemos si en una tupla se dan inclusiones opuestas?

El SGBD GEFRED, en cambio, es conservador, de manera que exige atributos *crisp* o difusos en la clave primaria con algún criterio de redundancia (como ser exactamente iguales).

3.3 Implementación de la BDRD y Lenguaje FSQL

En esta sección vamos a mostrar primeramente la arquitectura de la BDRD que vamos a utilizar en este proyecto. Tras esto, explicaremos los aspectos más importantes y significativos del lenguaje FSQL, que nos va a permitir acceder de forma difusa a la información almacenada en la Base de Datos Difusa.

3.3.1 Implementación de la BDRD: *FIRST*

Muchos científicos se han planteado la forma de efectuar consultas flexibles o difusas a bases de datos (a través del álgebra y cálculo relacional difuso, por ejemplo). Las ventajas más importantes son:

- *Condiciones Difusas*. Proveer más respuestas cuando las consultas clásicas no lo hacen por ser demasiado restrictivas.
- *Resultado Difuso*. Conseguir, para cada uno de los elementos recuperados, un grado que indique en qué medida cumple las condiciones establecidas. Esto nos permite, por ejemplo, ordenar por orden de importancia los elementos recuperados para elegir sólo los más apropiados.

Básicamente, el problema tiene dos vertientes:

- *Consultas Flexibles a Bases de Datos Difusas* (con imprecisión).
- *Consultas Flexibles a Bases de Datos Clásicas* (sin imprecisión).

Un sistema de consultas flexibles debería considerar esta segunda opción, ya que en la actualidad la inmensa mayoría de bases de datos existentes son clásicas (*crisp*) y puede ser útil implantar un sistema de consulta difusa, sin necesidad de convertir la base de datos a difusa.

Una consulta flexible es una consulta que tiene condiciones flexibles, es decir, para que una tupla aparezca en el resultado no es necesario cumplir estrictamente las condiciones impuestas en la consulta: los elementos del resultado pueden cumplir dichas condiciones de forma parcial. Hay muchos aspectos que pueden tenerse en cuenta:

- *Conceptos Imprecisos* (etiquetas lingüísticas). Por ejemplo, "dame los alumnos *jóvenes*". Este tipo de condiciones necesitan un umbral para indicar a partir de qué grado de cumplimiento los elementos serán recuperados. Pueden permitirse modificadores lingüísticos: "muy", "poco", "no muy", "más o menos": "Dame los alumnos *muy* jóvenes".
- *Conectivos Lógicos*, para unir condiciones simples: *NOT*, *AND*, *OR, ...* Suelen usarse los operadores típicos para la negación (1-*x*), conjunción (*t-norma del mínimo*) y disyunción (*s-norma del máximo*), pero pueden usarse otros [Yage91, DuPr95].

Utilizaremos, además, distintos comparadores difusos: *aproximadamente igual*, *mayor difuso*, *mayor o igual difuso*, *mucho mayor que*, etc.

En [Medi94] se expuso un módulo para permitir extender la capacidad de un SGBDR clásico para que pueda representar y manipular información imprecisa. Este módulo, llamado FIRST (*Fuzzy Interface for Relational SysTems*, Interfaz Difuso para Sistemas Relacionales), utiliza GEFRED como modelo teórico y los recursos del modelo relacional clásico para poder representar este tipo de información.

Además, para poder efectuar las operaciones típicas sobre la BDRD (creación de tablas, de etiquetas, consultas flexibles, etc.), se ha extendido el lenguaje SQL para que permita tratar los nuevos datos. Esta extensión se ha llamado FSQL (*Fuzzy SQL*) y será explicada en la sección 3.3.2.

3.3.1.1 Esquema General de FIRST

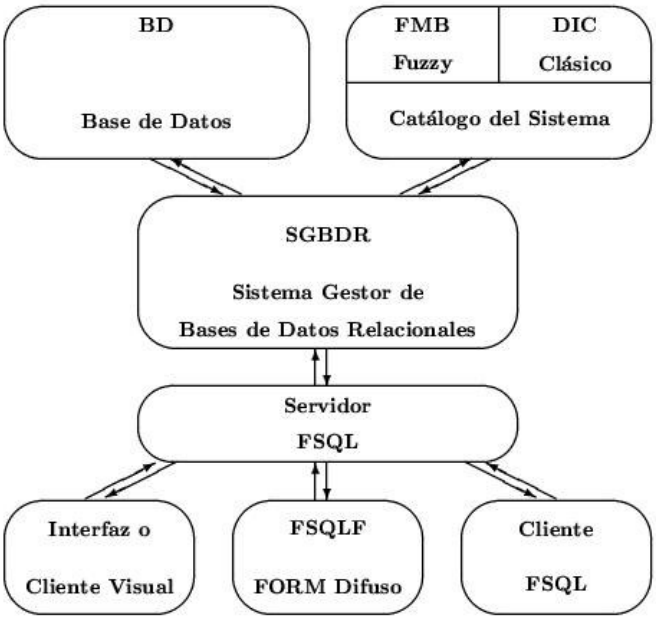


Figura 3.8: Esquema general de FIRST [Gali99]

La Figura 3.8 muestra el esquema general de *FIRST*. Como la idea de partida es la de construirlo sobre un gestor de bases de datos convencional, todos los desarrollos a realizar toman a dicho gestor como el elemento principal al que van dirigidas todas las peticiones. En breve, explicamos cada uno de esos módulos:

- **SGBDR** (Sistema Gestor de Bases de Datos Relacionales, *Relational DabaBase Management System*, RDBMS). Todas las operaciones que hayamos concebido para la extensión difusa que representa nuestra implementación, se traducirán a peticiones al SGBDR anfitrión (en general en SQL). Las peticiones al SGBDR se realizarán empleando el lenguaje SQL o FSQL, dependiendo si la consulta

involucra o no relaciones y/o condiciones difusas. Como veremos, las sentencias FSQL serán procesadas por el Servidor FSQL.

- **BD** (Base de Datos). Almacena, en formato relacional toda la información que sea de interés, igual que cualquier otra base de datos. La única diferencia es que nuestra base de datos permitirá el almacenamiento de información difusa en sus tablas. La forma en que se representan los datos en dichas tablas dependerá de la naturaleza de los mismos y se verá en los próximos apartados. En general, los atributos que representen conceptos difusos serán especificados en la creación de la tabla como *atributos difusos*. Existen tres tipos de atributos difusos: Tipo 1 (*crisp*), Tipo 2 (posibilístico) y Tipo 3 (escalares), pero sólo el primer tipo de atributos será utilizado en la aplicación desarrollada en este proyecto y será explicado más adelante.
- **FMB** (*Fuzzy Metaknowledge Base*, Base de Metaconocimiento Difuso). El "diccionario" o "catálogo del sistema" de un SGBDR representa aquella parte del sistema que almacena información sobre los datos recogidos en la base de datos, así como otro tipo de informaciones: usuarios, permisos, accesos, datos de control, etc. Dentro de este catálogo hemos incluido la FMB, que extiende esta parte del sistema a fin de que pueda recoger aquella información necesaria relacionada con la naturaleza "imprecisa" de la nueva colección de datos a procesar. Su organización y la información que almacena se verán más adelante.
- **Servidor FSQL**. Su objetivo es captar las sentencias en lenguaje FSQL y traducirlas a un lenguaje que entienda el SGBDR, el lenguaje SQL. Para efectuar esta traducción utilizará la información almacenada en la FMB. Este Servidor está íntegramente programado en el lenguaje PL/SQL de Oracle© y se encuentra implantado como una colección de módulos almacenados en el Servidor Oracle. El Servidor FSQL podrá ser instalado en cualquier plataforma donde exista una implementación de Oracle©.
- **Cliente FSQL**. Se trata de un programa que hace de interfaz entre el hombre (u otro programa) y el Servidor FSQL. Este programa puede ser muy simple, pues el trabajo principal de una operación FSQL será efectuado por el Servidor FSQL. El programa Cliente FSQL puede ser programado para cualquier Sistema Operativo y en cualquier lenguaje de programación. Por supuesto, es posible desarrollar Clientes FSQL visuales, que ayuden a editar sentencias difusas sin necesidad de

conocer la sintaxis de FSQL ni la de SQL. Uno de los objetivos de este proyecto es precisamente crear un cliente para la recuperación de imágenes, como se analiza en el Capítulo 4.

En cuanto a los atributos difusos, únicamente vamos a centrarnos en los del Tipo 1 (*crisp*) que serán los utilizados en este proyecto. Son atributos clásicos (*crisp*) pero sobre los que podremos efectuar consultas difusas, y que pueden tener definidas etiquetas lingüísticas sobre ellos. Permiten la utilización en las consultas de:

- *Comparadores difusos* (14 tipos de comparadores).
- *Constantes difusas*:
  - *UNKNOWN*, *UNDEFINED* y *NULL*, en el sentido de Umano-Fukami.
  - *$[a,b,c,d]*. Distribución de posibilidad trapezoidal. Véase Figura 3.9(a).
  - *$label*. Etiqueta lingüística definida como un trapecio en la FMB.
  - *[n,m]*. Intervalo "entre *a* y *b*" (*a=b=n* y *c=d=m*). Véase Figura 3.9(b).
  - *#n*. Aproximadamente *n* (*b=c=n* y *n-a=d-n=margen* definido en la FMB). Véase Figura 3.9(c).

La FMB almacena las etiquetas (con su definición asociada a cada una), el margen y la distancia mínima *M* para considerar dos valores como "muy separados" (usada en los comparadores del tipo "mucho mayor" y "mucho menor").

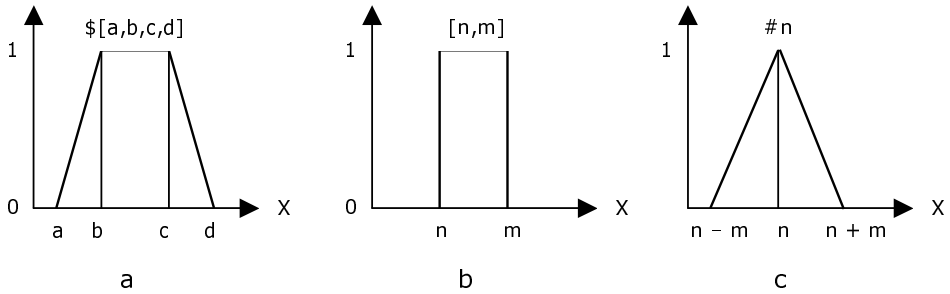


Figura 3.9: Tres tipos de constantes difusas. (a) Distribución Trapezoidal. (b) Intervalo "entre n y m". (c) "Aproximadamente n"

3.3.1.2 Tablas de la FMB

La FMB almacena información sobre los datos difusos en forma relacional. **OBJ#** y **COL#** son dos atributos que almacenan sendos números que identifican una columna concreta dentro de una tabla concreta. Todas las tablas de la FMB tienen esos dos atributos como clave primaria (o como parte de ella). *Oracle*© identifica cada tabla del sistema con un número OBJ#, que se puede obtener consultando la tabla *USER_OBJECTS*. Dentro de una tabla se identifica cada columna con un número COL# que puede consultarse en *USER_TAB_COLUMNS* (ver Figura 3.10).

Vamos a mostrar las seis tablas más importantes de la FMB:

- **FUZZY_COL_LIST (FCL)**. Los atributos difusos son los siguientes:
  - OBJ#, COL#. Atributo difuso.
  - F_TYPE. Tipo de atributo difuso (1, 2 ó 3).
  - LEN. Longitud máxima de una distribución de posibilidad en atributos. Tipo 3 (máximo 10).
  - COM. Comentario (usualmente se pone el nombre del atributo).
- **FUZZY_OBJECT_LIST (FOL)**. Descripción de los objetos difusos definidos para cada atributo (OBJ#, COL#). Los atributos difusos son los siguientes:
  - FUZZY_ID. Identificador del objeto difuso (clave externa a FLD y FND).
  - FUZZY_NAME. Nombre del objeto (sin espacios).
  - FUZZY_TYPE. Tipo del objeto: etiquetas trapezoidales (0), etiquetas de tipo 3 (1), cualificadores (2), cuantificadores relativos y absolutos (3,4).
- **FUZZY_LABEL_DEF (FLD)**. Definición de etiquetas trapezoidales. Los atributos difusos son los siguientes:
  - FUZZY_ID. Identificador del objeto difuso.
  - ALFA, BETA, GAMMA, DELTA. Los cuatro valores del trapecio.

- **FUZZY_APPROX_MUCH (FAM)**. Valores para el margen (MARGEN) y el valor mínimo o distancia mínima M para considerar dos valores muy separados (MUCH) para atributos difusos Tipo 1 ó 2. El valor del margen será empleado en este proyecto con cierta importancia y podrá ser modificado desde el propio Cliente FSQL.
- **FUZZY_NEARNESS_DEF (FND)**. Medidas de proximidad, semejanza o similitud (DEGREE) entre cada dos etiquetas de un atributo difuso Tipo 3 (FUZZY_ID1, FUZZY_ID2).
- **FUZZY_COMPATIBLE_COL (FCC)**. Indica las parejas de atributos difusos Tipo 3 que son compatibles entre sí: con el mismo dominio. Esto no sólo evita tener que redefinir las etiquetas y la relación de similitud sino que además permite comparar dos atributos difusos compatibles, ya que el sistema sabe que tienen el mismo dominio.

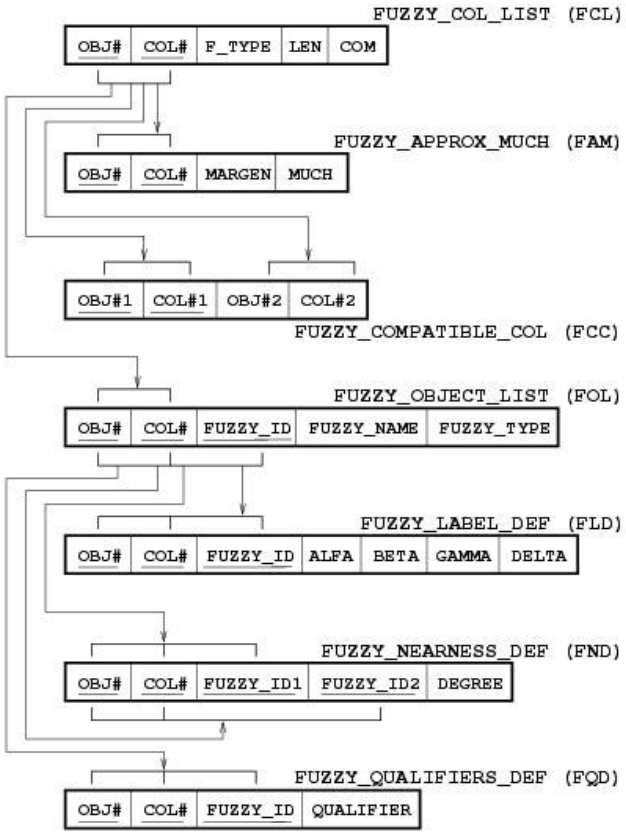


Figura 3.10: Tablas de la FMB (Esquema Gráfico)

3.3.2 Lenguaje para Consultas Difusas: *Fuzzy SQL*

Primeramente haremos un repaso somero al lenguaje SQL. Tras esto, mostraremos las características fundamentales del lenguaje FSQL.

3.3.2.1 Lenguaje SQL

SQL (*Structured Query Language*) es una evolución del lenguaje SEQUEL de D. Chamberlain y R.F. Boyce (1974) y fue implementado por primera vez por IBM en su base de datos relacional llamada SYSTEM R.

ISO (*International Standards Organization*) y ANSI (*AmericanNational Standars Institute*) desarrollaron una versión estándar en 1986, llamada SQL-86 o SQL1. Posteriormente, se desarrolló SQL2 o SQL-92. Actualmente se desarrolla SQL3, que incluye conceptos de bases de datos orientadas a objetos.

SQL es un lenguaje estándar para gestión de bases de datos relacionales. Está incluido en muchos SGBD (DBMS), como DB2 (de IBM), Oracle, Ingres, Informix, SyBase, Access, SQL Server, etc. Las mismas sentencias sirven en distintos SGBD. Si se usan sólo las características estándares facilita la tarea de migrar de SGBD, sin embargo, hay funciones no estándar en algunos de ellos.

SQL es fácil de aprender y utilizar, ya que tiene similitudes con el Álgebra Relacional, aunque se parece más al Cálculo y es más fácil e intuitivo que ambos lenguajes formales.

3.3.2.1.1 Comando SELECT

El comando más importante es, quizás, el de consulta, ya que nos debe permitir expresar una gran cantidad de tipos de consultas, que pueden involucrar multitud de expresiones y objetos diferentes (tablas, vistas, comparadores, subconsultas, modos de ordenación y agrupamiento...). Este comando tiene el siguiente formato:

- SELECT <Select-list>

  Expresiones a recuperar (atributos, funciones, operadores).

- FROM <Table-list>

  Tablas necesarias para la consulta (incluyen tablas de reunión, subconsultas, etc.).

- [WHERE <condición>]

  Condición de selección de tuplas, incluyendo condiciones de reunión.

- [GROUP BY <atributos_para_agrupar>]

  Atributos por los que agrupar el resultado (cada grupo tendrá los mismos valores en estos atributos). Las funciones de grupo o de agregación se aplicarán sobre cada uno de estos grupos. Se aplicarán sobre todas las tuplas si no existe esta cláusula.

- [HAVING <condición_de_grupo>]

  Condición sobre los grupos (no sobre las tuplas).

- [ORDER BY <atributos_para_ordenar>]

  Atributos por los que ordenar. El orden de estos datos influye en la forma de presentar los resultados finales. Puede ponerse el nombre de los atributos por los que se quiere ordenar o un número indicando la posición del atributo en la <Select-list>.

3.3.2.1.2 Comandos INSERT, DELETE y UPDATE

El comando *INSERT* sirve para introducir nuevos datos en la base de datos y tiene el siguiente formato general:

- INSERT INTO <tabla> [(<lista-atributos>)] VALUES (<lista-expresiones>)

  - <tabla>. Es la tabla (vista o subconsulta) donde se desea insertar una tupla.
  - <lista-atributos>. Es una lista de atributos entre paréntesis y separados por comas que indican el orden de las expresiones siguientes. Esta lista es opcional y, si no aparece, se

toma la lista de atributos de <tabla> en el orden en el que se usaron en su creación.

- <lista-expresiones>. Es una lista de expresiones entre paréntesis y separados por comas que expresan los valores que deseamos insertar en los correspondientes atributos. Esta lista y la palabra VALUES pueden sustituirse por una subconsulta que contendrá los datos que serán insertados. Usando una subconsulta puede insertarse varias tuplas a la vez.

El comando DELETE sirve para borrar datos (tuplas) en la base de datos y tiene el siguiente formato general:

- DELETE <tabla> WHERE <condición>

Que borra las tuplas de <tabla> que cumplen la <condición> que se indica.

El comando UPDATE sirve para actualizar valores de datos antiguos en la base de datos y tiene el siguiente formato general:

- UPDATE <tabla> SET <asignación> [WHERE <condición>]

  - <tabla>. Es la tabla (vista o subconsulta) donde se desean actualizar datos.
  - <asignación>. Es una asignación con los siguientes dos formatos posibles para actualización de un único atributo o varios de ellos respectivamente:

    <atributo> = <expr> que expresa que se actualizan todos los atributos a la expresión que se indica.

    <lista de atributos> = <subconsulta> que expresa que se actualizan todos los atributos de la lista con los

datos de la subconsulta (que debe ser con igual número y tipo de atributos).

3.3.2.2 Lenguaje FSQL

Vamos a analizar en este apartado únicamente la sentencia `SELECT`, sabiendo que el resto de sentencias DML, aun siendo distintas, no suponen cambios importantes en su sintaxis y semántica. También mostraremos la sentencia `CREATE TABLE` como único representante DDL.

La sentencia `SELECT` es una sentencia tan potente como compleja y flexible, muy fácil de usar en consultas simples y no tan fácil de usar en consultas complejas debido a su potencia y versatilidad. Esta sentencia es tan potente que rara vez se llega a utilizar todo su poder expresivo para realizar una consulta, pues lo normal es efectuar consultas mucho más simples de lo que `SELECT` permite.

A veces, para simplificar la escritura y el entendimiento de consultas complicadas se usan vistas intermedias que son creadas como subconsultas a la base de datos.

El lenguaje FSQL es una auténtica extensión de SQL. Esto significa que todas las sentencias válidas en SQL lo son también en FSQL. Además, FSQL incorpora algunas novedades para permitir el tratamiento de información imprecisa. Básicamente, las extensiones efectuadas a esta sentencia son las siguientes:

- **Etiquetas Lingüísticas**: Si un atributo es susceptible de tratamiento difuso entonces pueden definirse etiquetas sobre él. Estas etiquetas son precedidas, por convenio, por el símbolo $ para distinguirlas fácilmente de otros identificadores.

  Hay 2 tipos de etiquetas que serán usadas en diferentes tipos de atributos difusos.

  1. Etiquetas para atributos con un *dominio subyacente ordenado*. Cada etiqueta de este tipo tiene asociada, en la FMB, una distribución de

posibilidad trapezoidal como la de la Figura 3.9(a). Así, por ejemplo, podemos definir las etiquetas *$Muy_Bajo*, *$Bajo*, *$Normal*, *$Alto* y *$Muy_Alto*, sobre el atributo *Altura* de una persona. Por ejemplo, el atributo *$Alto* puede ser definido como una distribución de posibilidad con los siguientes valores (en centímetros) α=185, β=195,γ=200 y δ=210.

2. Etiquetas para atributos con un *dominio subyacente no ordenado*. Aquí, puede haber una relación de similitud definida entre cada dos etiquetas del dominio y almacenada en la FMB. El grado de similitud entre cada dos etiquetas será un valor del intervalo [0,1]. Por ejemplo, para un atributo que almacenara el color del pelo de una persona podríamos definir las etiquetas *$Rubio* y *$Pelirrojo* e indicar (en la relación de similitud) que ambos valores son similares en grado 0.6.

- **Comparadores Difusos**. Además de los comparadores clásicos típicos (=, >, ...), SQL incluye los comparadores difusos de la Tabla 3.1. Como en SQL, los comparadores difusos pueden comparar una columna de una tabla con una constante o dos columnas del mismo tipo o de tipos compatibles.

	Comp. Difuso	**Significado**
Posibilidad	FEQ	Fuzzy EQual: Posiblemente Igual (sobre escalares o números difusos)
	FGT	Fuzzy Greater Than: Posiblemente Mayor que
	FGEQ	Fuzzy Greater or EQual: Posiblemente Mayor o Igual que
	FLT	Fuzzy Less Than: Posiblemente Menor que
	FLEQ	Fuzzy Less or EQual: Posiblemente Menor o Igual que
	MGT	Much Greater Than: Posiblemente Mucho Mayor que
	MLT	Much Less Than: Posiblemente Mucho Menor que
Necesidad	NFEQ	Necessarily Fuzzy EQual: Necesariamente Igual que (o incluido en)
	NFGT	Necessarily Fuzzy Greater Than: Necesariamente Mayor que
	NFGEQ	Necessarily Fuzzy Greater or EQual: Necesariamente Mayor o Igual que
	NFLT	Necessarily Fuzzy Less Than: Necesariamente Menor que
	NFLEQ	Necessarily Fuzzy Less or EQual: Necesariamente Menor o Igual que
	NMGT	Necessarily Much Greater Than: Necesariamente Mucho Mayor que
	NMLT	Necessarily Much Less Than: Necesariamente Mucho Menor que

Tabla 3.1: Comparadores Difusos de FSQL [Gali99]

Los comparadores de posibilidad son más generales (menos restrictivos) que los de necesidad. Por tanto, los comparadores de necesidad recuperan menos tuplas y estas tuplas cumplirán necesariamente con las condiciones impuestas en la consulta.

En atributos con dominio subyacente no ordenado (difusos Tipo 3) sólo puede usarse el comparador difuso `FEQ`, puesto que carecen de orden. El comparador difuso de "desigualdad" o "posiblemente distinto" no se ha considerado porque puede modelarse negando una comparación con `FEQ`:

```
NOT <Atributo Difuso> FEQ <Atributo o Cte>
```

En este proyecto, únicamente vamos a utilizar atributos difusos del Tipo 1 y solamente vamos a comparar estos atributos con un valor *crisp*, por las propias necesidades de nuestra aplicación. Un valor *crisp* puede ser otro atributo difuso del Tipo 1, un atributo no difuso o una constante no difusa o tipo intervalo.

Atrib_T1 FCOMP *crisp*

Este tipo de comparaciones puede efectuarse utilizando comparadores clásicos. Si se emplean comparadores difusos, el lenguaje FSQL define que el valor *crisp* de la derecha será difuminado empleando el valor del **margen** (*mg*) del atributo difuso Tipo 1 que haya a la izquierda:

- Si es un valor *crisp n*, se convertirá en un valor aproximado #*n*.
- Si es un intervalo [*n*,*m*], se convertirá en el correspondiente trapecio $[*n* - *mg*, *n*, *m*, *m* + *mg*].

De esto se deduce que, por ejemplo, Atrib_T1 FEQ 6 y Atrib_T1 FEQ #6 son equivalentes.

Con los comparadores difusos MGT/NMGT y MLT/NMLT se difumina también el valor *crisp* del atributo de la izquierda. Esto consigue una comparación

más gradual, lo cual parece lógico debido a que estos comparadores son difusos "per se" (no tienen su correspondiente comparador clásico).

- **Umbral de Cumplimiento** (*threshold*, τ): Para cada condición simple se puede establecer un umbral de cumplimiento (por defecto es 1) con el formato siguiente:

  `<Condición simple> THOLD` τ

  indicando que la condición debe ser satisfecha con un grado mínimo de $\tau \in [0,1]$ para que la tupla en cuestión aparezca en la relación resultante. La palabra reservada *THOLD* es opcional y puede ser sustituida por cualquier comparador crisp tradicional (=, ≤,...), modificando así, lógicamente, el significado de la consulta. La palabra THOLD es equivalente a usar el comparador *crisp* ≥.

- **Función** `CDEG(<atributo>)`. Una llamada a la función `CDEG` (*Compatibility DEGree*) puede ser colocada en la lista de selección (expresiones tras la palabra reservada `SELECT`).

  Su argumento es un atributo y muestra una columna con el grado de compatibilidad o cumplimiento de la condición de la consulta para el atributo que se indica. Si aparecen operadores lógicos (NOT, AND y/o OR), el cálculo de este grado de cumplimiento es efectuado usando las funciones que se indican en la Tabla 3.2, pero FSQL permite modificar las funciones a emplear. Para ello hay que modificar la vista *FSQL_OPTIONS*. En esta vista el usuario puede establecer las funciones a usar para cada uno de los 3 operadores lógicos. Obviamente, la función que se indique en esa vista debe estar implementada en el SGBD y el usuario debe tener acceso a la misma.

<Condición> (con operadores lógicos)	**CDEG(<Condición>)**
<cond1> AND <cond2>	min(CDEG(<cond1>),CDEG(<cond2>))
<cond1> OR <cond2>	max(CDEG(<cond1>),CDEG(<cond2>))
NOT <cond1>	1- CDEG(<cond1>)

Tabla 3.2: Operaciones usadas por defecto para el cálculo de la función CDEG de FSQL con operadores lógicos

Estas funciones pueden ser implementadas por un usuario particular para su uso personal. La función para el NOT tendrá un único argumento numérico, mientras que las funciones para el AND y para el OR tendrán ambas dos argumentos numéricos.

La precedencia de los operadores lógicos es la habitual, i.e., de mayor a menor precedencia están NOT, AND y OR.

Si el argumento de la función `CDEG` es un atributo, entonces la función `CDEG` sólo usa las condiciones que incluyen a ese atributo. Si el atributo indicado como argumento de `CDEG` no aparece en la condición, entonces, esta función no es aplicable a dicho atributo, pero en vez de dar un error se procede a devolver grado 1 para todas las tuplas. También podemos usar un asterisco como argumento como se explica a continuación.

- **CDEG(*)**. Si esta función tiene un asterisco como argumento, entonces calcula y muestra el grado de cumplimiento de la condición para cada tupla. Este cálculo es efectuado como se explicó en el punto anterior pero teniendo en cuenta todos los atributos empleados en la condición, y no sólo uno de ellos.

- **Carácter Comodín %**: El empleo de este carácter es similar al del utilísimo carácter comodín * de SQL, pero éste, además de incluir todas las columnas de las tablas indicadas en la parte `FROM` de la consulta, también incluye las columnas con el grado de cumplimiento de aquellos atributos relevantes. O sea, en el resultado también encontraremos columnas donde la función `CDEG` está aplicada a cada uno de los atributos que aparecen en la condición. Si un atributo difuso no aparece en la cláusula `WHERE`, su `CDEG` no es aplicable y por tanto no aparecerá su `CDEG` si se usa el comodín %.

- **Constantes Difusas**. En FSQL podemos usar diversos tipos de constantes difusas. Estos tipos son detallados en la Tabla 3.3.

Constante Difusa	Significado
UNKNOWN	Valor desconocido pero el atributo es aplicable.
UNDEFINED	Atributo no aplicable o sin sentido.
NULL	Ignorancia total: no sabemos nada sobre eso.
$[α,β,γ,δ]	Distribución de posibilidad trapezoidal (α≤β≤γ≤δ): ver Figura 3.9(a).
$label	Etiqueta Lingüística. Puede ser un trapecio o un escalar (definidos en FMB).
[*n*,*m*]	Intervalo "Entre *n* y *m*" (α=β=*n* y γ=δ=*m*).
#*n*	Valor difuso "Aproximadamente *n*" (β=γ=*n* y *n*-α=δ-*n*=margen en FMB)

Tabla 3.3: Constantes difusas que pueden ser usadas en FSQL

Como se ha dicho anteriormente las etiquetas están registradas en la FMB, al igual que el valor del margen para valores aproximados.

- **Condición con** `IS`. Otra clase de condición que podemos usar en FSQL tiene el siguiente formato:

```
                                        { UNKNOWN
<Atributo_Difuso> IS [NOT] {UNDEFINED
                                        {   NULL
```

Observaciones sobre la condición con `IS`:

- Este tipo de condición (sin NOT) será cierta si el valor del atributo difuso de la izquierda (`<Atributo Difuso>`) es la constante situada a la derecha.
- Si el atributo no es difuso y la constante de la derecha es NULL, esta constante será entendida de la forma que lo haga el SGBD (si este lo permite). En particular, Oracle© permite valores NULL como valor posible de los atributos, aunque esto puede ser evitado estableciendo la restricción *NOT NULL* en los comandos *CREATE TABLE* o *ALTER TABLE*.
- Si `FEQ` es usado en vez de *IS* el significado es distinto. Con `FEQ` se compara el grado de compatibilidad entre el atributo y la constante y no simplemente si el atributo es igual a la constante.

Existen varios comandos DDL en el lenguaje FSQL, pero únicamente vamos a mostrar el comando `CREATE TABLE` que se utilizará en el script de creación de la base de datos para la aplicación.

- **Tipos de Datos Difusos**: Se han añadido tres nuevos tipos de datos, uno para cada Tipo de atributo difuso. Por comodidad en el aprendizaje, cada nuevo tipo tiene dos nombres. Estos son los tres nuevos tipos y su formato:

  1. Atributos difusos **Tipo 1**: Se escribe con las palabras reservadas `CRISP` o `FTYPE1` seguido de dos números entre paréntesis y separados por una coma, que indican los valores que tendrá el nuevo atributo difuso en las columnas `MARGEN` y `MUCH` de la tabla `FUZZY_APPROX_MUCH`. A continuación se puede poner opcionalmente el tipo base del atributo, es decir, el dominio subyacente, sobre el que tomaran valores los valores *crisp* que tenga la relación. Por defecto se tomará `NUMBER` como tipo base. Pueden producirse errores semánticos, por ejemplo si se define el tipo base como un tipo difuso.

  2. Atributos difusos **Tipo 2**: Se escribe con las palabras reservadas `POSSIBILISTIC` o `FTYPE2` seguido de dos números entre paréntesis y separados por una coma, que indican los valores que tendrá el nuevo atributo difuso en la tabla `FUZZY_APPROX_MUCH`. Igual que en los Tipo 1, a continuación se puede poner opcionalmente el tipo base sobre el que tomaran valores las distribuciones de posibilidad que tenga la relación.

  3. Atributos difusos **Tipo 3**: Se escribe con las palabras reservadas `SCALAR` o `FTYPE3` seguido opcionalmente de un número entre paréntesis que indica el número de datos máximo para los valores de dicho atributo, es decir el número máximo de elementos en las distribuciones de posibilidad sobre escalares (valor del atributo `LEN` de la tabla `FUZZY_COL_LIST`). Suponemos que no hay más de 10 datos (y 1 como mínimo y por defecto). Opcionalmente se puede añadir a la

definición de este tipo de atributos la palabra `DOMAIN` seguida del nombre de un atributo (que puede ser de la misma tabla), para indicar que el atributo que estamos definiendo en la tabla que queremos crear será compatible con el atributo que se indica tras `DOMAIN` y podrá, por tanto, tomar sus valores sin tener que definirlos de nuevo para este atributo. La cláusula `DOMAIN` hace que se inserte una fila en la tabla `FUZZY_COMPATIBLE_COL`.

- **Subconsultas, constantes, condiciones y expresiones difusas**. Cuando estos elementos forman parte de una sentencia `CREATE TABLE` se permite utilizar sus variantes difusas. Por ejemplo, para especificar un valor constante por defecto, tras la palabra reservada `DEFAULT` puede situarse cualquier tipo de constante, sea difusa (Tabla 3.3) o no lo sea.

- **Restricciones de columna**. A una columna difusa le podemos imponer varias restricciones especiales (además de las ya existentes para atributos clásicos) como `NOT NULL`, `NOT UNDEFINED` y `NOT UNKNOWN` (que prohíben los valores correspondientes), `NOT LABEL`, `NOT CRISP`, `NOT TRAPEZOID`, `NOT INTERVAL` y `NOT_APPROX` (que prohíben valores del tipo indicado respectivamente) y algunas restricciones más.

4. RECUPERACIÓN DE IMÁGENES USANDO ATRIBUTOS DIFUSOS

Este capítulo nos va a mostrar todos los detalles sobre cada una de las fases implicadas en el proceso de recuperación de imágenes utilizando consultas difusas, mediante una base de datos difusa. La Figura 4.1 muestra a grandes rasgos este proceso.

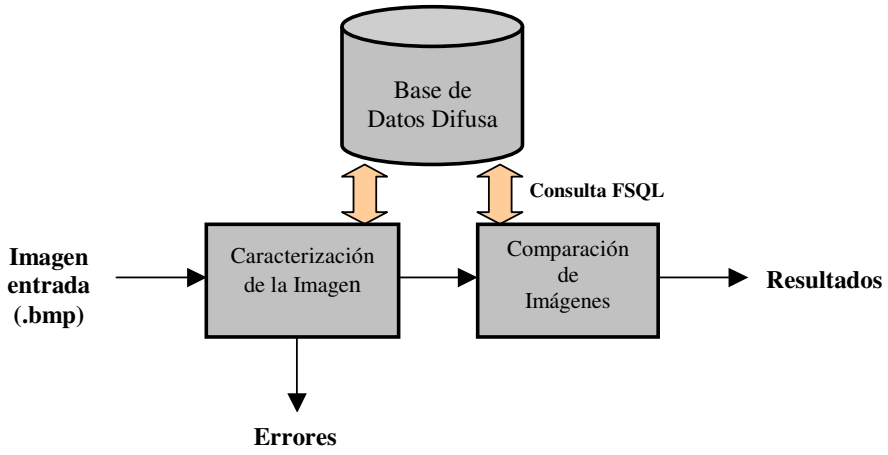


Figura 4.1: Esquema del Proceso de Recuperación de Imágenes

La *imagen de entrada*, que debe estar en formato *.bmp*, es el punto de inicio de nuestro proceso. Esta imagen sufre una serie de transformaciones hasta obtenerse una imagen del contorno de la figura o forma obtenida. A partir de ella obtenemos su función de curvatura y, a continuación, los puntos más representativos o puntos característicos de la forma, extraídos a partir de dicha función, que se añaden a la *Base de Datos*, concluyendo de esta manera la fase de *Caracterización de la Imagen*. Aquí acabaría el proceso en el caso de adición de imágenes a la base de datos.

En el proceso de búsqueda, los puntos obtenidos de la *imagen de entrada* pasan a ser utilizados en el proceso de *Comparación de Imágenes*, junto con las imágenes de la *Base de*

Datos Difusa que están implicadas en la búsqueda, realizándose el proceso de comparación (mediante consultas *FSQL*) y obteniéndose los resultados, que serán mostrados por pantalla.

Vamos a distinguir dos grandes apartados en este capítulo. En el primero ampliaremos el proceso de *Caracterización de la Imagen*, mientras que en el segundo analizaremos con detalle el proceso de *Comparación de Imágenes*.

4.1 Caracterización de la Imagen

Ampliamos la información mostrada en la Figura 4.1, centrándonos en la caja *Caracterización de la Imagen*, y analizando cada una de las cuatro subfases que la componen: *Preprocesamiento de la Imagen*, *Obtención del Contorno, Cálculo de la Función de Curvatura* y *Extracción de Puntos Característicos*, como muestra la Figura 4.2.

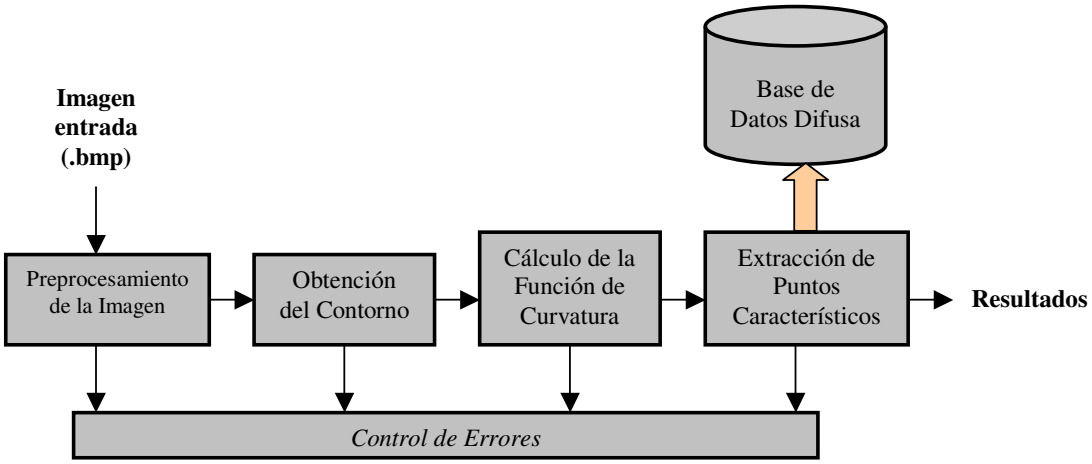


Figura 4.2: Esquema del Proceso de Caracterización de la Imagen

La primera fase de *Preprocesamiento de la Imagen* trata de transformar la imagen para aislar la figura del fondo y así, facilitar el proceso de *Obtención del Contorno*, aplicando determinados filtros y algoritmos. Una vez obtenido el contorno, tendremos una sucesión de puntos que comienza y acaba en el mismo punto, pues se supone un contorno cerrado. A partir de esta sucesión podremos realizar el *Cálculo de la Función de Curvatura*, cuyo

análisis exhaustivo nos llevará al proceso de *Extracción de Puntos Característicos*, mediante el cual caracterizamos la imagen de entrada.

Si procedemos a añadir una imagen a la base de datos, el proceso concluye insertando los datos de la imagen, con todos sus puntos característicos. En otro caso, si realizamos una búsqueda, la conclusión de la *Caracterización de la Imagen* es el comienzo de la fase de *Comparación de Imágenes*, sin que haya sido necesaria la intervención de la base de datos en esta primera fase.

En general, no es válida cualquier imagen en este proceso sino que hemos restringido el tipo de imagen permitido, con objeto de simplificar y acelerar partes del proyecto que no tienen una importancia crucial en cuanto al objetivo de este trabajo se refiere. A continuación, se detallan estas condiciones que aclaran el tipo de imagen válida y los aspectos que van a tenerse en cuenta en todas las imágenes:

- Únicamente se admiten *archivos de imagen en el formato .bmp* para las imágenes de entrada, con una configuración de *256 colores* y cuyos píxeles tengan las tres componentes RGB iguales, o lo que es lo mismo, *todos sus píxeles sean grises*.
- Únicamente serán consideradas aquellas imágenes que *posean una única forma (o figura)* en su seno. En el caso de imágenes con varias formas (tras el resultado del proceso de umbralización), puede ocurrir que se considere una de ellas como la forma válida y las otras queden consideradas como fondo. También puede ocurrir que, si algunas formas poseen algún punto de unión, las forma final sea una única figura que compone a varias. En el último caso, puede que no se considere a ninguna de ellas como forma válida (por tener un número muy pequeño de puntos en su contorno, en función del número total de puntos de objeto).
- *Solamente se utilizará el contorno como descriptor de la forma*, sin tener en cuenta otros descriptores como el área, el color o textura de la forma u otros aspectos.
- *No se tendrá en cuenta si la imagen posee huecos* en su interior, es decir, una parte de la imagen con color de fondo rodeada por puntos del color de la forma.
- Además de una única figura, el *fondo de la imagen debe ser homogéneo*, es decir, debe poseer un color claramente frecuente (superior al 50% del número de puntos de la imagen). En otro caso, la imagen será desechada.

- Solamente se considerarán válidas aquellas imágenes cuyo *número de puntos característicos* obtenidos tras la fase de *Caracterización de la Imagen* sea *superior a 2*. Aquellas que no superen esta cota inferior poseen muy poca información como para ser comparadas, por lo que serán desechadas.

Estos requisitos, aunque bastante restrictivos, nos permiten obviar el desarrollo de un algoritmo preciso de *segmentación*, puesto que no es el objetivo de este proyecto y podemos así, ceñirnos al proceso de caracterización y comparación, verdaderos pilares de este trabajo.

4.1.1 Preprocesamiento de la Imagen

La etapa de preprocesamiento consiste en una serie de transformaciones a partir de la imagen de entrada hasta obtener la imagen con la figura aislada del fondo, todo dispuesto para aplicar el algoritmo de obtención del contorno.

La Figura 4.3 explica las fases que, como si de un proceso industrial se tratara, permite transformar la imagen de entrada en otra más adecuada para la obtención del contorno de la forma.

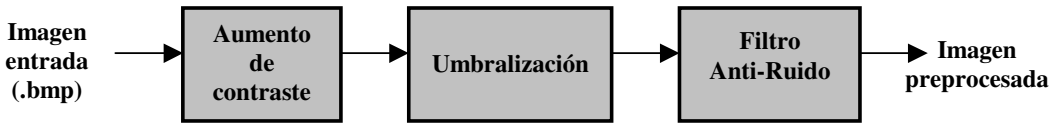


Figura 4.3: Esquema del Preprocesamiento de la Imagen

En primer lugar, aplicaremos la técnica de *Aumento de Contraste* para intentar aclarar la imagen si está muy oscura y viceversa, oscurecer la imagen si ésta se halla demasiado clara. En cualquier caso, el contraste de la imagen aumenta y ayuda a establecer posteriormente qué puntos de la imagen forman parte de la figura y cuáles del fondo.

Tras este proceso, a la imagen resultante se le aplica el proceso de *Umbralización*, mediante el cálculo previo del histograma, de manera que distinguimos entre *objeto* (o *forma*) y *fondo*.

A continuación, a la imagen umbralizada le aplicaremos un *Filtro Anti-Ruido*, a través del cual eliminaremos ruido de la imagen y corregiremos ciertas situaciones que pueden provocar fallos en el proceso de obtención del contorno si no se tratan en esta fase.

4.1.1.1 *Aumento de contraste*

En determinadas imágenes, la frontera entre objeto y fondo es muy imprecisa, ya que ambas partes poseen un color muy cercano y resulta difícil, incluso para el ojo humano, determinar qué es objeto y qué es fondo. Tal es el caso de la Figura 4.4(a), que muestra la imagen *2plat.bmp*. Se aprecia que es una imagen muy oscura y que puede resultar infructuoso el cálculo del contorno sobre ella si no somos capaces de aumentar el contraste de la imagen.

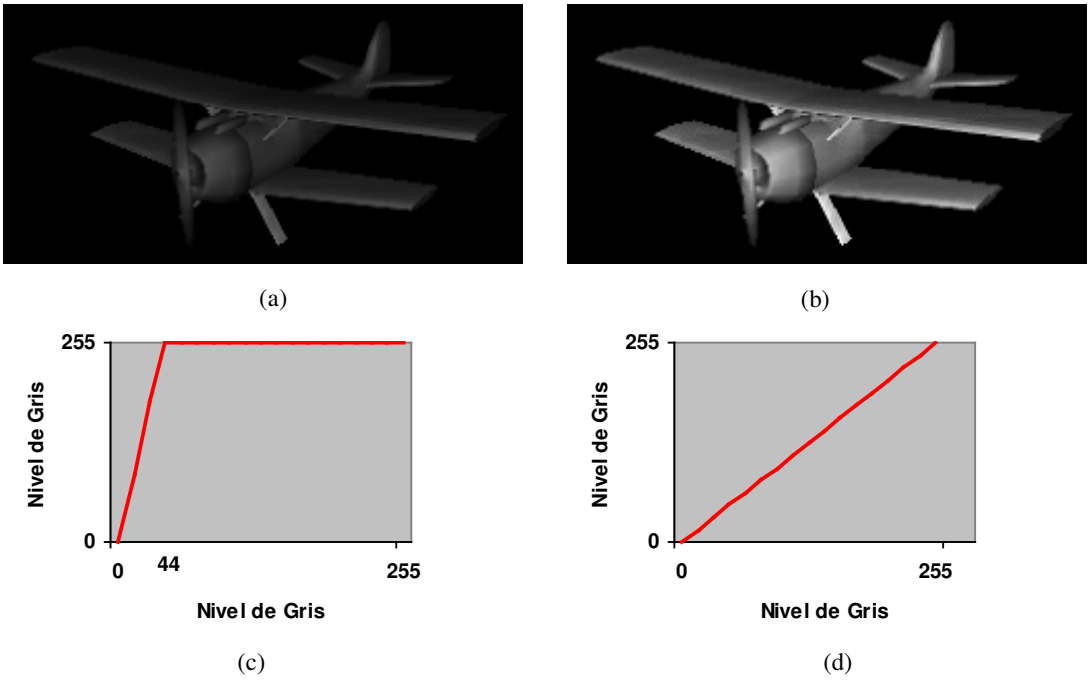


Figura 4.4: (a) Imagen *2plat.bmp*. (b) Imagen aumentada en contraste. (c) Transformación cuando *Min* = 0 y *Max* = 44. (d) Transformación cuando *Min* = 0 y *Max* = 255.

En primer lugar, debemos calcular el *histograma* de la imagen, es decir, un vector con las frecuencias de aparición de cada uno de los 256 niveles de gris posibles. Tras esto,

obtendremos el valor más oscuro y el valor más claro de la imagen, es decir, aquel valor más cercano a 0 y más cercano a 255, respectivamente, con frecuencia distinta de cero.

A partir de estos dos valores, generamos la siguiente función y la aplicamos sobre todos los puntos de la imagen, donde *Max* representa el color más claro y *Min* el más oscuro, y donde $c_{i,j}$ indica el color actual (antes de aplicar la función) de cada píxel de la imagen:

$$c_{i,j}^{nuevo} = \left[\frac{255}{Max - Min}\right] \cdot c_{i,j} - \left[\frac{255}{Max - Min}\right] \cdot Min \qquad (4.1)$$

Tal y como indica la Ecuación 4.1, el color nuevo calculado de cada píxel (i,j) se obtiene a partir de esta función. Podemos ver en la Figura 4.4(c) una representación gráfica de la función de transformación realizada (con *Min* = 0 y *Max* = 44) de la imagen de la Figura 4.4(a) donde se aprecia claramente una recta ($y = 5'8 \cdot x$) con pendiente mayor que 1, lo que implica que va a haber un aumento muy grande del contraste de la imagen. El resultado se muestra en la Figura 4.4(b), donde se aprecia la imagen mucho más clara y contrastada. Para completar este análisis, vamos a comentar tres casos extremos:

- Si *Max* = *Min*, entonces obtendríamos un denominador nulo. Sin embargo, cuando este proceso se encuentra con esta situación es porque únicamente hay un color en la imagen y, por tanto, la imagen no posee ningún objeto, ya que se considera que solamente hay fondo, por lo que la imagen se considera inválida.
- Si *Max* - *Min* = 1, entonces nos encontraríamos solamente con dos colores (*Min* y *Max*). Por tanto, la Ecuación 4.1 quedaría reducida a:

$$c_{i,j}^{nuevo} = 255 \cdot c_{i,j} - 255 \cdot Min = 255 \cdot (c_{i,j} - Min)$$

  - Si $c_{i,j} = Min$, entonces el nuevo color es 0.
  - Si $c_{i,j} = Max$, entonces el nuevo color es 255.

  Y, por tanto, tendríamos una recta con la máxima pendiente posible entre el punto $x = Min$ y $x = Max$, partiendo el histograma en dos colores, 0 (negro) desde 0 hasta *Min* y 255 (blanco) desde *Max* hasta 255.

- Si *Min* = 0 y *Max* = 255, entonces la Ecuación 4.1 queda reducida a:

$$c_{i,j}^{nuevo} = c_{i,j}$$

lo que equivale a no realizar ninguna transformación en los píxeles de la imagen. La recta obtenida sería $y = x$, donde cada punto mantiene su color original (Figura 4.4(d)).

En resumen, el resultado final de esta fase es la imagen original posiblemente aumentada en contraste; posiblemente porque, como se ha comentado en el último de los casos extremos, puede que no sea posible aumentar más el contraste de la imagen de entrada.

4.1.1.2 *Umbralización*

Una vez que hemos aumentado el contraste de la imagen de entrada, el siguiente paso es aislar la figura del fondo en la imagen, es decir, el proceso de *segmentación* del objeto del fondo. Concretamente utilizaremos el proceso de *binarización*, mediante el cual el fondo estará en color negro y el objeto en color blanco.

Este proceso necesita previamente un nuevo cálculo del histograma de la imagen, ya que la imagen de entrada ha podido verse modificada por el apartado anterior y, por consiguiente, también lo habrá hecho su histograma. Tras esto realizamos un simple cálculo para poder distinguir si la imagen es válida o no para nuestra aplicación:

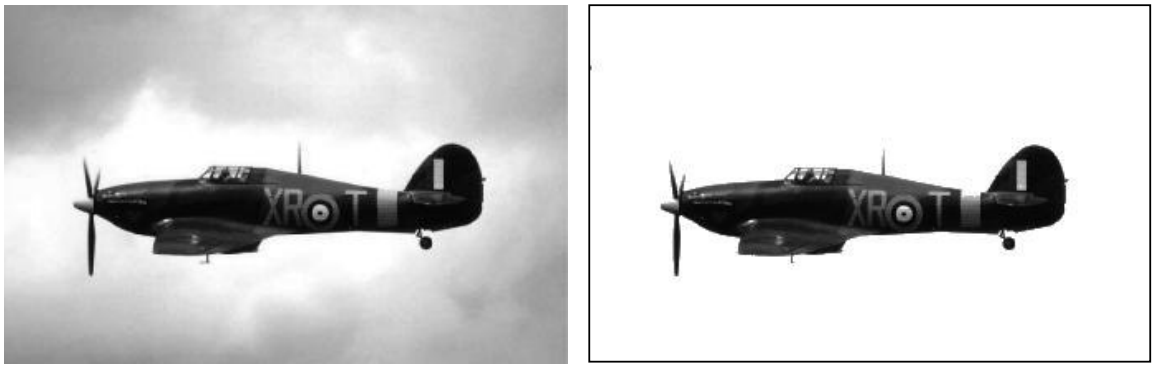
$$frc_{\max} > 0.5 \times ancho \times alto \qquad (4.2)$$

donde $frc_{\max}$ indica la frecuencia máxima del color más abundante en la imagen, mientras que *ancho* y *alto* nos indican las dimensiones de la imagen de entrada.

Por tanto, la Ecuación 4.2 especifica que solamente aquellas imágenes cuyo color más frecuente lo sea en más de un 50% respecto al número total de puntos de la imagen, serán

consideradas imágenes con fondo homogéneo, es decir, imágenes con un fondo claramente definido. Todos los puntos de la imagen con un color distinto al del fondo (en un intervalo que definiremos a continuación) serán considerados puntos del objeto.

Lógicamente, esto supone una reducción bastante importante en el número y tipo de imágenes de entrada válidas en este proceso, ya que por ejemplo, imágenes como la de la Figura 4.5(a) no pueden ser utilizadas.



(a) (b)

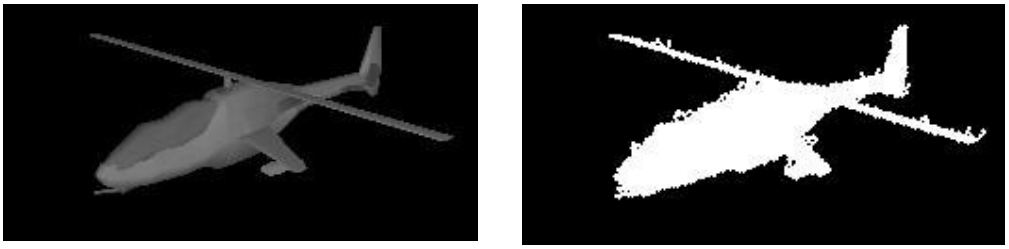
Figura 4.5: (a) Imagen *Hurrican.bmp*, con un fondo no homogéneo. (b) Imagen modificada con fondo homogéneo.

Para el ejemplo de la Figura 4.5(a), podríamos utilizar cualquier programa de retoque fotográfico y obtener la Figura 4.5(b), que es perfectamente válida (cumple la Ecuación 4.2) y, por consiguiente, se obtendrá satisfactoriamente el contorno del avión.

Hemos buscado la simplicidad en esta fase del proyecto, para luego centrar el esfuerzo en la fase de *Comparación de Imágenes*, que es realmente nuestro objetivo.

Como se ha descrito anteriormente, será considerado como objeto todo aquel punto cuyo color sea distinto del color de fondo. Pero resulta conveniente considerar como color de fondo un rango de valores centrados en el color de fondo resultante, es decir, si el color más frecuente del histograma es x, entonces, consideraremos como color de fondo todos aquellos puntos cuyo color se halle en el intervalo $[x - margen_{umbraliz.}, x + margen_{umbraliz.}]$, donde $margen_{umbraliz.}$ es un valor configurable y su valor por defecto es 5.

Este aspecto de la umbralización es delicado, puesto que si el color de fondo es compartido con la frontera del objeto, podemos obtener formas con mucho ruido, con muchos entrantes y salientes que van a ocasionar un aumento del número de puntos característicos en la fase final, debido a que el objeto toma puntos del fondo. Esto nos lleva, en la etapa de búsqueda, a una distinción con imágenes en principio similares, si éstas no han tenido este mismo problema en su frontera. Mostramos en la Figura 4.6 un ejemplo con el valor de *margen de umbralización* por defecto, es decir, 5.



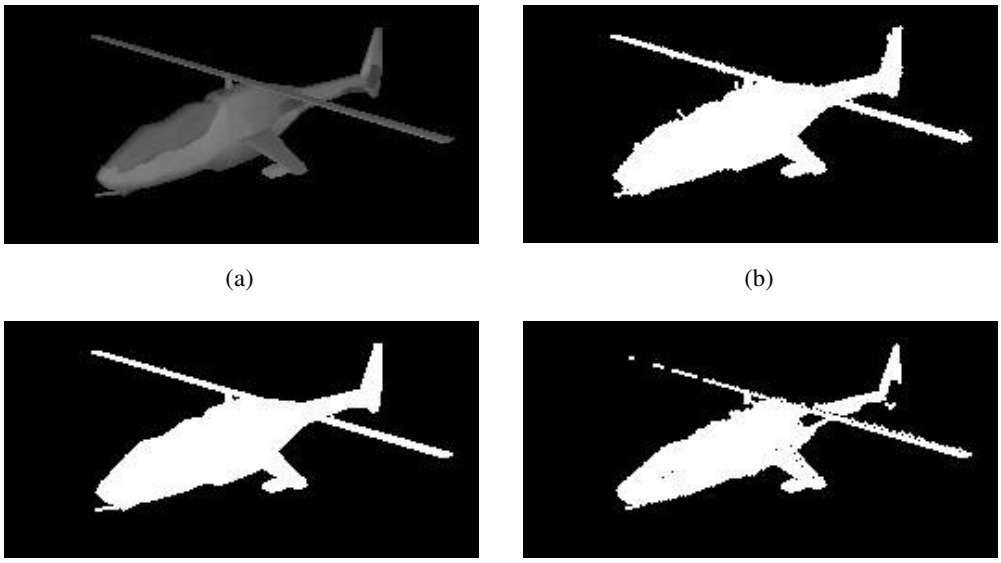
(a) (b)

Figura 4.6: (a) Imagen *AH-84.bmp*. (b) Imagen umbralizada con $margen_{umbraliz.}$ = 5.

Finalmente, obtenemos que la imagen umbralizada tiene mucho ruido y esto puede producir malos resultados en etapas posteriores. Por supuesto, podemos modificar este margen e intentar corregir esta situación. A continuación mostramos en la Figura 4.7 varios resultados con distintos valores de *margen de umbralización*.

En la Figura 4.7(b) se aprecia una mejora notable, pero es en la Figura 4.7(c) donde se aprecia la forma del helicóptero claramente, sin ruidos, lo que permitirá obtener un contorno preciso y adaptado realmente a la figura de la imagen de entrada.

Evidentemente, un margen de umbralización muy grande puede provocar el mismo efecto, aunque ahora es el fondo el que toma puntos del objeto, reduciendo el tamaño de éste y desfigurando la forma, como muestra la Figura 4.7(d).



(a) (b)

(c) (d)

Figura 4.7: (a) Imagen *AH-84.bmp*. (b) Imagen umbralizada con $margen_{umbraliz.}$ = 8. (c) $margen_{umbraliz.}$ = 20. (d) $margen_{umbraliz.}$ = 150.

Será conveniente, por tanto, elegir un valor adecuado de *margen de umbralización* para poder conseguir, en la siguiente etapa del proceso, un contorno fiel a la forma y evitar resultados inesperados en las etapas de búsqueda.

El proceso de umbralización acaba aquí, pero puede ocurrir que, en determinadas imágenes, sea el objeto el que esté considerado como fondo y viceversa, el fondo sea considerado como objeto. Supongamos una imagen, como la de la Figura 4.8(c), en la que el histograma nos indica que el color más abundante es el gris, que es el color del rectángulo, es decir, el color del objeto. En principio, el objeto considerado sería el marco que rodea a ese rectángulo (en negro), mientras que el fondo lo constituiría el propio rectángulo (en gris). Para solucionar esto, una vez que se ha umbralizado, se comprueba el número de puntos del objeto que son adyacentes a los bordes de la imagen. Si este número de puntos es mayor que el 50% del perímetro de la imagen (no del objeto), entonces intercambiamos objeto por fondo.

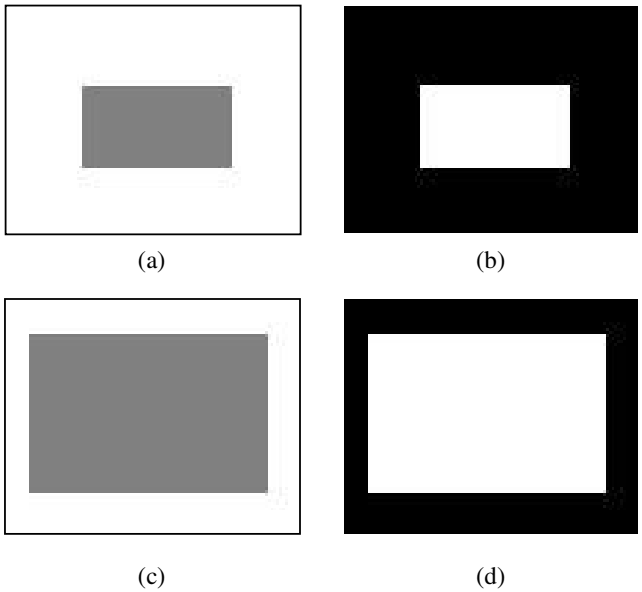


Figura 4.8: (a) Imagen *rectangulo.bmp*. (b) Imagen *(a)* umbralizada. (c) Imagen *rectangulogran.bmp*. (d) Imagen *(c)* umbralizada.

En el caso de la Figura 4.8(a), el color mayoritario es el blanco y, por consiguiente, no es necesario intercambiar objeto por fondo, porque se decide correctamente. Sin embargo, en el caso de la Figura 4.8(c), el color mayoritario es el gris (rectángulo). Por tanto, se comprueba que el objeto considerado limita en más del 50% con los bordes de la imagen (limita con todos). En consecuencia, se invierten los papeles y objeto y fondo se intercambian, obteniendo el resultado esperado que se muestra en la Figura 4.8(d).

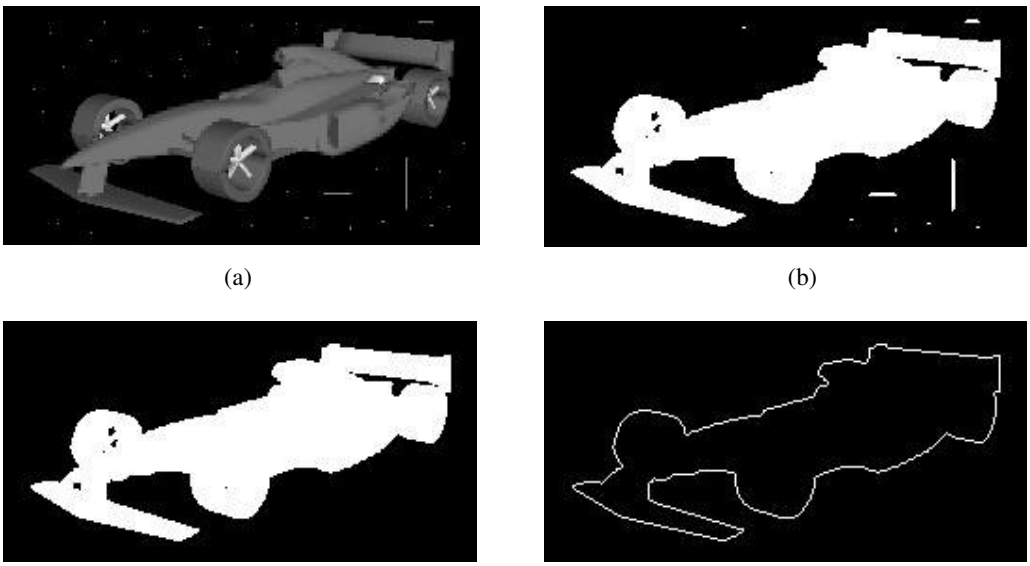
En este otro ejemplo de la Figura 4.9, también se considera como objeto el rectángulo, ya que el marco, aunque posea menos puntos, "toca" directamente con los límites de la imagen en más de un 50% de los puntos adyacentes al borde de la imagen, por lo que se considera al marco como fondo y al rectángulo como objeto.

Figura 4.9: (a) Imagen de entrada. (b) Imagen umbralizada.

4.1.1.3 *Filtro Anti-Ruido*

Tras haber distinguido objeto de fondo mediante la umbralización, (objeto de color blanco y fondo de color negro), solamente nos queda en esta fase, eliminar posible ruido en la imagen y corregir ciertas situaciones determinadas que pueden provocar, si no se solucionan ahora, problemas en la fase de *Obtención del Contorno*.

En primer lugar, tratamos de eliminar ruido aislado que pueda aparecer en la imagen, puesto que el algoritmo de obtención del contorno de la siguiente fase puede considerar como objeto alguno de esos píxeles que forman ruido y llevarnos a una situación errónea. Para ello, realizamos varias iteraciones sobre la imagen (hasta que no existan píxeles que eliminar), y eliminamos todos aquellos puntos cuyo número de píxeles vecinos sea igual o inferior a *Nº de vecinos*. Este valor es configurable y tiene un valor de 2 por defecto. Un ejemplo sobre esto se muestra en la Figura 4.10.



(a) (b)

(c) (d)

Figura 4.10: (a) Imagen *formula1.bmp*. (b) Imagen umbralizada con *nº vecinos* = 0. (c) Imagen umbralizada con *nº vecinos* = 2. (d) Imagen del contorno.

En la imagen umbralizada de la Figura 4.10(c) apreciamos como el ruido ha sido eliminado por el filtro anti-ruido, que ha suprimido los píxeles cuyo número de puntos adyacentes (vecinos) sea igual o inferior a 2 (por defecto), aunque este parámetro puede

modificarse (de 0 a 3). Así, no se pierde tiempo desechando pequeños contornos inválidos (porque no tienen la longitud mínima requerida), como muestra la Figura 4.10(b).

En la Figura 4.10(b) hemos mostrado la imagen umbralizada obtenida si la opción "*Nº de vecinos*" se establece a 0 sobre la imagen de entrada de la Figura 4.10(a). El ruido consistente en único píxel aislado del objeto (blanco) ha sido eliminado (no tiene vecinos), pero el resto no puede ser suprimido. Si en la fase posterior de obtención del contorno, nos encontramos con una de estas falsas formas, se desechará posteriormente por longitud de contorno menor que la cota inferior establecida. Por tanto, el contorno obtenido (cuyo análisis realizaremos en el siguiente apartado) es el mismo prácticamente que con *Nº Vecinos* igual a 2 (Figura 4.10(d)). No obstante, el tiempo invertido en calcular un contorno inválido y desecharlo es mayor que aplicar este filtro varias veces hasta dejar la forma limpia de ruido puntual y lineal.

Es interesante recordar con esta última figura como el contorno extraído de la imagen umbralizada es únicamente el contorno exterior de la misma, ignorándose cualquier tipo de hueco existente en el interior de la misma.

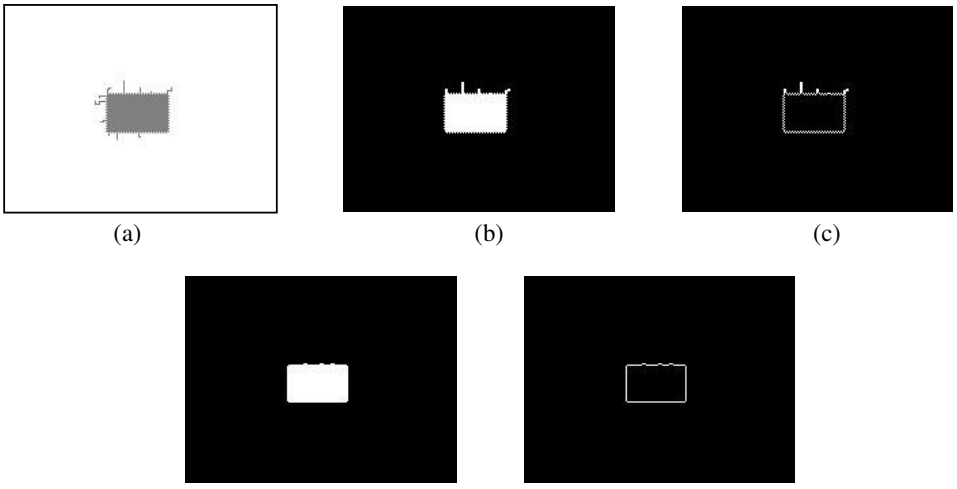


Figura 4.11: (a) Imagen de entrada. (b) Imagen umbralizada con *Nº vecinos* = 2. (c) Imagen del contorno a partir de *(b)*. (d) Imagen umbralizada con *Nº vecinos* = 3. (e) Imagen del contorno a partir de (d).

Las figuras 4.11(b) y 4.11(c) nos muestran las imágenes umbralizadas y del contorno para un valor de "*Nº Vecinos*" igual a 2. Los pequeños entrantes no pueden ser eliminados y provocan un contorno que sigue las pequeñas aristas, lo que implica que la imagen sea considerada muy distinta a un rectángulo normal de las mismas dimensiones.

En cambio, las figuras 4.11(d) y 4.11(e) nos muestran las imágenes umbralizadas y contorno para un valor de "*Nº Vecinos*" igual a 3. El resultado es que obviamos las aristas delgadas y las consideramos como ruido (desechándolas, por tanto).

Por tanto en una imagen muy accidentada, si deseamos precisión, debemos poner un valor bajo de la opción "*Nº Vecinos*", que tiende a extraer un contorno más fiel y exacto respecto a la imagen de entrada. Si deseamos una visión más global de la figura, podemos poner un valor mayor de este parámetro.

En segundo lugar, este filtro corrige una serie de situaciones que pueden darse en determinadas imágenes para evitar fallos en la obtención del contorno. Veamos un ejemplo en la Figura 4.12 para explicar más fácilmente este hecho.

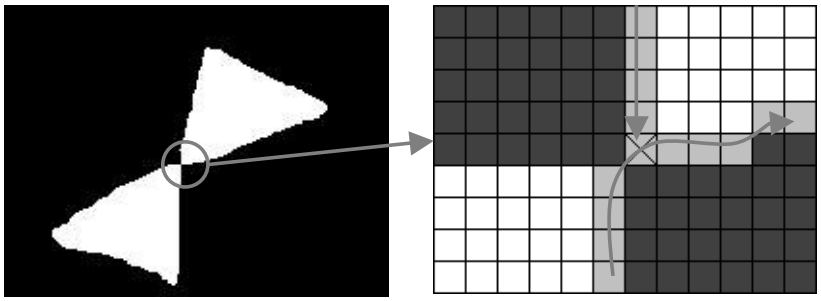


Figura 4.12: (a) Imagen de entrada. (b) El contorno no puede continuar por un punto ya visitado.

Esto ocurre aunque pongamos la opción *Nº Vecinos* a 3, porque los puntos de unión entre ambos "triángulos" tienen, cada uno de ellos, 4 vecinos en total y, por tanto, no pueden ser eliminados. Pero, aunque pudieran ser eliminados, lo que nos interesa realmente es proporcionarle al algoritmo de obtención del contorno una arista con dos píxeles de grosor, al menos, para que pueda pasar el contorno por un sentido y volver por el otro y así, completar el perímetro de la figura. Si lo eliminásemos, solamente podríamos obtener uno de los dos

"triángulos" como objeto de la imagen. En este tipo de situaciones (Ver Figura 4.13), se añade un punto extra que no varía el contorno apenas, pero permite que el algoritmo de obtención del contorno pueda completar la figura entera.

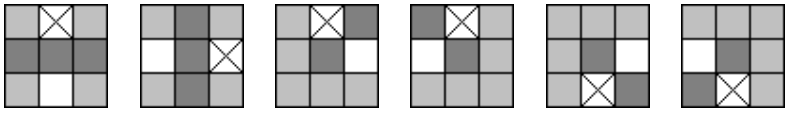


Figura 4.13: De (a) a (f) Casos posibles en el proceso de aumento de grosor de las aristas.

La Figura 4.13 nos muestra todas las combinaciones posibles en las que intervendrá este filtro para hacer más gruesas las aristas. Se aplicará únicamente en puntos cuyo número de vecinos sea superior al valor de *Nº Vecinos* establecido, puesto que si no lo cumple será eliminado por el filtro anterior.

El punto central representa el centro de la máscara que estamos analizando. Todos los puntos gris oscuro representan puntos del objeto. Los puntos blancos implican puntos del fondo de la imagen. Los puntos de color gris claro pueden ser del objeto o del fondo porque no importan en el estudio de cada caso, salvo el hecho de que algunos tendrán que ser del objeto para cumplir que el número de vecinos sea superior al valor *Nº de vecinos* establecido. Por último, el punto marcado con una aspa sobre un fondo blanco es el que se añade para hacer más gruesa la arista.

Teniendo en cuenta que este proceso se realiza para cada punto de la imagen, el resultado es que evitamos situaciones en las que el algoritmo de obtención del contorno no puede volver al punto inicial del mismo porque existe un estrechamiento (de un píxel de grosor) como muestra la Figura 4.12 y, como no se pueden repetir píxeles en el contorno, es imposible completar el perímetro de la figura.

4.1.2 Obtención del Contorno

Una vez realizadas las transformaciones y filtros anteriores, debemos encontrarnos con una imagen sin ruido, salvo zonas del objeto demasiado gruesas para considerarse como tal. Además, tendremos en color blanco el objeto y en color negro el fondo. Necesitamos, a continuación, ejecutar un algoritmo que nos permita seguir fielmente el borde del objeto y nos devuelva un vector con una sucesión de puntos cíclica (comienza y termina en el mismo punto) que represente este contorno del objeto.

El *Algoritmo de la Tortuga* es el método más simple para este fin, ya que aplicar métodos como el *método de detección de ejes de Canny* [Cann86] (más complejo y más completo) no tiene sentido en una imagen con dos colores únicamente, es decir, con imágenes umbralizadas.

Además, el *Algoritmo de Canny* presenta el inconveniente de que no todos los contornos generados terminan cerrados, ya que aparecen discontinuidades en él. Ni siquiera en las imágenes simples para las que funciona el *Algoritmo de la Tortuga* devolvía un contorno cerrado el *Método de Canny*, por lo que se convertía en imprescindible implementar un método que completara el contorno calculado por el *Método de Canny*, aumentando aun más la complejidad del proceso.

Vamos a utilizar una variante del *Algoritmo de la Tortuga* diseñada para este proyecto, usando la relación 8-vecinos y con ciertos aspectos distintos y más completos que especificamos a continuación. Su funcionamiento es bastante simple y se describe mediante la Figura 4.14.

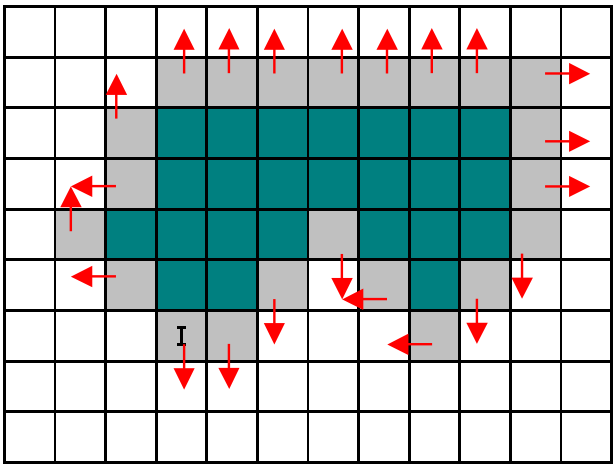


Figura 4.14: Ejemplo de la ejecución del *Algoritmo de la Tortuga*. Se muestra el contorno final obtenido.

En primer lugar, se busca el primer punto del objeto (el orden es de abajo a arriba y de izquierda a derecha). El punto localizado está etiquetado en la Figura 4.14 como *I*. Por tanto, *I* es el punto inicial (y, si todo ocurre correctamente, el punto final) de nuestro contorno. A continuación se busca el primero de los vecinos válidos que continúe el borde del objeto. Los vecinos son buscados en el sentido contrario a las agujas del reloj a partir de la flecha indicada en cada punto.

El primer punto (*I*) comprueba si el contorno continua justo debajo de él. Al comprobar que el punto corresponde al fondo, prueba el siguiente punto en el sentido anti-horario, es decir, el que está en la diagonal abajo-derecha. Tampoco pertenece al objeto. Seguidamente, continuando este orden, se prueba el punto situado a la derecha y se comprueba que éste sí pertenece al objeto y, además, es un punto *válido*, por lo que ese punto se suma al vector del contorno. Este nuevo punto pasa a ser el actual y volvemos a buscar el siguiente vecino. Así continuamos hasta encontrarnos con el punto inicial (éxito) o hasta que no encontremos vecinos posibles para continuar el contorno (fracaso). Además, para que un punto vecino se considere *válido*, debe cumplir varias condiciones:

- Sea un punto, evidentemente, que forme parte del objeto.
- No exista ya en el vector del contorno (salvo el caso del punto inicial, en el que habríamos concluido el proceso).

- Sea adyacente (con la relación 4-vecinos) con al menos, un punto del fondo, para asegurarnos que estamos tomando verdaderamente un punto del contorno y no del interior del objeto (*salida 4 vecinos*).
- Que tanto el punto actual como el punto vecino sean adyacentes a un mismo punto del fondo con la relación 8-vecinos (*salida común 8-vecinos*).

Las flechas indican, como hemos dicho, el primer vecino a analizar en el contorno. La Figura 4.15 nos muestra el criterio que se sigue para calcular esta dirección inicial de búsqueda de puntos del contorno.

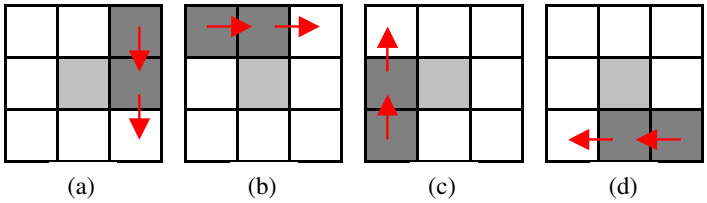


Figura 4.15: Los ocho posibles casos de inicio de posibles vecinos.

Los puntos gris oscuro nos indican los dos posibles puntos actuales en cada caso, mientras que el gris claro nos indica el punto anterior del contorno. Por ejemplo, en el primero de los casos de la Figura 4.15, el vecino primero que hay que examinar es el situado debajo del punto gris oscuro en cuestión, que es el apuntado por la flecha correspondiente. Además, al iniciar un contorno, se considera el caso 4.15(a), es decir, que el punto anterior al de inicio y el propio punto de inicio forman una línea horizontal y, por tanto, le corresponde analizar como primer posible vecino el punto justamente debajo de él.

El hecho de elegir estos puntos de inicio no es aleatorio, ni mucho menos. Si eligiésemos un punto de inicio al azar, podríamos tener complicaciones en imágenes con huecos como la de la Figura 4.16. Podemos apreciar en este ejemplo como el punto marcado con el aspa blanca es igualmente válido como siguiente vecino que el punto marcado con el aspa negra, ya que no ha sido considerado antes, es adyacente (en la relación 4-vecinos) con el punto del fondo justo debajo de él y comparte (en la relación 8-vecinos) dos puntos

adyacentes con el punto actual. Pero, sin embargo, no forma parte del contorno real de la figura.

La causa es que un hueco del objeto tiene el color del fondo y, por tanto, se considera como tal. La solución para esto es comenzar a buscar los vecinos siempre por un punto que tenga garantizado que no pertenezca a un hueco del objeto. Es por esto que la flecha vertical de la figura indica una dirección que, con toda seguridad por como hemos planteado el problema, no pertenece a un hueco. Esta dirección corresponde al caso de la Figura 4.15(c) indicado por esta flecha en la Figura 4.16, en la que además, la flecha horizontal indica los dos puntos implicados en ese estudio (el punto anterior y el punto actual - el apuntado -).

Por tanto, como el píxel al que apunta la flecha vertical no es del objeto, obtenemos el siguiente punto en sentido anti-horario, que es el indicado por el aspa negra, obviando así el punto del aspa blanca que, aunque es válido, es analizado siempre después. De esta manera, solventamos el problema de los huecos y obtenemos un contorno exacto de la forma en cuestión.

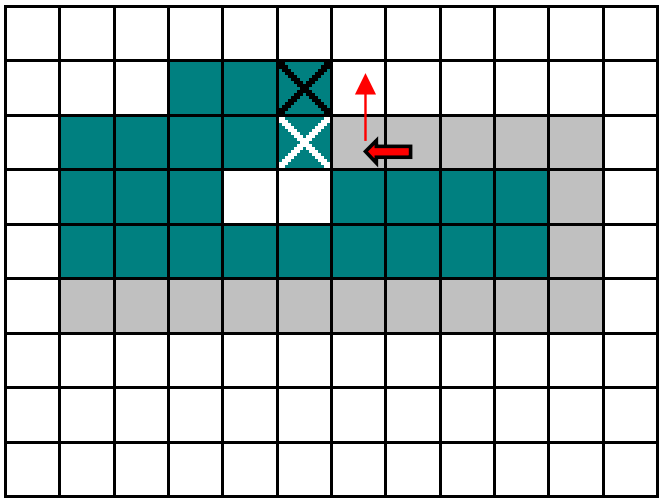


Figura 4.16: Ejemplo de obtención del contorno en una imagen con huecos.

El otro problema que podemos encontrarnos al intentar obtener el contorno de una imagen es que ésta tenga ruido demasiado grande para que haya sido eliminado por el *Filtro Anti-Ruido*. En ese caso podría ocurrir que se tomara como contorno el borde del ruido. Para

evitar esta situación, se calcula una cota mínima del número de puntos que debería tener al menos la imagen (basada en el número total de puntos de la imagen). Si el contorno obtenido no llega a tener esa mínima cantidad de puntos, será considerado como ruido y será ignorado. Esta cota mínima queda establecida por la siguiente expresión:

$$CotaMin = 2 \cdot (diagonal - 1)$$

donde *diagonal* representa la longitud de la diagonal de un supuesto cuadrado situado con uno de sus vértices hacia abajo. Esto es así porque el mínimo perímetro posible con un número determinado de puntos del objeto se consigue con esta forma, ya que las líneas en diagonal avanzan $\sqrt{2}$ veces más que en vertical y horizontal. La variable *diagonal* se define como sigue (*nptos* es el número total de puntos del objeto):

$$diagonal = \lfloor \sqrt{2 \times nptos} \rfloor$$

Es decir, este cálculo trata de aproximar, con el número total de puntos de la imagen, el cuadrado más parecido en cuanto a número de puntos se refiere. Si por ejemplo, la imagen tiene 28 puntos, *diagonal* valdrá 7, como en la Figura 4.17 y, por tanto, *CotaMin* será igual a 12, como efectivamente corresponde al contorno de dicha figura.

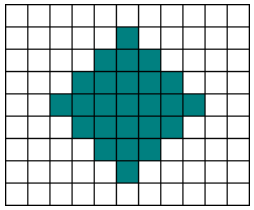
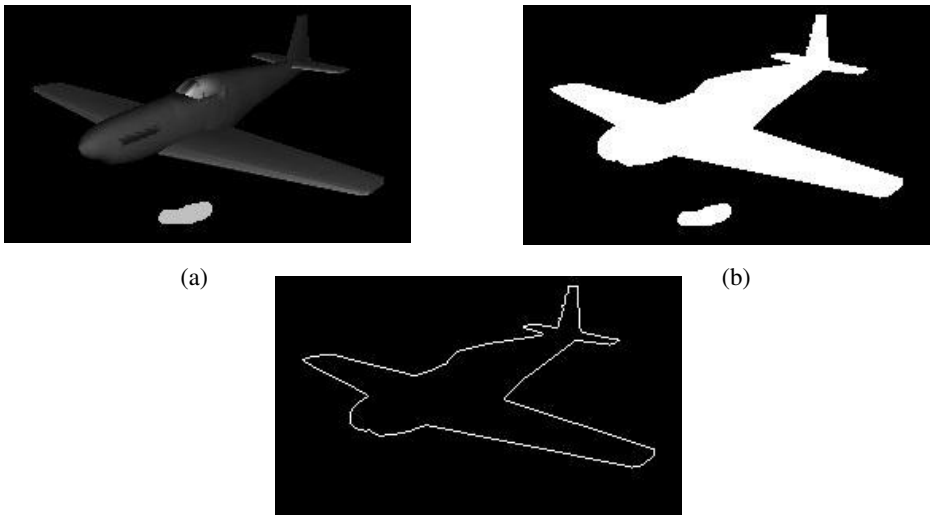


Figura 4.17: Cuadrado con uno de sus vértices hacia abajo

Si el contorno obtenido no supera este umbral, se considera que nos hemos encontrado con ruido, porque es imposible obtener un contorno de longitud menor que *CotaMin* con ese número de puntos del objeto. Por ello, el verdadero objeto no ha sido localizado aún. Suprimiremos los puntos del falso contorno y volveremos a iniciar el proceso de nuevo hasta dar con un contorno correcto o hasta quedarnos sin puntos del objeto.

En el siguiente ejemplo (Figura 4.18), se obtiene correctamente el contorno de la imagen, a pesar de que previamente se encuentra con ruido que no ha podido ser eliminado por el filtro anti-ruido.



(a) (b)

(c)

Figura 4.18: (a) Imagen de entrada. (b) Imagen umbralizada. (c) Imagen del contorno.

En este ejemplo, la imagen de entrada tiene una parte que no forma parte del objeto. Aparece en la Figura 4.18(a) y 4.18(b), ya que la umbralización no puede eliminar ruido de tal magnitud. Sin embargo, la Figura 4.18(c), que muestra el contorno final obtenido, no incluye este ruido puesto que, aunque ha sido considerado por la umbralización, la fase de obtención del contorno lo desecha por tener un perímetro inferior a la cota mínima establecida. Por consiguiente, se elimina ese contorno, volvemos a aplicar el filtro anti-ruido y se busca de nuevo un contorno correcto, pero sin tener en cuenta los puntos del contorno desechado. Al final, únicamente encontramos un contorno válido, el que se muestra en la Figura 4.18(c), si es que es posible encontrar alguno.

4.1.3 Cálculo de la Función de Curvaturas

En esta etapa, partimos de una sucesión de puntos que identifican el contorno de la imagen de entrada y tratamos de obtener una función con los valores de curvatura de cada punto del perímetro.

Cada valor (x,y) de esta función representa el valor de curvatura (y) del punto x del contorno de la imagen. Por ejemplo, el valor de curvatura del punto $x_0 = 0$ representa la curvatura del punto inicial del contorno (y_0).

La función de curvatura utilizada en este proyecto es mostrada en la Sección 2.9 con un mayor nivel de detalle. Aquí, únicamente mostraremos su comportamiento más funcional y práctico.

El valor de curvatura está en el rango $[-\pi,+\pi]$, donde un valor positivo de curvatura indica un punto de curvatura convexa, mientras que un valor negativo de curvatura indica un punto de curvatura cóncava. Además, independientemente del signo, un valor cercano a 0 indicará ausencia de curvatura o curvatura muy escasa, mientras que un valor cercano a $|\pi|$ indicará una curvatura muy aguda.

Tenemos que proporcionarle a la función de cálculo de curvatura que hemos utilizado, un valor especial, denominado *sigma*. Este valor determina el tamaño del entorno de curvatura a analizar centrado en cada punto considerado, es decir, cuántos puntos del contorno tenemos en cuenta alrededor de cada punto analizado para calcular la curvatura que posee.

En la gráfica correspondiente a la Figura 4.19 (función de curvatura de un cuadrado), se observa, además de la curva de curvatura, dos líneas discontinuas paralelas al eje x. Estas líneas definen los umbrales inferior y superior, que tendrán su utilidad en la siguiente etapa de *Extracción de Puntos Característicos*.

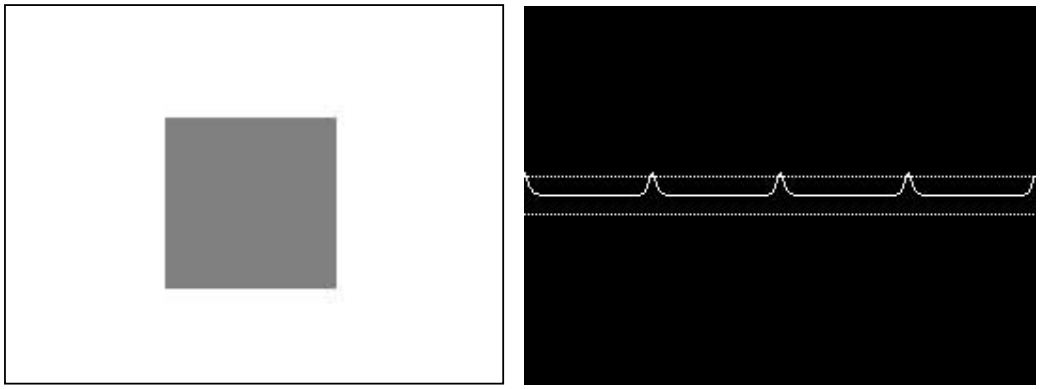


Figura 4.19: (a) Imagen *cuadrado.bmp* (b) Gráfica de curvatura de la imagen *cuadrado.bmp*

En la Figura 4.19(b) vemos la gráfica de curvatura obtenida para la imagen de la Figura 4.19(a). Apreciamos en la gráfica de curvatura los cuatro picos que identifican a un cuadrado. Téngase en cuenta que el pico de la derecha es el mismo que el de la izquierda, ya que el contorno es una estructura circular. Los umbrales están establecidos en el 10% y cada pico tiene una curvatura del 12% (todos iguales al ser un cuadrado), por lo que aparecen un poco por encima del umbral superior. Además, el primero de los picos está justo al principio de la gráfica, debido a que el punto inicial para calcular el contorno es la esquina inferior-izquierda del cuadrado, es decir, justo en uno de los vértices de la forma. Por otra parte, las distancias entre un pico y el siguiente (incluyendo el último con el primero) son exactamente iguales, como ocurre en un cuadrado.

La cuestión principal que se plantea en esta sección es qué valor de *sigma* poner (en el grupo de imágenes) para obtener una función de curvatura ideal. Si bien existen métodos que nos permiten estimar el *sigma* adecuado para cada imagen [ArGa98], puede no ser coherente aplicar valores de *sigma* distintos para imágenes del mismo tipo. Veamos algunos ejemplos en la Figura 4.20 y analicemos los resultados.

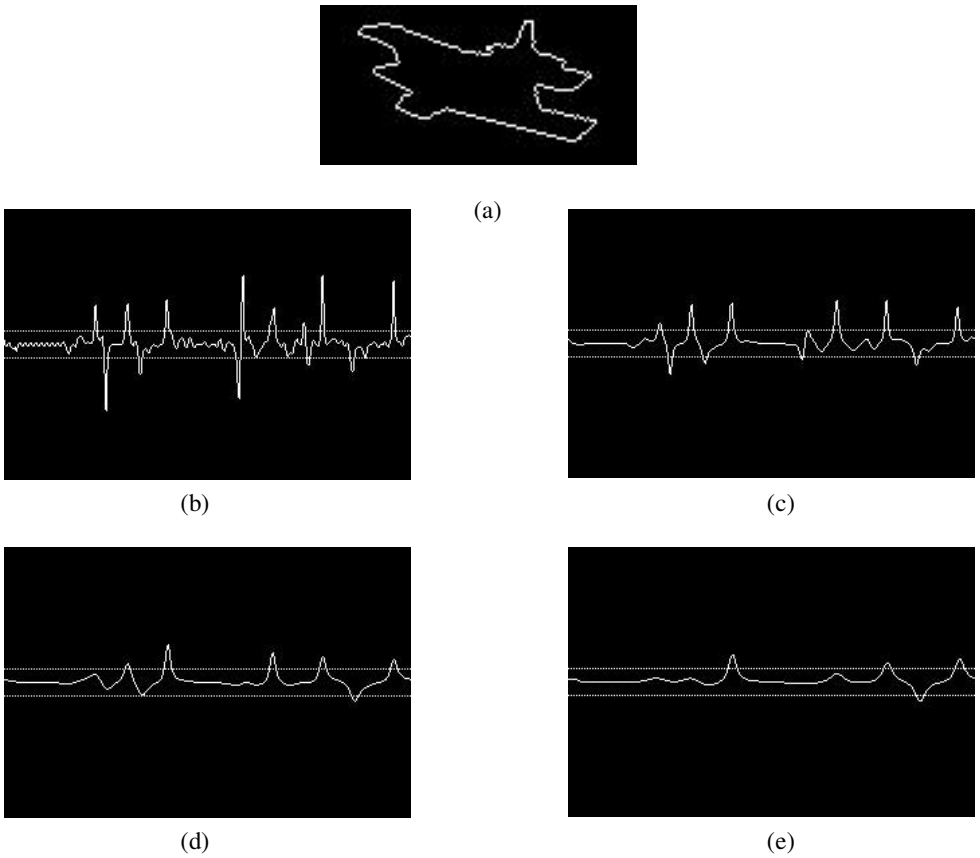


Figura 4.20: (a) Imagen contorno. (b) Gráfica de curvatura con *sigma* = 1'5. (c) Con *sigma* = 3. (d) Con *sigma* = 6. (e) Con *sigma* = 10.

Como se puede apreciar en la Figura 4.20, a medida que vamos aumentando el valor de *sigma*, la gráfica de la curvatura se va suavizando, lo que provocará la extracción (en la fase siguiente) de un número menor de puntos característicos. Es decir, aumentar el valor de *sigma* implica la obtención de un número menor de puntos característicos de la imagen estudiada.

Un *sigma* pequeño implica utilizar poca información de curvatura por cada punto del contorno, mientras que un *sigma* grande implica analizar muchos puntos vecinos a cada punto, manteniendo únicamente los picos más representativos de la figura. Sin embargo, un *sigma* grande provoca también una disminución en la magnitud de curvatura de los puntos. Podemos observar la evolución del primer pico negativo de la izquierda (por ejemplo) de todas las gráficas y apreciaremos como va menguando su tamaño a medida que aumentamos el valor de *sigma*.

Además, los picos que se mantienen en las gráficas a medida que incrementamos el valor de *sigma* no tienen por qué ser los más agudos, sino aquellos que implican en su curvatura a un mayor número de píxeles. Puede incluso suceder que, picos que en una gráfica aparezcan muy juntos, al observarlos en otra gráfica con un *sigma* mayor se hayan unido en un único pico.

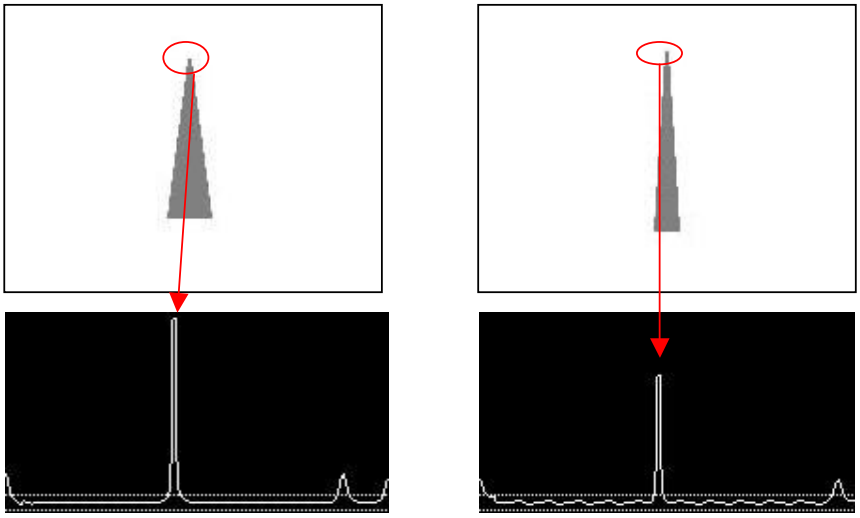
En general, dependiendo de cada imagen y de los aspectos que deseemos resaltar de cada una de ellas, utilizaremos un valor de *sigma* u otro. Eso sí, es realmente necesario mantener un criterio para todas las imágenes de un mismo grupo, puesto que si no se hace podríamos obtener resultados verdaderamente incongruentes.

Antes de acabar este apartado, queremos mencionar tres problemas encontrados en la ejecución de la función de curvatura en determinadas imágenes.

Uno de los problemas que se manifestaron en el cálculo de la función de curvatura es que, si el punto inicial del contorno (o uno cercano a él) tiene una curvatura considerable, la función no lo detectaba. Por ejemplo, en el caso de la Figura 4.19(b), el primer pico (que también tiene una parte en el final de la gráfica) no aparecía, obteniéndose un total de 3 puntos característicos en un cuadrado, lo cual no hace falta decir que era incorrecto.

La solución que se tomó fue la siguiente: una vez calculada la función de curvatura, se busca un punto en la gráfica que no supere ninguno de los umbrales en un rango centrado en ese punto (*punto liso*). A continuación, calculamos de nuevo la función de curvatura pero, esta vez, con el punto inicial del contorno en ese *punto liso*, es decir, desplazando hacia la izquierda el contorno en *punto liso* unidades. Tras esto, presentamos la gráfica (ahora contemplando todos los picos existentes) desplazando de nuevo el resultado en *punto liso* unidades, pero esta vez a la derecha. El resultado es que obtenemos la gráfica como si se hubiese calculado desde el punto inicial del contorno, pero evitando el error de que no capte los posibles valores iniciales de curvatura. De este modo, podemos ver un resultado correcto en la Figura 4.19(b).

Otro caso mucho más grave que se ha producido utilizando esta función de curvatura es el que se explica a continuación, utilizando las dos imágenes siguientes y sus dos funciones de curvatura respectivas (ver Figura 4.21).



(a) (b)

Figura 4.21: (a) Imagen *triangulo7.bmp* y su gráfica de curvatura. (b) Imagen *triangulo8.bmp* y su gráfica de curvatura.

A pesar de que el triángulo de la imagen de la Figura 4.21(a) tiene el vértice central más agudo que el de la imagen 4.21(b), la gráfica de curvatura de ésta última, nos muestra menor curvatura que la primera, además de mucho ruido en las zonas donde no existe curvatura (aristas).

La función de curvatura no responde satisfactoriamente cuando tenemos picos muy agudos, considerándolos de menor curvatura que la esperada. Podemos intentar suavizar la forma mediante el filtro anti-ruido y con valores mayores de *sigma*, para evitar encontrarnos con este tipo de picos, pero no siempre da resultado.

El último caso de problemas con la función de curvatura es la aparición de picos dobles en lugares del contorno donde solamente aparece un vértice de curvatura. Veamos un ejemplo para mayor claridad en la Figura 4.22.

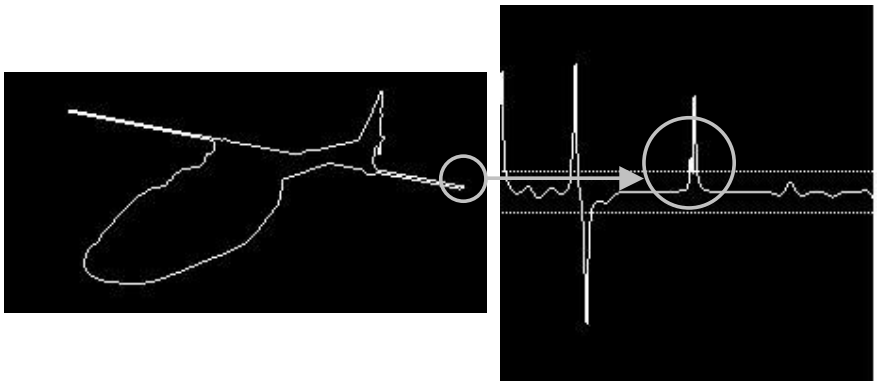


Figura 4.22: Imagen *Bell-216.bmp* y su gráfica de curvatura. Los umbrales superior e inferior están establecidos en 11.

La circunferencia grande de la derecha nos indica un doble pico: el primero de poca curvatura y, junto a éste, el segundo de curvatura notable. La distancia relativa entre ellos es de 0, y ambos tienen el mismo signo (positivo, porque son convexos). En principio, simplemente se trata de un borde muy agudo (como indica la circunferencia pequeña), pero debería responder con un pico muy agudo, pero nunca con dos. Nos encontramos ante otro error grave de la función de cálculo de curvatura, que parece no desenvolverse muy bien en estos casos extremos, es decir, con bordes demasiado "afilados", como el de la Figura 4.22. Este problema realmente viene causado por la utilización de esta función sobre un contorno digital, en el que los ángulos son múltiplos de 45º, y no de la función de curvatura en sí (véase Sección 2.9 para más información). Podemos modificar, nuevamente el filtro anti-ruido y el valor de *sigma* para intentar evitar este problema, aunque de nuevo, no siempre podremos soslayar el error.

4.1.4 Extracción de Puntos Característicos

Una vez obtenida la función de curvatura de la imagen de entrada, extraemos los puntos más representativos de la misma. De estos puntos no nos interesará solamente su valor de curvatura sino otros atributos necesarios para el proceso de *Comparación de Imágenes* que se abordará en los siguientes apartados.

La imagen del contorno obtenida, el valor de *sigma* de la fase anterior y los márgenes de curvatura inferior y superior son las piezas clave para configurar este proceso, igualmente importante que los anteriores para extraer la máxima información sobre la imagen de entrada.

En primer lugar, debemos buscar los denominados *intervalos de curvatura*, formados por varios puntos seguidos del contorno, cuyo valor de curvatura en valor absoluto supera los umbrales, es decir, si es positivo, el valor de curvatura es mayor que el umbral superior; si es negativo, el valor de curvatura es menor que el umbral inferior. El número de intervalos obtenidos coincide con el número de puntos característicos final de la imagen.

El proceso comienza por el punto inicial del contorno, examinando el valor de curvatura de cada uno de los puntos. Cuando encuentra un punto que supera el umbral, establece el inicio de un intervalo, anotando el punto de inicio de dicho intervalo. A continuación, sigue examinando puntos, buscando encontrar el primer punto que no supere el umbral para cerrar el intervalo. Este proceso se realiza hasta llegar al final del contorno. Esta primera parte es reflejada en la Figura 4.23, en un ejemplo sencillo.

En ella apreciamos en gris claro los intervalos obtenidos, mientras que, en gris más oscuro, tenemos el resto de puntos de la gráfica de curvatura. Los umbrales de curvatura superior e inferior se sitúan horizontalmente también en gris claro.

El proceso empieza desde el primer punto, es decir, desde el primer punto gris oscuro de la izquierda. Como no supera ningún umbral, el sistema continua analizando el siguiente punto. Así llegamos hasta el punto indicado por la primera flecha, que sí supera este umbral. Creamos el primer intervalo (de signo positivo), anotando su inicio, es decir, el punto número 7 del contorno.

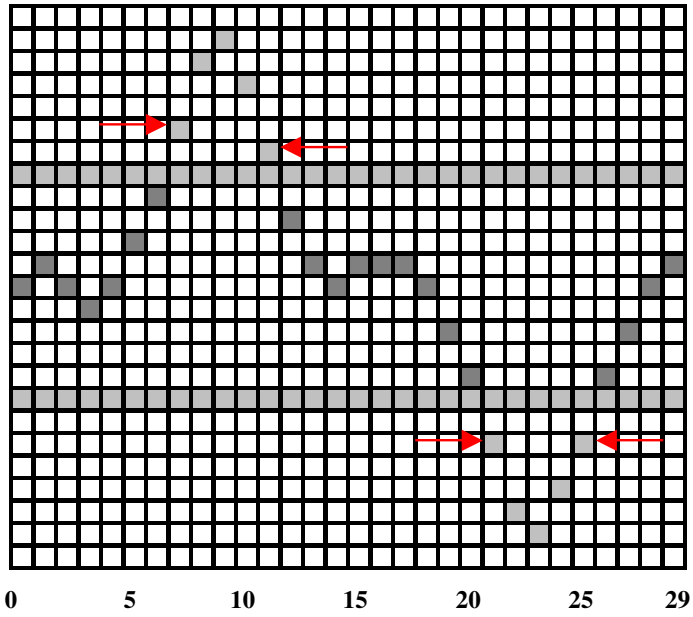


Figura 4.23: Ejemplo de la obtención de los intervalos de curvatura

Continuamos examinando puntos del contorno, pero ahora nos interesa encontrar el primer punto que no supere el umbral superior para cerrar el primer intervalo. Esto sucede en el punto número 12. Por ello, anotamos el punto anterior (el indicado por la flecha de cierre) como fin del intervalo. *1er intervalo* = [7,11] con signo *positivo*.

Volvemos a repetir el proceso desde el principio, es decir, tratamos de localizar un punto que supere el umbral de curvatura. Encontramos que el punto número 21 tiene una curvatura absoluta superior al umbral inferior de curvatura, por lo que creamos el segundo intervalo (este es negativo) anotando este punto como inicio de intervalo y buscamos el primer punto con curvatura absoluta inferior para cerrar el intervalo. El punto de cierre del segundo intervalo es el indicado por la segunda flecha de cierre, es decir, el punto número 25. *2º intervalo* = [21,25] con signo *negativo*.

El proceso continua de esta forma hasta que hayamos computado todos los puntos de la gráfica de curvatura. Como se ha dicho anteriormente, el número de intervalos obtenidos coincide exactamente con el número de puntos característicos de la imagen analizada.

Una vez obtenidos los intervalos, debemos calcular el punto del intervalo con la curvatura (en valor absoluto) máxima, para identificar qué punto del intervalo va a considerarse como punto característico. Para ello, recorremos todos los intervalos obtenidos y calculamos el punto de curvatura máxima en valor absoluto de cada intervalo, anotando este valor y su posición en el contorno. Tras esto, debemos calcular y almacenar la información obtenida, agrupándola en los siguientes atributos:

- *Número total de puntos característicos*, que coincide exactamente con el número de intervalos existentes.
- *Longitud total de la imagen*, que coincide con el número de puntos de los que consta el vector del contorno, ya que está medida en número de píxeles.
- *Orden* del punto característico. El primer punto característico obtenido es el número 1, el siguiente punto obtenido es el número 2, y así sucesivamente hasta obtener tantos puntos como indica el atributo *Número total de puntos característicos*.
- *Signo*. Indica si el punto característico en cuestión define una curvatura convexa (si es positivo) o cóncava (si es negativo).
- *Valor de curvatura*. Valor en tanto por ciento respecto a π que indica el grado de curvatura, en valor absoluto, del punto característico en cuestión. Por ejemplo, 50% representa una curvatura, en valor absoluto, igual a $\pi/2$.
- *Distancia absoluta* al siguiente punto característico. Representa el número de puntos (número de píxeles) del contorno existente entre el punto característico en cuestión y el siguiente. En el caso de que sea el último punto característico, es la distancia entre éste y el primero, es decir, teniendo en cuenta que el vector del contorno es cíclico.
- *Distancia relativa* al siguiente punto característico. Es calculada en tanto por ciento respecto a la *Longitud total de la imagen*, atributo indicado más arriba. Esta distancia nos va a permitir preservar la invariabilidad al tamaño en las búsquedas de imágenes, como se indicará posteriormente (véase Sección 7.1).

El proceso de *Caracterización de la Imagen* concluye guardando los atributos explicados justo arriba. En el caso de que estemos añadiendo una imagen a la base de datos, estos valores se almacenarán en la base de datos y concluirá el proceso de adición. En el caso de que estemos realizando una búsqueda, los valores de los atributos serán almacenados en la

memoria principal para ser comparados con los atributos de otras imágenes de la base de datos en la siguiente fase (*Comparación de Imágenes*).

4.2 Comparación de Imágenes

Ampliamos la información mostrada en la Figura 4.1, centrándonos en la caja *Comparación de Imágenes*, y analizando cada una de las tres subfases que la componen: *Preprocesamiento de la Consulta*, *Ejecución de la Consulta* y *Presentación de los Resultados*, como muestra la Figura 4.24.

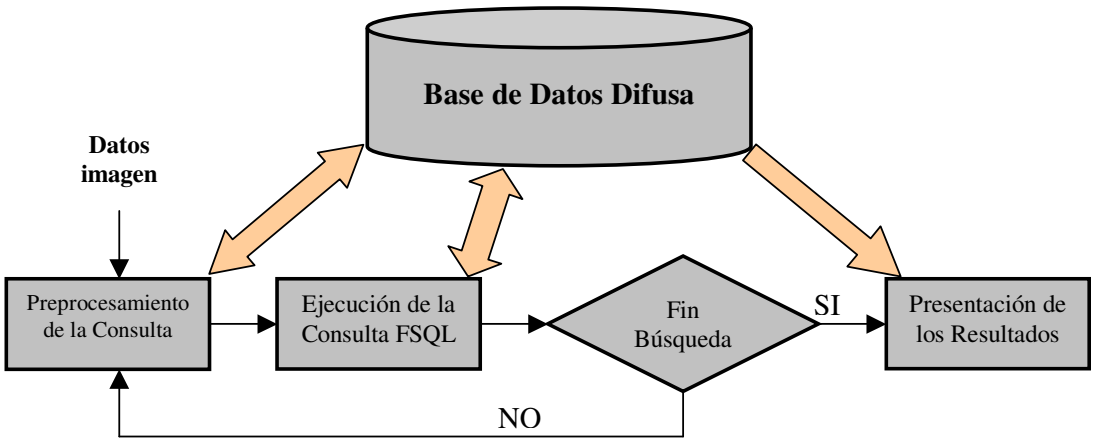


Figura 4.24: Esquema del Proceso de Comparación de Imágenes

Si realizamos una búsqueda de una imagen, habremos obtenido en la fase anterior los datos de la caracterización que serán la entrada de este proceso. En el caso de una búsqueda por datos, los puntos característicos serán los introducidos por el usuario, siendo innecesaria por tanto, dicha caracterización explicada en la sección anterior.

El funcionamiento general de esta fase es el siguiente: en función del tipo de búsqueda y de las imágenes implicadas en el proceso, prepararemos los datos necesarios en objetos temporales de la base de datos, durante la fase *Preprocesamiento de la Consulta*, de manera que igualemos el número de puntos característicos de la imagen de entrada y de las

imágenes de la base de datos para poder realizar la siguiente fase. Tras esto, construiremos y ejecutaremos las consultas (fase *Ejecución de la Consulta*). Pero, no tenemos por qué acabar aquí. Es posible que tengamos que ejecutar otro u otros grupos de consultas, que previamente tienen que ser preparadas, dependiendo de las imágenes y de las opciones que participen en dicha búsqueda.

Continuaremos de esta manera hasta ejecutar la última de las consultas necesarias. En ese instante se procederá a mostrar los resultados obtenidos en la búsqueda, durante la fase de *Presentación de los Resultados*.

Supongamos el siguiente ejemplo para aclarar el proceso de *Comparación de Imágenes*. Deseamos buscar una imagen en la base de datos. Hemos obtenido 10 puntos característicos de esa imagen de entrada mediante la fase de *Caracterización de la Imagen*. En la base de datos tenemos imágenes con 10 puntos característicos, pero también imágenes con más y con menos puntos que la imagen de entrada.

El proceso se basa en comparar, uno a uno, los puntos característicos de la imagen de entrada con los puntos de la imagen de la base de datos mediante una consulta FSQL, cuyo formato se muestra en la Sección 5.4. Aquellas imágenes que tengan exactamente 10 puntos podrán compararse directamente con la imagen de entrada a través de la consulta FSQL sin realizar apenas modificaciones (no existe prácticamente la fase de *Preprocesamiento de la Consulta*). Sin embargo, las imágenes de la base de datos que tengan, por ejemplo, 12 puntos tendrán que ver reducido este número a 10 para poder realizar tal comparación por pares de puntos. Por ello, la fase de *Preprocesamiento de la Consulta* se encarga de quitar 2 de los 12 puntos de estas imágenes para que la siguiente fase (*Ejecución de la Consulta FSQL*) se encuentre con todas las imágenes con el mismo número de puntos que la imagen de entrada y poder, de esta manera, realizar las comparaciones. En el caso de imágenes que tengan, por ejemplo, 8 puntos característicos, como no podemos "inventarnos" puntos, la solución adoptada es reducir el número de puntos de la imagen de entrada de 10 a 8, de manera que volvemos a igualar el número de puntos entre imagen de entrada e imágenes de la base de datos y podemos ejecutar (en la siguiente fase) la consulta FSQL.

Antes de comenzar a analizar todas las subsecciones, es necesario indicar que vamos a mostrar en el apartado *Preprocesamiento del la Consulta* y en el apartado *Ejecución de la Consulta* todos los posibles casos que pueden darse en función de las opciones de búsqueda e imágenes de la base de datos implicadas, y como preparan y ejecutan estos procesos una o varias consultas para realizar la búsqueda de cada uno de esos casos.

4.2.1 Preprocesamiento de la Consulta

En general, esta fase debe transformar los datos de la imagen de entrada (sus atributos) y los datos de las imágenes que van a ser comparadas con la de entrada (que están en la base de datos) para adecuar los atributos y poder realizar correctamente la comparación. Solamente podremos comparar imágenes con el mismo número de puntos característicos, sin embargo, no podemos obviar que imágenes muy parecidas (casi iguales) pueden diferir en algún punto característico y mantener un notable parecido. Por ello, el eje central de esta fase será convertir las imágenes al mismo número de puntos (lo que supondrá una penalización) para poder desarrollar posteriormente, una consulta difusa correcta, valiéndose para ello de objetos temporales y otras consultas a la base de datos.

En cuanto a la penalización, se hará siempre en función del número de puntos de la figura. No puede ser igual la penalización de pasar una imagen de 4 puntos a 3 (pérdida del 25% de la información de la figura) que pasar de 15 puntos a 14 (pérdida del 6'66% de la información de la figura), aunque ambas hayan perdido un único punto. Precisamente, ese es el criterio utilizado para penalizar las imágenes. Si tenemos que convertir una imagen de n puntos en otra de n-x, el máximo parecido que podrá haber entre ellas, una vez igualadas ambas, será el mostrado en la siguiente expresión:

$$penaliz = \frac{100 \cdot (n - x)}{n} \% \qquad (4.3)$$

Además, a esta expresión hay que añadir la posibilidad de penalizar por valores de curvatura, es decir, penalizar más la imagen si el punto eliminado tenía una curvatura muy aguda, y viceversa, penalizar en menor medida los puntos de menor curvatura. Para ello podemos configurar esto con la opción *Penaliz. por curvatura*, cuyo valor por defecto es 5%,

y puede hallarse entre 0% (desactivada) y 10%. El valor que introducimos representa la penalización de un punto cuya curvatura sea del 100%. Si posee menor curvatura, la penalización será inferior. La expresión siguiente nos muestra este hecho:

$$penaliz_{curvat} = \frac{curvatura}{100} \cdot factor_{penaliz} \qquad (4.4)$$

donde *curvatura* indica el valor de curvatura del punto en cuestión y $factor_{penaliz}$ representa el valor introducido en la configuración de la aplicación.

4.2.1.1 Preprocesamiento de la consulta para el modo de *Búsqueda por Imagen*

Si la imagen de entrada ha resultado tener *n* puntos característicos, y teniendo en cuenta los valores del *margen mayor* (m_{may}) y del *margen menor* (m_{men}) establecidos en la configuración, las imágenes de la base de datos (o del grupo de imágenes elegido de la base de datos) que van a ser comparadas con la de entrada, son las incluidas en el siguiente intervalo:

$$[n - m_{men}, n + m_{may}] \qquad (4.5)$$

Denominaremos *imágenes iguales* a la imágenes de la base de datos que posean el mismo número de puntos característicos que la imagen de entrada, es decir, *n*. De la misma forma, definimos *imágenes mayores* a aquellas imágenes pertenecientes al intervalo de la Ecuación 4.5 con un número de puntos característicos mayor que *n*, e *imágenes menores* a aquellas imágenes pertenecientes al intervalo de la Ecuación 4.5 con un número de puntos menor que *n*.

Lógicamente, cuanto menor sea el valor de estos márgenes, menor número de imágenes tendremos en cuenta en la búsqueda. Además, por defecto estos valores están establecidos en 5, que es el valor máximo posible. La razón de esto último es que una imagen que se diferencia de otra en más de 5 puntos característicos no deben parecerse casi nada.

Por tanto, aquellas imágenes que no estén en el intervalo de la Ecuación 4.5 no serán ni siquiera mostradas en los resultados (hemos de deducir que su parecido será del 0%). Las imágenes que sí pertenezcan a ese intervalo serán analizadas por grupos.

El grupo de imágenes mayores y menores con el que se va a comparar la imagen de entrada depende de una serie de opciones. Vamos a analizar las posibilidades existentes en función de que esas opciones estén o no estén activadas:

- *Sin 1-Mejora. Sin Rotación.*

  - *Imágenes mayores o iguales*

    En primer lugar, introducimos las imágenes iguales (*n* puntos) en una tabla temporal. Tras esto, introducimos las imágenes mayores en la misma tabla temporal, pero ahora tenemos que eliminar tantos puntos de cada una de estas imágenes como diferencia de puntos tenga la imagen en cuestión con la imagen de entrada, para que la búsqueda pueda realizarse. Llamemos *p* al número de puntos de una imagen mayor. Tendremos, por tanto, que eliminar *p-n* puntos de la imagen mayor para convertirla en una imagen igual; y así para todas las imágenes mayores implicadas en la búsqueda.

    Los puntos a eliminar son aquellos con el mínimo valor de curvatura; es el criterio propuesto aquí (salvo en el caso de la técnica de 1-Mejora que se explicará más tarde), ya que entendemos que un punto característico de mayor curvatura tiene mayor importancia en la descripción de la forma que un punto de escasa o pequeña curvatura.

    Una vez que disponemos de la imagen de entrada (por la fase anterior) y de las imágenes mayores e iguales adaptadas para la búsqueda en una tabla temporal, podemos construir y ejecutar la consulta pertinente (ver apartado siguiente).

- *Imágenes menores*

En el caso de las imágenes menores, el procedimiento a seguir es distinto, puesto que es la imagen de entrada la que tiene que ver reducido su número de puntos característicos, mientras que las imágenes de la base de datos no tienen que modificarse. Nos vemos obligados, por tanto, a definir m_{men} conjuntos de imágenes que requerirán m_{men} consultas distintas.

Por cada conjunto de imágenes, eliminaremos tantos puntos como sea necesario de la imagen de entrada (que es la que tiene un número mayor de puntos) para igualar el número de puntos con las imágenes menores de ese conjunto en cuestión.

Por tanto, el plan de ejecución es el siguiente: miramos si existen imágenes de n-1 puntos característicos. Si las hay, eliminamos un punto (el de menor curvatura) de la imagen de entrada, calculamos su penalización ($penaliz + penaliz_{curv}$, de las ecuaciones 4.3 y 4.4 respectivamente) y pasamos a la siguiente fase, para que construya y ejecute la consulta correspondiente. Una vez ejecutada la consulta o bien si no existían imágenes de n-1 puntos, repetimos el proceso para las imágenes de n-2 puntos, donde ahora eliminamos dos puntos de la imagen de entrada (dos penalizaciones) para comparar con las imágenes de este conjunto de imágenes de n-2 puntos. Y así continuamos hasta examinar las imágenes de n-m_{men} puntos.

Por tanto, tendremos que realizar como máximo un total de m_{men} consultas a la base de datos para examinar las imágenes menores. No tenemos más remedio si queremos partir de más información a menos (de más puntos característicos a menos), ya que si quisiéramos realizar todo este proceso en una única consulta (caso anterior), tendríamos que añadir puntos a las imágenes menores, pero entonces ¿qué orden y qué valores de curvatura, signo y distancia deberían tener estos puntos? No es posible responder a tal cuestión.

Resumiendo, tendremos *como máximo* que generar y ejecutar un total de 1 + m_{men} consultas a la base de datos (en el caso de no utilizar ni la 1-Mejora ni la Rotación). Lógicamente, si por ejemplo, el conjunto de imágenes menores con n-1 puntos es vacío, entonces esa consulta no se realizará, de ahí que se indique el número de consultas *como máximo*.

- *Sin 1-Mejora. Con Rotación.*

Como el primer punto característico de una imagen es calculado siempre igual, es decir, comenzando desde abajo hacia arriba, si dos imágenes son iguales, ambas tendrán los mismos puntos característicos en el mismo orden y comenzando por el mismo punto. Sin embargo, si por ejemplo una de las dos imágenes es igual que la otra pero rotada 90º, los puntos característicos serán los mismos, también estarán ordenados igual que en la imagen original, pero no comenzará por el mismo punto que la anterior. Pues bien, activar la rotación implica analizar las imágenes comenzando desde todos los posibles puntos característicos existentes, lo que por supuesto implica un número mayor de consultas.

- *Imágenes mayores o iguales*

Realizamos la misma operación descrita para el apartado anterior (sin 1-Mejora y sin Rotación), pero como se verá en la siguiente sección, se lanzan varias consultas cambiando los puntos característicos de inicio para probar la rotación. Por tanto, como existen n puntos posibles de inicio, tendremos que realizar n consultas para buscar las imágenes mayores e iguales.

- *Imágenes menores*

Es también el mismo caso que para el apartado anterior (sin 1-Mejora y sin Rotación). Sin embargo, para cada una de las consultas de los conjuntos de imágenes menores (m_{men} en total) tendremos que probar el punto característico de inicio. Tendremos que realizar, por tanto, un total de m_{men}*n consultas para buscar (teniendo en cuenta la rotación) las imágenes menores. Por tanto, primero se

lanzan las consultas del conjunto con imágenes de n-1 puntos, después imágenes con n-2 puntos, y así sucesivamente hasta n-m_{men} puntos, como se indicó en el caso primero.

Por consiguiente, tendremos que realizar un total de (m_{men}+1)*n consultas para este caso en el que consideremos la rotación.

- *Con 1-Mejora. Sin Rotación.*

  Es necesario indicar en este punto en qué consiste la técnica de 1-Mejora. Solamente es indicada para aquellas imágenes que posean un punto característico más o un punto característico menos que la imagen de entrada, es decir, n+1 ó n-1 puntos. Se trata de analizar qué punto de la imagen que tenga un número mayor de ellos debe eliminarse para maximizar el parecido. Es decir, si antes eliminábamos siempre el punto de mínima curvatura, ahora probamos a eliminar uno a uno cada punto de la imagen y obtener la combinación que supone el máximo parecido.

  Lógicamente, este proceso es muy ineficiente para el caso de imágenes de 2 o más puntos de diferencia que la imagen de entrada, puesto que las combinaciones elevan exponencialmente el número de consultas a realizar. Para el caso concreto de imágenes con una diferencia de 2 puntos con respecto a la imagen de entrada, tendríamos que analizar el parecido que supone eliminar todos los pares de puntos posibles de la imagen (para igualarla en número de puntos a la otra imagen) y ver cuál supone un parecido mayor. De ahí que la técnica solamente se aplique para imágenes con un punto más o un punto menos que la imagen de entrada.

  La Ecuación 4.6 nos muestra el número de combinaciones necesarias para aplicar la técnica de la d-mejora, donde x es el número de puntos de la imagen a comparar, n es el número de puntos de la imagen de entrada y d es $|n-x|$:

$$N^{o}\, combinaciones = \binom{n}{d} \qquad (4.6)$$

n	*x*	Nº combinaciones	*n*	*x*	Nº combinaciones
10	8	45	10	6	210
15	13	105	15	11	1365
20	18	190	20	16	4845
25	23	300	25	21	12650
10	7	120	10	5	252
15	12	455	15	10	3003
20	17	1140	20	15	15504
25	22	2300	25	20	53130

Tabla 4.1: Nº de combinaciones posibles para distintos valores de *n* y *x*

Como podemos apreciar en la Tabla 4.1, el número de combinaciones es demasiado elevado para tener en cuenta la *d*-mejora (salvo la 1-Mejora) en este proceso, ya que arrojaría unos resultados muy pobres.

- *Imágenes mayores o iguales*

Realmente solamente necesitamos una única consulta para comparar las imágenes mayores e iguales de la base de datos, si bien hay que indicar que las imágenes con n+1 puntos no serán calculadas en este apartado.

- *Imágenes menores*

De nuevo, el cálculo es el mismo que en el primero de los casos, salvo que no tenemos en cuenta las imágenes con n-1 puntos característicos, cuyo análisis se realiza un poco más abajo. Por tanto, el número de consultas a generar es una por cada conjunto de imágenes. Como hemos suprimido el conjunto (n-1), tendremos que ejecutar m_{men} - 1 consultas a la base de datos.

- *Imágenes con un punto más y un punto menos que la imagen de entrada*

Estas son las imágenes objetivo de la técnica de 1-Mejora como se ha mencionado más arriba. Veamos los dos casos: *1-Mejora superior*, que implica a las imágenes con un punto característico más que la imagen de entrada y *1-Mejora inferior*, que implica a las imágenes con un punto característico menos que la imagen de entrada.

En el caso de *1-Mejora superior*, a las imágenes con n+1 puntos característicos se les debe suprimir cada uno de los n+1 puntos posibles y analizar su parecido, con una consulta por cada opción. Por tanto tendremos que generar y ejecutar n+1 consultas: elimino el 1er punto y guardo el resultado, elimino el 2º punto, comparo si es mejor que el 1º y, en ese caso, guardo el resultado, y así sucesivamente hasta probar eliminar el último de los n+1 puntos de la imagen mayor.

El caso de la *1-Mejora inferior* es equivalente al de la 1-Mejora superior, con la salvedad de que ahora es la imagen de entrada la que tiene que probar a suprimir, uno a uno, todos los puntos característicos y analizar los resultados. Como la imagen de entrada tiene n puntos característicos (la comparada tiene n-1 puntos), el número de consultas a generar y ejecutar será n.

Por tanto, tenemos un total de $2n+m_{men}+1$ consultas a ejecutar para buscar imágenes en la base de datos utilizando la técnica de 1-Mejora.

4.2.1.2 Preprocesamiento de la Consulta para el Modo de *Búsqueda por Patrón*

En el caso de búsqueda por patrón, la filosofía es distinta. No tratamos de comparar directamente el parecido entre imágenes, sino que buscamos la inclusión de un patrón en todas las imágenes implicadas en la búsqueda, por lo que, cualquier imagen con un número de puntos igual o mayor que el patrón participará en dicha búsqueda, a diferencia de la búsqueda por imágenes que afecta únicamente a imágenes que pertenecen al intervalo de la

Ecuación 4.5. Puede considerarse, por tanto, que un patrón es como una parte de un contorno, de la cual extraemos sus puntos característicos.

En este caso, el número de consultas dependerá de la imagen que tiene más puntos característicos de la base de datos, o en el caso de buscar en un único grupo de imágenes, la imagen con mayor número de puntos característicos de dicho grupo. Si por ejemplo, tenemos una imagen patrón de 4 puntos y la imagen con más puntos característicos implicada en la búsqueda tiene 20, tendremos que probar si existe el patrón en la imagen con 20 combinaciones, es decir, comparando el primer punto del patrón con el primero de cada imagen (y el resto de puntos en orden), el primero del patrón con el segundo de la imagen (y el resto en orden), y así sucesivamente hasta el número total de puntos característicos de cada imagen y teniendo en cuenta la estructura cíclica que supone cada imagen. Además, podemos realizar en una única consulta comparaciones con imágenes con un número de puntos característicos diferentes, gracias a la utilización del operador *módulo* (véase siguiente sección)

4.2.2 Ejecución de la Consulta FSQL

En la fase anterior hemos explicado cuántas y cuáles son las consultas necesarias para realizar determinadas búsquedas, en función de las opciones activadas. Además, hemos explicado que debemos tener el mismo número de puntos característicos en dos imágenes si queremos compararlas, así como los mecanismos para lograr esto.

En esta otra fase queremos mostrar, sin entrar en detalles de implementación, el formato que tiene cada uno de los tipos de consultas que van a utilizarse durante la búsqueda y cómo puede utilizarse el servidor FSQL para comparar las imágenes.

En primer lugar, hay que mencionar que el único comparador difuso que vamos a utilizar va a ser **FEQ** (*Fuzzy EQual*) (véase Sección 3.3.2.2). Este comparador corresponde al conjunto difuso que expresa el concepto “aproximadamente *n*”, donde *n* es el valor de un atributo difuso (*crisp*) (véase un ejemplo en la Figura 4.25).

Por cada punto característico de la imagen de entrada, la consulta incluye 3 comparaciones difusas con *FEQ*, una comparación para la *curvatura*, otra para la *distancia* y otra para el *signo*.

Todas las comparaciones están enlazadas mediante el operador *AND*. La traducción del operador AND de FSQL es el *mínimo* de los parecidos de los atributos indicados, aunque el servidor FSQL permite utilizar otra función como traducción de la conectiva AND, o lo que es lo mismo, elegir otra *t-norma* distinta al *mínimo* ($\wedge$) (véase Definición 3.13). Por tanto, por cada punto característico de la imagen de entrada se obtendrá un parecido final, a partir de los parecidos obtenidos de sus atributos:

$$p_{\mathrm{i}} = min(p_{curvat,\mathrm{i}}, p_{dist,\mathrm{i}}, p_{signo,\mathrm{i}})$$

donde $p_{curvat,\mathrm{i}}$, $p_{dist,\mathrm{i}}$ y $p_{signo,\mathrm{i}}$ representan el parecido del atributo *curvatura*, *distancia* y *signo* respectivamente del punto característico *i* de la imagen de entrada con el punto característico *i* de la imagen de la base de datos. El término p_{i} representa el parecido total del punto característico *i* a partir de los parecidos de cada uno de sus atributos.

Una vez obtenidos los parecidos de cada uno de los puntos característicos de la imagen de entrada, si quisiéramos obtener aquellas imágenes iguales a la entrada, podríamos utilizar el operador *mínimo* para obtener el parecido global de la imagen (el mínimo de los parecidos de todos los puntos característicos de la imagen). Sin embargo, como nos interesa obtener el parecido de todas las imágenes (no solamente las más parecidas), utilizamos la media aritmética (*AVG*) para obtener el parecido global de la imagen, es decir, el parecido final de una imagen será el resultado de obtener la media aritmética de los parecidos de cada uno de sus puntos característicos.

Por otro lado hay que aclarar que, estrictamente hablando, el parecido en lógica difusa se mide en un grado difuso entre 0 y 1. Sin embargo, en este proyecto se ha considerado el parecido final de la imagen multiplicado por 100 para expresarlo en tanto por ciento, donde 0% indica parecido nulo y 100% parecido total.

En general, podemos diferenciar dos grandes grupos, según si realizamos una *búsqueda por imagen* o una *búsqueda por patrón*.

4.2.2.1 Ejecución de la Consulta para el Modo de *Búsqueda por Imagen*

La distancia a tener en cuenta en la consulta de búsqueda por imágenes es un valor de distancia relativa a la longitud total del contorno de cada imagen.

Las opciones *margen curvatura* y *margen distancia* nos permiten definir el grado de parecido entre atributos de los puntos característicos de imágenes distintas en la base de datos difusa, ya que pueden configurarse. Un valor del margen de curvatura (distancia) grande (cercano al 100%) supone considerar más parecidas curvaturas (distancias) distintas, mientras que un margen de curvatura (distancia) pequeño (cercano al 0%) supone considerar menos parecidas las curvaturas (distancias) distintas. Realmente, estamos representando el conjunto difuso "aproximadamente *curv*", donde m_{cur} denota el margen de curvatura y *curv* el valor de curvatura del punto característico en cuestión de la imagen de entrada (es igual para la distancia). Consideramos como puntos parecidos los incluidos en el intervalo de la Ecuación 4.7:

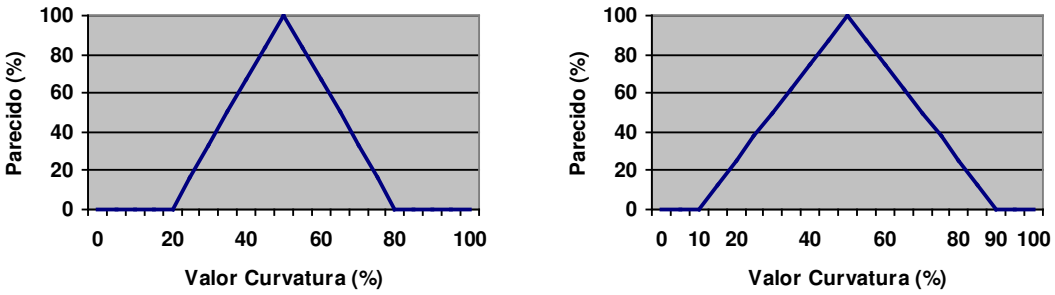
$$[curv - m_{cur} + 1,\ curv + m_{cur} - 1] \qquad (4.7)$$

Sin embargo, todos los valores de curvatura incluidos en el intervalo no tienen el mismo parecido. Para estudiar el parecido entre las imágenes utilizando conjuntos difusos, se definen dos rectas: una desde $curv - m_{cur} + 1$ hasta *curv* y otra desde *curv* hasta $curv + m_{cur} - 1$ (donde *x* es el valor de curvatura del punto a comparar):

$$y = 100 - 100 \cdot \frac{(curv - x)}{m_{cur}}(\%), \text{ para el intervalo } [curv - m_{cur} + 1\ ,\ curv] \qquad (4.8)$$

$$y = 100 - 100 \cdot \frac{(x - curv)}{m_{cur}}(\%), \text{ para el intervalo } [curv\ ,\ curv + m_{cur} - 1] \qquad (4.9)$$

Supongamos un ejemplo, en el que m_{cur} = 30 y *curv* = 50. Observemos, en primer lugar, la gráfica que representa las ecuaciones 4.8 y 4.9 (Figura 4.25(a)) y, por tanto, que representa una definición del conjunto difuso "aproximadamente 50", con un margen de 30.



(a) (b)

Figura 4.25: Conjuntos Difusos que expresan el concepto "aproximadamente 50". (a) *curv* = 50 y m_{cur} = 30. (b) *curv* = 50 y m_{cur} = 40.

Observando en la Figura 4.25(a) dos ejemplos de conjuntos difusos que expresan el concepto "aproximadamente 50", vemos que si el valor de curvatura del punto a comparar es 50, el parecido es del 100%. Si el valor de curvatura fuera 40, como pertenece al intervalo [*curv* - m_{cur} + 1 , *curv*], es decir, [21,50] obtenemos que tiene un parecido (calculado según la Ecuación 4.8) del 66'7%. Si el valor de curvatura fuera 60, como pertenece al intervalo [*curv* , *curv* + m_{cur} - 1], es decir, [50,79] obtenemos que tiene un parecido (calculado según la Ecuación 4.9) del 66'7%. O sea, a medida que nos alejamos del valor de *curv*, tanto por la izquierda (menor curvatura) como por la derecha (mayor curvatura) el parecido disminuye en un 1/m_{cur} %. Por ello, el parecido para el caso de x = 40 y para el caso de x = 60 es el mismo, porque ambas tienen un valor de curvatura 10 unidades menos (más) respectivamente, que el valor de curvatura del punto en cuestión.

Si analizamos el mismo ejemplo, pero con m_{cur} = 40 (Figura 4.25(b)) además de incluir más valores con parecido no nulo, los que ya se parecían en el caso anterior ahora lo harán con mayor porcentaje, de ahí que la Figura 4.25(b) tenga más anchura que la Figura 4.25(a). Aplicando de nuevo la Ecuación 4.8, obtenemos ahora un parecido del 75%.

Por tanto, estos márgenes nos permiten establecer cuán parecidos son los puntos (en cuanto a curvatura o distancia se refiere) de las imágenes. Por supuesto, decir que este estudio realizado sobre la curvatura es exactamente igual que para la distancia, simplemente

analizando el valor del margen de distancia, que también puede ser configurado. Realmente, todos los atributos difusos *crisp* (o del tipo 1) deben ser definidos con ese "margen" que expresa el significado "aproximadamente *n*", donde *n* es un valor del dominio del atributo considerado.

El signo es también un atributo difuso, pero con la siguiente simplificación: si ese punto tiene el mismo signo que la imagen de entrada, el parecido en cuanto al signo es del 100%. Si ese punto tiene distinto signo que la imagen de entrada, el parecido en cuanto al signo es del 50%. Evidentemente, es lógico penalizar más un cambio de signo que un pequeño cambio de curvatura o distancia, que no supone tanta diferencia en la descripción de la forma.

4.2.2.2 Ejecución de la Consulta para el Modo de *Búsqueda por Patrón*

Con la búsqueda por patrón, queremos buscar también una imagen en la base de datos, pero no se trata de comparar ambas directamente, sino buscar que el patrón que representa la imagen de entrada se halle en algún lugar de los contornos de las imágenes de la base de datos que van a ser comparadas. El formato de la consulta es igual que en el caso anterior, con ciertas excepciones:

- Todas las imágenes implicadas que tengan un número de puntos característicos igual o mayor al número de puntos del patrón (imagen de entrada) serán consideradas durante el proceso.
- Tendremos tantos grupos de comparaciones difusas (una comparación por cada atributo difuso de cada punto) en la consulta FSQL como puntos característicos tenga el patrón de búsqueda. Además, en la sección anterior, indicamos que el número de consultas necesarias coincide con la imagen de más puntos característicos de las que participan en la búsqueda.
- A diferencia del caso anterior, los valores de distancia que vamos a utilizar para la comparación entre puntos característicos son valores absolutos, es decir, no relativos a la longitud total del contorno, sino en número de píxeles. Las distancias de los puntos característicos del patrón no pueden ir relativas a su longitud global, puesto que se supone que forma parte de una imagen mayor y, en todo caso, debería ir

relativa a la longitud de cada imagen comparada, hecho muy costoso (desde el punto de vista temporal) de realizar. Por tanto, aunque no podemos contar en este caso con la invariabilidad al tamaño (véase Sección 7.1), podemos realizar la búsqueda en términos absolutos, que sí nos permitirá comprobar si el patrón forma parte de todas las imágenes implicadas en la búsqueda.

- Suprimimos el consultar la distancia del último punto del patrón con el primero, porque no tiene sentido, ya que se supone que este último punto en la imagen comparada estará seguido de otro punto, que no tiene que ser el primero del patrón, sino otro que existe en la imagen comparada y no en el patrón.
- Buscamos el patrón en la imagen comparada, comenzando en ésta última por todos los posibles puntos característicos de inicio. Por ejemplo, si tenemos un patrón de 4 puntos y una imagen comparada de 7 puntos, tendremos que realizar (como se dijo en el apartado anterior) 7 consultas. La primera compara los 4 puntos característicos del patrón de entrada con los 4 primeros puntos de la imagen comparada. La segunda consulta compara los 4 puntos característicos del patrón de entrada con los puntos 2 al 5 de la imagen comparada (ambos inclusive), y así hasta realizar la última consulta, la 7ª, que compara los 4 puntos del patrón de entrada con los puntos 7, 1, 2 y 3 de la imagen comparada, es decir, consideramos esta última imagen de forma cíclica. Para realizar estas operaciones nos valemos del operador *módulo*, de ahí que nos sirva una consulta para todas las imágenes implicadas en la búsqueda, y no hay que realizar una consulta por cada imagen, sino solamente por cada combinación de puntos. En el Capítulo 5 se detallan estos aspectos sobre la implementación realizada.

4.2.3 Presentación de los Resultados

Una vez realizadas las consultas pertinentes, los resultados tendrán que ser mostrados por pantalla para finalizar el proceso de *Comparación de Imágenes*.

Los resultados están compuestos por tuplas, que constan del *nombre de la imagen* y el *parecido* con la imagen de entrada, medido en tanto por ciento. Pero, sin embargo, el parecido obtenido por la consulta no es el parecido final. Debemos restar, para todas las imágenes implicadas en la búsqueda, la penalización calculada para ella, obteniéndose el parecido final:

$$parec_{i,final} = parec_{i,parcial} - penaliz_i\ ,\ \forall i \qquad (4.10)$$

donde $penaliz_i$ se calcula sumando las penalizaciones de las ecuaciones 4.3 y 4.4, es decir, la penalización por diferencia de puntos respecto a la imagen de entrada y la penalización por curvatura, respectivamente.

En el caso de búsqueda por imagen, aquellas imágenes que posean el mismo número de puntos que la imagen de entrada tendrán un valor $penaliz_i = 0$, por lo que el parecido final coincidirá con el parecido parcial. El resto de imágenes tendrán un valor de $penaliz_i$ distinto de cero. En el caso de búsqueda por patrón, como resulta evidente, no utilizamos la penalización, pues todas las imágenes con un número de puntos característicos igual o mayor a la imagen de entrada (patrón) pueden incluir el patrón, independientemente del número de puntos concreto que tengan.

5. DISEÑO E IMPLEMENTACIÓN

En este capítulo vamos a comentar determinados aspectos de la aplicación, en cuanto a su diseño e implementación. En primer lugar indicaremos el lenguaje de programación utilizado en el proyecto y las razones de por qué se ha elegido éste y no otro. Más adelante, mostraremos el diagrama de clases utilizado durante el diseño de la aplicación. Tras esto, reflejaremos el esquema general de la base de datos difusa utilizada. Después, mostraremos el esquema general de una consulta FSQL ejecutada desde la aplicación.

5.1 Lenguaje de Programación Visual C++ 6.0

El lenguaje elegido para la implementación de esta aplicación es *Microsoft Visual C++ 6.0*. Esta elección se basa en gran medida en que es *orientado a objetos*, por lo que nos permite explotar todas las cualidades de este paradigma de programación de *software*:

- *Sencillez*. El problema se plantea en términos del mundo real y no en términos del computador. El *Análisis Orientado a Objetos* permite pasar directamente del dominio del problema al modelo del sistema.
- *Consistencia*. Los atributos y operaciones quedan claramente encapsulados en cada clase correspondiente, reduciendo la distancia entre el punto de vista de los datos y el punto de vista del proceso.
- *Reusabilidad*. En cuanto a la *Programación Orientada a Objetos*, podemos cambiar determinadas clases sin tener que alterar el sistema completamente. Por ejemplo, si deseamos modificar la función de curvatura, únicamente tendremos que modificar la clase *CCurvatura*, mientras respetemos la interfaz establecida.

Además, el entorno de programación es un entorno integrado de desarrollo para aplicaciones *C* y *C++* sobre diversas plataformas. La función que calcula la curvatura, por ejemplo, estaba en formato *C*, por lo que no supuso grandes esfuerzos incorporarla a la aplicación, como parte de una clase, simplemente modificando algunos detalles del módulo.

Otro de los motivos de por qué se ha elegido este lenguaje es por su eficiencia desde el punto de vista temporal. Mientras que lenguajes como *Java* necesitan ser interpretados, *C++* nos permite compilar el archivo ejecutable, con la consiguiente mejora en los tiempos de respuesta de la aplicación, que es un punto considerablemente importante de este proyecto.

Además, como el S.G.B.D. utilizado es *Oracle 8i* (necesario para disponer del *Servidor FSQL*), podemos disponer de una serie de objetos que permiten conectarnos a dicha base de datos (*Oracle Objects for OLE Class Library*) de manera sencillísima, y estos objetos únicamente están disponibles para *Visual C++ 6.0* y *Visual Basic 6.0*. No obstante, también es posible utilizar el sistema de conexión *ODBC* desde éste u otros lenguajes.

Por otra parte, *Visual C++* se caracteriza, al ser un entorno visual, por una gran *sencillez para diseñar formularios* (entorno visual), lo que ha facilitado el diseño amigable de la aplicación, sin la pérdida de *robustez* que, en ocasiones, muchos de estos entornos sufren.

Por último, mencionar que esta aplicación ha sido realizada para ser utilizada con PCs bajo el sistema operativo *Windows*© de *Microsoft*.

5.2 Diagrama de clases

Mostraremos un *diagrama de clases*, en el que las clases estarán dispuestas en jerarquías que comparten una estructura de datos y un comportamiento comunes, y se relacionan con otras clases. Cada clase define los atributos que contiene cada uno de los objetos o instancias y las operaciones que realizan o sufren estos objetos.

Las *clases* serán representadas por un rectángulo, que nos muestra el nombre de la clase, sus atributos y las principales operaciones a realizar sobre ellas. Además, las clases estarán relacionadas entre sí mediante el uso de *asociaciones*, representadas mediante líneas que relacionan a las clases implicadas.

Desde un punto de vista global, la aplicación maneja formas (o figuras) que serán utilizadas por los distintos formularios de los que se compone nuestro programa. No mostraremos todos los atributos y operaciones que posee cada clase en todos los casos, pero sí los más importantes y descriptivos de la funcionalidad de cada una de ellas. La Figura 5.1 muestra el diagrama de clases completo de la aplicación.

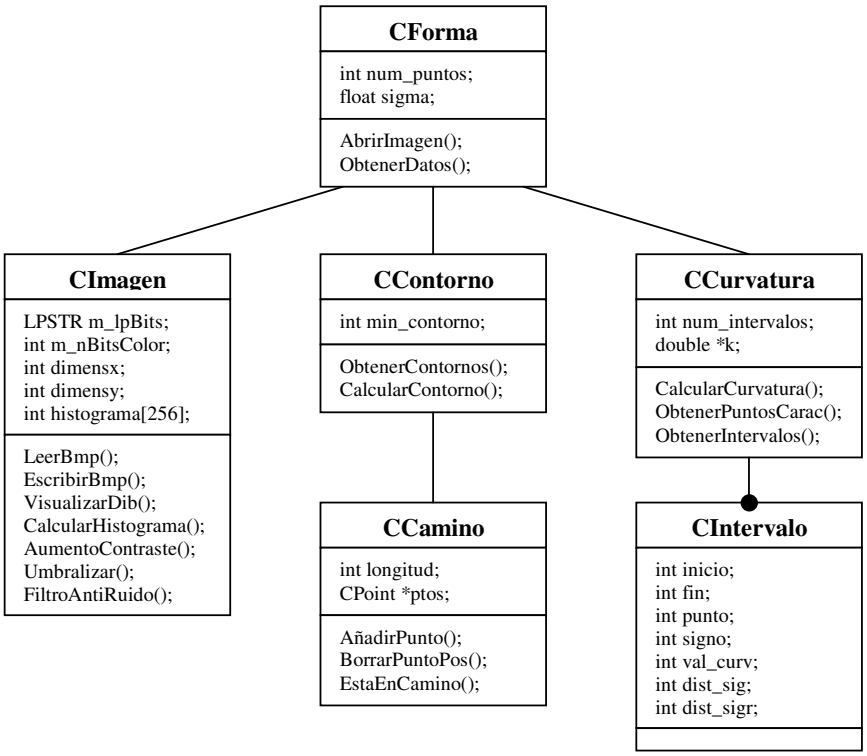


Figura 5.1: Diagrama de Clases de la Aplicación.

5.2.1 Clase *CForma*

Esta es la clase padre de todas las clases de la aplicación. Solamente destacamos dos de sus atributos porque los restantes quedan definidos con las relaciones establecidas con las otras clases del diagrama (Figura 5.1).

Una forma (*CForma*) está compuesta de una imagen (clase *CImagen*) de entrada y de todas las transformaciones que se realizarán sobre ella, como el aumento de contraste o el

proceso de umbralización. Al final del proceso, la imagen resultado es una imagen que muestra el contorno de la figura segmentada.

Además, está compuesta por un vector de puntos (clase *CPoint*, que es una clase propia de *Visual C++*), que representan cada uno de ellos, las coordenadas (*x*,*y*) del punto de la imagen que forma parte del contorno.

Por último, la forma tiene definida una curvatura (clase *CCurvatura*) que nos permite (a través de los umbrales definidos) distinguir los intervalos (clase *CIntervalo*) de los que consta la forma, o lo que es lo mismo, los puntos que caracterizan o describen la forma en cuestión.

En cuanto a sus atributos, *CForma* tiene el número de puntos pertenecientes a la forma (*num_puntos*) y el valor *sigma* con el que se calculó la curvatura de la misma. Este último valor no pertenece a la clase *CCurvatura* porque es indicado por el usuario y no puede ser encapsulado en esta clase.

Las operaciones más importantes definidas en esta clase son *AbrirImagen()*, que nos permite abrir una imagen a partir de un fichero *.bmp*, creando una instancia de la clase *CImagen* para la forma en cuestión. Y la operación *ObtenerDatos()*, que posee como parámetros de entrada a las opciones configurables de la aplicación, que junto con la imagen de entrada obtenida con la operación anterior, nos permiten realizar la fase de *Caracterización de la Imagen* al completo. Es decir, su resultado es obtener los puntos característicos de la forma analizada, usando todas las relaciones con el resto de clases.

5.2.2 Clase *CImagen*

Esta clase nos va a permitir encapsular todos los atributos y todas las operaciones realizadas sobre un mapa de bits (*bitmap*). Es decir, podremos desde acceder al nivel de gris de cada píxel, hasta aplicar el proceso de umbralización, pasando por calcular el histograma de la imagen o visualizarla en un contexto de dispositivo (*DC*) [Ceba98].

En la última fase de la *Caracterización de la Imagen*, dispondremos de una imagen del contorno de la imagen de entrada, que habrá sido obtenida siguiendo todos los pasos adecuados, a partir de la imagen de entrada y sufriendo una serie de transformaciones como si de un proceso industrial se tratara.

En general, el lenguaje *Visual C++* distingue [Ceba98] dos formatos de mapas de bits: mapas de bits dependientes del dispositivo (*DDB*) y mapas de bits independientes del dispositivo (*DIB*). Un DDB es un objeto GDI (*Graphics Device Interface*), mientras que un DIB es un fichero *.bmp*. Nosotros vamos a utilizar mapas de bits del tipo DIB, puesto que nuestra entrada es un archivo de imagen *.bmp*.

Para ello, creamos la clase *CImagen*, que permite acceder a los atributos almacenados en el archivo de imagen, como su dimensión (anchura y altura de la imagen), número de colores de la paleta, o el color de cada uno de los píxeles que componen la imagen. Y, a esto, le añadimos operaciones más complejas como el cálculo de histograma o la umbralización de la imagen.

Los atributos que destacamos de esta clase son los siguientes:

- *m_lpBits*. Representa un puntero que indica el comienzo de los píxeles de la imagen. Utilizando este atributo podremos obtener el nivel de gris de cualquier punto (*GetBmpPixel()*) o actualizar cualquier píxel con cierto nivel de gris (*SetBmpPixel()*).
- *m_nBitsColor*. Representa el número de bits por píxel utilizado en la imagen en cuestión, o lo que es lo mismo, la resolución de la imagen de entrada. Nos servirá para asegurarnos que su valor es 8, es decir, 2^8 (256) colores en la imagen, puesto que solamente admitimos en la aplicación imágenes con esta resolución.
- *dimensx* y *dimensy*. Representan la anchura y altura (respectivamente) de la imagen en cuestión. Estas variables de tipo *int* son utilizadas fundamentalmente como límites en todos los bucles en los que hay que aplicar una máscara o filtro a la imagen.

- *histograma.* Es un vector de 256 enteros, que representa el histograma de la imagen, es decir, la frecuencia de aparición en la misma de cada uno de los 256 niveles de gris posibles en el seno de la imagen.

En cuanto a operaciones, destacamos las siguientes:

- *LeerBmp()* y *EscribirBmp()*. La primera nos permite pasar la información del fichero *.bmp* que se pasa por parámetro a la estructura adecuada en memoria principal. Una vez realizada esta operación, todos los atributos anteriores (salvo *histograma* que se calcula después) tendrán actualizados sus valores correctamente. La segunda realiza la operación inversa, es decir, pasa todos los atributos de la imagen a un nuevo fichero *.bmp* que se indica por parámetro.
- *VisualizarDib()*. Nos permite mostrar el mapa de bits, o mejor dicho, la sección deseada del mapa de bits (para realizar el *scrolling*) en pantalla.
- *CalcularHistograma()*. Nos permite actualizar el atributo *histograma*, realizando una pasada en la imagen y calculando las frecuencias de aparición de cada uno de los posibles niveles de gris en la imagen, quedando estos cálculos almacenados en dicho atributo.
- *AumentoContraste()*, *Umbralizar()* y *FiltroAntiRuido()* desempeñan las funciones ya explicadas en el Capítulo 4 de este documento, realizando transformaciones a la imagen.

5.2.3 Clase *CContorno*

Esta clase trata de encapsular todos los atributos y operaciones relacionadas con la obtención del contorno a partir de la imagen (clase *CImagen*) de entrada. En general, el principal atributo de esta clase es su relación con la clase *CCamino*, que está constituida por una serie de puntos (clase *CPoint*) que constituyen el contorno final.

Como atributo a destacar, mencionamos *min_contorno*, que supone una cota mínima establecida que debe cumplir toda imagen para que el camino obtenido sea considerado como contorno de la forma.

La operación principal de esta clase se denomina *ObtenerContornos()*, que a su vez, llama al procedimiento *CalcularContorno()*. Esta jerarquía está establecida porque una vez obtenido un candidato a contorno (camino) de la forma mediante *CalcularContorno()* (procedimiento recursivo), debemos asegurarnos que realmente constituye un contorno correcto para la forma (entre otras cosas, debe superar el mínimo establecido en el atributo *min_contorno* como se mencionó anteriormente). Si el contorno no es válido, la operación *ObtenerContornos()* elimina el camino obtenido y trata de localizar otro, llamando de nuevo a la función *CalcularContorno()*.

5.2.4 Clase *CCamino*

Esta clase encapsula los atributos y operaciones necesarias para almacenar candidatos a contorno (caminos), llevando una cuenta de los puntos visitados y de la longitud total del camino, parámetros esenciales para la aplicación.

En cuanto a sus atributos, tenemos los siguientes: *ptos*, que es una lista de objetos del tipo *CPoint* que representa el camino calculado hasta el momento. Al final del proceso del cálculo del contorno, si todo ha ido bien, estará constituido por el contorno de la imagen. Y el atributo *longitud*, que representa el número total de puntos de los que consta el contorno, esto es, el perímetro de la forma calculada medido en número de puntos.

Las operaciones más importantes son las típicas de tratamiento de listas enlazadas: *AñadirPunto()*, que añade un nuevo punto al camino estudiado. *BorrarPuntoPos()*, que elimina un punto del camino a través de su posición en él y, por último, *EstaEnCamino()*, que nos indica si el punto que se pasa por parámetro pertenece o no al camino actual. Esta operación es muy importante ya que evita ciclos infinitos en el contorno.

5.2.5 Clase *CCurvatura*

Esta clase va a encapsular todos los atributos y operaciones útiles para el cálculo de la función de curvatura y la extracción de los puntos característicos a partir de ella. El hecho de

que ambos procesos pertenezcan a la misma clase es porque el proceso de extracción de puntos no es más que un análisis exhaustivo de la propia función de curvatura, por lo que se ha considerado interesante y eficiente agrupar ambos aspectos en la misma clase.

Uno de los atributos más importantes en el programa es *k*, que representa la función de curvatura (vector con elementos del tipo *double*), en la que existen tantos elementos como longitud tiene el contorno de la forma. Cada valor del vector representa la curvatura de ese punto del contorno. El otro atributo fundamental para la aplicación es el número de intervalos (*num_intervalos*), o lo que es lo mismo, el número de puntos característicos que se han extraído de la forma. Los puntos extraídos, así como sus propiedades, quedan almacenados en cada instancia de la clase *CIntervalo* que analizaremos a continuación. Por tanto, existirán tantas instancias de la clase *CIntervalo* por cada instancia de la clase *CCurvatura*, como puntos característicos hayan sido extraídos, es decir, *num_intervalos* instancias.

Las tres operaciones esenciales de esta clase son las mostradas a continuación:

- *CalcularCurvatura()*. Realizamos los cálculos pertinentes sobre el vector del contorno, actualizando el valor de *k*, es decir, la curvatura de cada uno de los puntos del contorno. A continuación, llama al procedimiento *ObtenerIntervalos()* y, tras ello, a la operación *ObtenerPuntosCarac()*.
- *ObtenerIntervalos()*. Permite construir todas las instancias de la clase *CIntervalo*, es decir, todos los intervalos obtenidos, utilizando los valores de umbral superior e inferior configurados en la aplicación, para preparar el proceso de nombramiento de puntos característicos, actualizando el valor del atributo *num_intervalos*.
- *ObtenerPuntosCarac()*. Una vez obtenidos los intervalos, nombraremos los puntos característicos y calcularemos sus propiedades.

5.2.6 Clase *CIntervalo*

Esta clase define la estructura de un *Intervalo* en la aplicación. Posee varios atributos, pero ninguna operación (salvo su constructor, cuya función es inicializar los valores de los atributos).

Los atributos a tener en cuenta serán almacenados posteriormente en la base de datos y constituirán, algunos de ellos (en negrita), los *atributos difusos de Tipo 1* (véase Sección 3.3.2.2) que definirán o caracterizarán a la imagen en cuestión:

- *inicio* y *fin* indican la posición del punto del contorno que supone el inicio (fin) del intervalo, respectivamente.
- *punto* representa la posición del punto característico (en el contorno) nombrado para este intervalo.
- ***signo*** representa el tipo de curvatura del punto característico considerado (0 si es convexo y 1 si es cóncavo).
- ***val_curv*** identifica el valor de curvatura del punto característico de este intervalo, es decir, *k[punto]* pero reflejada en tanto por ciento respecto a π.
- ***dist_sig*** identifica la distancia existente entre el punto característico de este intervalo y el punto característico del siguiente, medido en tanto por ciento respecto a la longitud total del contorno.
- ***dist_sigr*** identifica la distancia en términos absolutos (medida en número de píxeles) desde el punto característico de este intervalo y el punto característico del siguiente intervalo.

5.3 Diseño de la Base de Datos Difusa

En esta sección, vamos a mostrar el *Diagrama Entidad/Relación* (Figura 5.2) de la *Base de Datos Difusa* utilizada en esta aplicación. Además, detallaremos cada una de las tablas y de las relaciones entre ellas. Mostramos las entidades temporales con líneas discontinuas y las claves primarias de cada una de las tablas aparecen subrayadas.

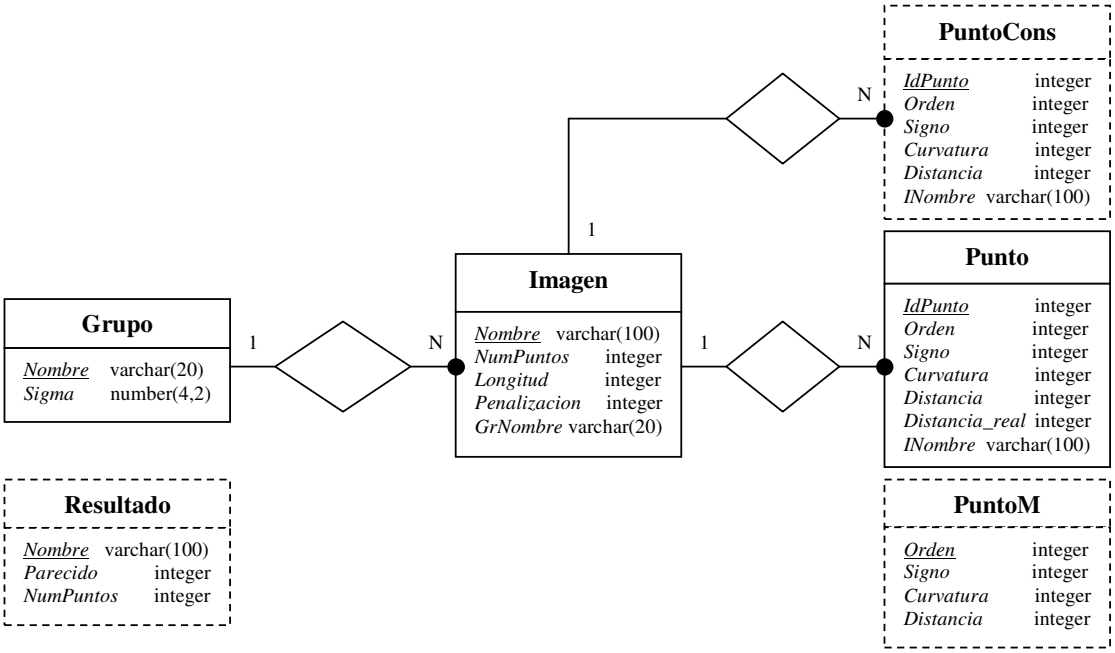


Figura 5.2: Diagrama Entidad/Relación de la Aplicación (en líneas discontinuas se muestran las entidades temporales)

En general, un grupo de imágenes (entidad *Grupo*) está compuesto por un conjunto de imágenes (entidad *Imagen*). Cada imagen, a su vez, está constituida por todos sus puntos característicos (entidad *Punto*). Además, utilizaremos las tablas *PuntoCons* y *PuntoM* para realizar los cálculos temporales con los puntos característicos durante el proceso de búsqueda. Por último, los resultados de la búsqueda obtenidos se almacenan en una entidad aislada (entidad temporal *Resultado*), que nos sirve para realizar cómodamente la ordenación de los resultados obtenidos y mostrarlos en la aplicación.

Los atributos *GrNombre* de la entidad *Imagen* e *INombre* de las entidades *Punto* y *PuntoCons* no deberían mostrarse en este diagrama entidad/relación, puesto que el proceso de migración de claves externas es posterior a esta fase de diseño. Sin embargo, para mostrar toda la información en un único esquema, hemos optado por incluir estos atributos y realizar esta aclaración.

Al traducir este diseño a tablas, obtenemos una tabla por cada entidad. A continuación, vamos a explicar detalladamente cada una de estas tablas.

5.3.1 Tabla *Grupo*

Está constituida por todos los grupos de imágenes definidos en la aplicación. Cada grupo estará constituido por su *Nombre* (20 caracteres como máximo) y el valor de *sigma*, que nos permitirá que todas las imágenes del mismo grupo sean calculadas con el mismo valor de *sigma*, evitando así incongruencias. Cada instancia de la tabla *Imagen* pertenecerá obligatoriamente a una instancia de la tabla *Grupo*, o lo que es lo mismo, no existe ninguna imagen en la base de datos que no pertenezca a un grupo de imágenes concreto.

5.3.2 Tabla *Imagen*

Está constituida por todas las imágenes existentes en la base de datos. Cada imagen está constituida por su *Nombre* (ruta en el que se halla la imagen en cuestión, con 100 caracteres como máximo), *NumPuntos* (número de puntos característicos que posee la imagen), *Longitud* (número de píxeles del contorno), *Penalización*, que es el valor que se restará del parecido calculado en el proceso de búsqueda de una imagen; esta penalización proviene de la diferencia de número de puntos característicos entre la imagen de entrada y la comparada. Por último, el atributo *GrNombre*, que aunque no debe aparecer en el modelo entidad/relación, lo mostramos en la Figura 5.2 para indicar que permite establecer la relación entre la tabla *Imagen* y la tabla *Grupo*.

Una misma imagen (mismo valor del atributo *Nombre*) no puede aparecer dos veces en esta tabla, ni siquiera en grupos de imágenes distintos, pues violaríamos la restricción de integridad de no duplicidad de la base de datos. Con esto tratamos de evitar que pueda introducirse en la base de datos una misma imagen (en cuanto a que es el mismo fichero *.bmp*) pero con distinta información (valor de los atributos de sus puntos) porque se haya calculado con distintos parámetros (por ejemplo, con distinto valor de *sigma*).

Toda imagen pertenece obligatoriamente a un grupo de imágenes, como se ha dicho anteriormente y, además, cada imagen tendrá tantos puntos característicos (tantas instancias de la tabla *Punto*) como indique su atributo *NumPuntos*.

5.3.3 Tabla *Punto*

Esta constituida por los puntos característicos de todas las imágenes de la base de datos. Los atributos son los siguientes:

- *IdPunto*, que representa la clave primaria de esta tabla. Es un valor entero.
- *Orden*, que indica la posición en la secuencia de puntos característicos de la forma que representa el punto en cuestión. Por ejemplo, si un punto característico tiene su atributo *Orden* igual a 1 es que representa el primer punto obtenido en la fase de extracción de puntos característicos. Es, por tanto, un valor entero.
- *Signo*, que representa el tipo de curvatura del punto en cuestión. Es un valor entero, en el que únicamente hay dos valores posibles: 0, que indica curvatura convexa y 1, que indica curvatura cóncava.
- *Curvatura*, que representa el valor de curvatura del punto en cuestión. Es un valor entero y se almacena en tanto por ciento respecto a π (máximo valor posible para la función de cálculo de curvatura). Por ejemplo, una curvatura del 50% representa una curvatura real de $\pi/2$.
- *Distancia*, que representa la distancia al siguiente punto característico desde el punto en cuestión. Es un valor entero y se almacena en tanto por ciento respecto a la longitud de la forma (atributo *Longitud* de la tabla *Imagen*). Por ejemplo, una distancia del 25% supone una distancia real al siguiente punto característico de 25% de la longitud total del contorno de la forma.
- *Distancia_real*, que representa también la distancia al siguiente punto característico desde el punto en cuestión. Es un valor entero y se almacena en valor absoluto, es decir, en número de píxeles entre ambos puntos. Este valor es útil únicamente en el caso de la modalidad de *búsqueda por patrón*.
- *INombre*, que permite establecer la relación con la tabla *Imagen*. Es decir, cada punto característico debe pertenecer a una imagen, que es la indicada por este atributo.

Dos puntos pueden tener el mismo valor en el atributo *Orden*, pero no si pertenecen a la misma imagen.

Los atributos *Signo*, *Curvatura*, *Distancia* y *Distancia_real* son atributos difusos del tipo 1 (*crisp*) (véase Sección 3.3.2.2.). Por tanto, para cada uno de estos atributos tiene que definirse en el servidor FSQL, entre otras cosas, el valor del margen de cada atributo. Esta información se encuentra en la tabla del servidor FSQL denominada *FUZZY_APPROX_MUCH* (*FAM*). Los valores por defecto, es decir, los incluidos en el *script* de instalación de la base de datos, son los siguientes: 2 para *Signo*, 70 para *Curvatura*, 70 para *Distancia* y 70 para *Distancia_real*. Además, los tres últimos pueden ser modificados en la aplicación. Sin embargo, el margen del signo siempre es 2, para lograr un 100% de parecido en puntos con el mismo signo y un 50% de parecido en imágenes con distinto signo.

5.3.4 Tabla *PuntoCons*

Es una tabla temporal encargada de almacenar las imágenes implicadas en la búsqueda, pero adaptándolas para dicho proceso. La adaptación, como se ha explicado en otros capítulos de este documento, consiste en igualar el número de puntos característicos de las imágenes que van a ser comparadas con la imagen de entrada (imagen a buscar), suprimiendo convenientemente algunos de esos puntos (véase Sección 4.2.1).

En determinados casos, para comparar una imagen de entrada con otra imagen en cuestión de la base de datos, tendremos que suprimir algún punto de esta última para poder realizar la comparación adecuadamente, pero en otros casos, no tendremos que eliminar ninguno. Debido a este hecho, todos estos cálculos se realizan sobre una tabla temporal, y así no perder los datos de las imágenes de la base de datos para posteriores consultas. Además, la idea de utilizar una *Vista* en vez de una tabla temporal fue desechada, ya que eliminar un punto de una imagen no es tan simple como ocultar dicho punto en la vista, ya que hay que modificar las distancias entre puntos cuando eliminamos uno de ellos.

La tabla *PuntoCons* posee los mismos atributos que la tabla *Punto*, salvo *Distancia_real*. La razón es que este atributo se utiliza en la búsqueda por patrón y, para este modo de búsqueda, no es necesario adaptar las imágenes para ser comparadas, por lo que los valores (en términos absolutos) de las distancias entre puntos no son necesarios.

Una vez utilizada en cada fase de la búsqueda por imagen, la tabla temporal *PuntoCons* debe suprimir todas sus tuplas (`DELETE * FROM PuntoCons`) para que pueda volver a emplearse.

5.3.5 Tabla *PuntoM*

Esta tabla temporal es utilizada únicamente cuando buscamos imágenes menores (aquellas que poseen un número de puntos característicos menor que la imagen de entrada) en la base de datos. En este caso, es necesario suprimir uno o más puntos de la imagen de entrada (no de las imágenes de la base de datos) porque es esta la que tiene más puntos característicos. Debido a este hecho, no establecemos ninguna relación con ninguna otra tabla, ya que únicamente nos sirve de estructura para almacenar la imagen de entrada adaptada a cada tipo de búsqueda. Por tanto, no nos servirán los atributos *Distancia_real* (por la misma razón de antes), *INombre* (no tenemos relación con ninguna tabla), ni tampoco *IdPunto*, ya que solamente tendremos en esta tabla los puntos de una misma imagen (la de entrada), sirviéndonos el atributo *Orden* como clave primaria.

Al igual que la tabla temporal anterior, después de cada consulta la tabla *PuntoM* debe ser vaciada (`DELETE * FROM PuntoM`) para poder realizar más consultas posteriormente.

5.3.6 Tabla *Resultado*

Esta tabla nos permite guardar los resultados de una búsqueda para facilitar su ordenación y organización, ya que las consultas SQL están preparadas para esto y, por tanto, nos facilitan enormemente este trabajo.

Como se menciona en la Sección 6.2, la aplicación nos permite obtener los resultados ordenados por cada uno de los campos de la búsqueda (*Nombre*, *Parecido* y *NumPuntos*), tanto de forma *ascendente* como de forma *descendente*, además de mostrar únicamente aquellas imágenes con un parecido igual o superior a un umbral y, por último, mostrar solamente un número determinado de imágenes resultado. Para ello, utilizamos las condiciones *WHERE* y la cláusula *ORDER BY*, que nos permite realizar esta ordenación en

una consulta, obtener los resultados en un objeto y volcarlo sobre la tabla correspondiente en el formulario de la aplicación.

Por último, para evitar mostrar resultados de búsquedas anteriores y provocar una confusión al usuario, una vez que cerremos la ventana de búsqueda o procedamos a realizar otra, eliminaremos todas las tuplas de la tabla *Resultado* (`DELETE * FROM Resultado`).

5.4 Consultas FSQL

La fase clave del proceso de *Comparación de Imágenes* es la de *Ejecución de Consultas FSQL*, mediante las cuales podemos obtener el parecido entre la imagen de entrada (imagen a buscar) y todas las almacenadas en la base de datos o en un grupo de ella, mediante sus puntos característicos.

En esta sección vamos a mostrar la estructura general de una consulta FSQL utilizada en esta aplicación para recuperar las imágenes más parecidas a una dada. Vamos a distinguir dos tipos de consultas: *consultas para el modo de búsqueda por imagen*, en la que habremos necesitado una fase de preprocesamiento en tablas temporales para adaptar las imágenes de la base de datos y la propia imagen de entrada a las condiciones de la búsqueda establecidas por el usuario. En el otro caso, *consultas para el modo de búsqueda por patrón*, no necesitamos una fase previa de adaptación a las imágenes, sino que es la propia consulta FSQL la que es capaz de diferenciar las imágenes válidas para este proceso, y en la que no buscamos la igualdad directa de ambas imágenes, sino la inclusión del patrón (contorno de la imagen de entrada) en alguna parte del contorno de las imágenes a comparar.

5.4.1 Modelo de Consulta FSQL para el Modo Búsqueda por Imagen

En el caso de la modalidad de búsqueda por imagen, partimos de que todas las imágenes implicadas en la búsqueda tienen el mismo número de puntos característicos, para poder realizar la consulta, o lo que es lo mismo, ya tenemos en la tabla *PuntoCons* las imágenes que van a ser comparadas, con su número de puntos característicos adecuado.

En general, el esquema de una consulta FSQL para este modo es el mostrado en la Figura 5.3, en el que $curvat_{pi}$, $dist_{pi}$ y $signo_{pi}$ representan el valor de curvatura, distancia al siguiente punto y signo del punto característico *i*, respectivamente, de la imagen de entrada.

```
SELECT Nombre,AVG(CDEG(*)) * 100 AS Parec
FROM Imagen,PuntoCons
WHERE Nombre = INombre
AND ((Orden = 1 AND Curvatura FEQ curvat_p1 THOLD 0 AND
      Distancia FEQ dist_p1 THOLD 0 AND Signo FEQ signo_p1 THOLD 0) OR
     (Orden = 2 AND Curvatura FEQ curvat_p2 THOLD 0 AND
      Distancia FEQ dist_p2 THOLD 0 AND Signo FEQ signo_p2 THOLD 0) OR
     (Orden = 3 AND Curvatura FEQ curvat_p3 THOLD 0 AND
      Distancia FEQ dist_p3 THOLD 0 AND Signo FEQ signo_p3 THOLD 0) OR
                  ..................................
     (Orden = n AND Curvatura FEQ curvat_pn THOLD 0 AND
      Distancia FEQ dist_pn THOLD 0 AND Signo FEQ signo_pn THOLD 0))
GROUP BY Nombre
ORDER BY Nombre;
```

Figura 5.3: Consulta FSQL de la aplicación en la búsqueda por imagen

La consulta agrupa los resultados por cada imagen (`GROUP BY Nombre`), es decir, obtendremos tantas tuplas como imágenes participen en la búsqueda en cuestión, devolviendo únicamente dos atributos tras la ejecución de dicha consulta FSQL:

- *Nombre*, que indica la ruta completa de la imagen comparada.
- *Parec*, que identifica el parecido de la imagen indicada por el atributo anterior con la imagen de entrada y que es definido como `AVG(CDEG(*)) * 100`.

La función *CDEG(*)* nos devuelve el parecido de cada uno de los puntos característicos de cada imagen, es decir, de cada una de las líneas (`Orden = i AND Curvatura FEQ` $curvat_{pi}$ `THOLD 0 ...`), ya que cada una de éstas comienza por una condición 'crisp' que es `Orden = i`. Sin embargo, a nosotros nos interesa el parecido global del objeto, por lo que realizamos la media aritmética (*AVG*) de los parecidos de cada uno de los puntos característicos y lo multiplicamos por 100 para presentarlo en tanto por ciento.

Cada una de las líneas que representa a un único punto característico supone un valor distinto para la función *CDEG(*)*, que calcula el parecido de la siguiente manera:

- Como la condición `Orden = i` es 'crisp', solamente aquellos puntos de la imagen con ese valor del atributo *Orden* podrán ser comparados, lo cual implica localizar el punto característico en cuestión, pero no arroja ningún valor en el parecido final.
- La condición `Curvatura FEQ curvat`$_{pi}$ `THOLD 0` implica la comparación de las curvaturas de los puntos de la imagen de entrada ($curvat_{pi}$) y de la imagen de la base de datos con la que se va a comparar. El comparador *FEQ* se encarga de calcular el parecido existente entre ambas curvaturas, teniendo en cuenta el valor del *margen de curvatura* (configurable desde la aplicación y con un valor por defecto del 70%). *THOLD* representa el umbral mínimo que debe cumplir el parecido para ser considerado por esta función. Si por ejemplo estableciéramos un valor de *THOLD* igual a 0.3 y el parecido entre ambas curvaturas fuera de 0.2, la imagen no aparecería en el resultado final, mientras que si *THOLD* fuera 0.1, el parecido final sería 0.2 y sí aparecería en el resultado final. Nosotros lo establecemos a 0 porque queremos obtener siempre todos los resultados posibles, no solamente aquellos muy parecidos.
- Igualmente ocurre para la condición `Distancia FEQ dist`$_{pi}$ `THOLD 0` en la que comparamos las distancias de los puntos de la imagen de entrada ($dist_{pi}$) y de la imagen de la base de datos con la que se va a comparar. El comparador *FEQ* se encarga de calcular el parecido existente entre ambas distancias, teniendo en cuenta el valor del *margen de distancia* (configurable desde la aplicación y con un valor por defecto del 70%). *THOLD* sigue siendo 0 por la misma razón de antes.
- Por último, la condición `Signo FEQ signo`$_{pi}$ `THOLD 0` en la que comparamos los signos de los puntos de la imagen de entrada ($signo_{pi}$) y de la imagen de la base de datos con la que se va a comparar. El comparador *FEQ* se encarga de calcular el parecido existente entre ambos signos, teniendo en cuenta el valor del *margen de signo* (no configurable desde la aplicación). *THOLD* sigue siendo 0 por la misma razón de antes.

Nos encontramos con tres atributos difusos (que se refieren al parecido entre los puntos característicos de cada imagen con respecto a curvatura, distancia y signo) relacionados por el operador *AND*. Este operador, a diferencia del operador *AND* de *SQL*, calcula el mínimo valor de los tres (*t-norma del mínimo*). Si por ejemplo, el parecido entre ambos puntos en cuanto a curvatura es 0.7, en cuanto a distancia es 0.3 y en cuanto a signo es 0.5, el parecido final entre ambos puntos característicos será 0.3, ya que es el menor valor de

los tres atributos difusos. Como se indicó en la Sección 4.2.2, esta *t-norma* puede modificarse en el servidor FSQL por otra.

Por tanto, para cada imagen y por cada uno de los puntos característicos indicados en la consulta, tendremos un valor que representa el parecido de cada uno de estos puntos, que tras aplicar la función *AVG* (media aritmética) nos devolverá el parecido medio de todos los puntos de cada imagen, o lo que es lo mismo, el parecido de todas las imágenes implicadas en la búsqueda.

Hay que mencionar que los resultados obtenidos tras la ejecución de esta consulta no tienen en cuenta la penalización (por puntos eliminados) que se calcula en la fase de *Preprocesamiento de la Consulta*. Será la aplicación (y no la base de datos) la que, tras ejecutar esta consulta para todas las imágenes implicadas en la búsqueda, reste del parecido obtenido, la penalización calculada anteriormente y obtenga, de esta manera, el parecido final para cada una de las imágenes de la base de datos.

5.4.2 Modelo de Consulta FSQL para el Modo Búsqueda por Patrón

En el caso de la modalidad de búsqueda por patrón, no necesitamos utilizar las tablas temporales *PuntoM* y *PuntoCons* para poder preprocesar la consulta, ya que simplemente, participarán en ella todas aquellas imágenes de la base de datos (o de un grupo de imágenes en cuestión si lo selecciona el usuario) cuyo número de puntos característicos sea igual o superior al número de puntos de la imagen de entrada (patrón de entrada). Y esto se añade a la consulta simplemente introduciendo la condición `NumPuntos >= maxptos`, donde *maxptos* es el número de puntos característicos de la imagen de la base de datos que posee un número mayor de puntos.

En general, el esquema de una consulta FSQL para el caso de búsqueda por patrón es el mostrado en la Figura 5.4, en el que $curvat_{pi}$, $dist_{pi}$ y $signo_{pi}$ representan el valor de curvatura, distancia al siguiente punto y signo del punto característico *i*, respectivamente, de la imagen de entrada. Además, *n* representa el número de puntos característicos del patrón (imagen de entrada). El término *maxptos* va a ser el que nos indique cuántas de estas

consultas vamos a necesitar ejecutar para poder analizar todas las posibles apariciones del patrón en las imágenes de la base de datos. Hay que considerar que el patrón puede aparecer emparejando su primer punto con cualquiera de los puntos característicos de la imagen comparada.

```
SELECT Nombre,AVG(CDEG(*)) * 100 AS Parec
FROM Imagen,Punto
WHERE Nombre = INombre
AND NumPuntos >= maxptos
AND ((Orden = MOD(0,NumPuntos)+1 AND Curvatura FEQ curvat_p1 THOLD 0 AND
     Distancia_real FEQ dist_p1 THOLD 0 AND Signo FEQ signo_p1 THOLD 0) OR
     (Orden = MOD(1,NumPuntos)+1 AND Curvatura FEQ curvat_p2 THOLD 0 AND
     Distancia_real FEQ dist_p2 THOLD 0 AND Signo FEQ signo_p2 THOLD 0) OR
     ..........................................................
     (Orden = MOD(n-1,NumPuntos)+1 AND Curvatura FEQ curvat_pn THOLD 0
     AND Signo FEQ signo_pn THOLD 0))
GROUP BY Nombre
ORDER BY Nombre;
```

Figura 5.4: Consulta FSQL de la aplicación en la búsqueda por patrón

Como se puede apreciar en la Figura 5.4, la consulta es parecida a la que se utiliza en la búsqueda por imagen, con ciertas diferencias. En primer lugar, la condición `NumPuntos >= maxptos` mencionada antes nos permite indicar qué imágenes van a participar en la búsqueda.

Por otro lado, el valor del atributo *Orden* es utilizado mediante el operador módulo (*MOD*) respecto a *NumPuntos*, es decir, respecto al número total de puntos característicos de cada imagen de la base de datos implicada en la búsqueda, para poder realizar todas las posibles combinaciones.

El ejemplo de la Figura 5.4 muestra la primera de las consultas a ejecutar (hay un total de *maxptos* consultas), en la que probamos los valores 0 a n-1, que corresponden a los puntos (tras aplicar el módulo y sumarle uno) 1 a n, es decir, tratamos de comparar los n puntos del patrón con los n primeros puntos de las imágenes de la base de datos (cada imagen i tendrá m_i puntos, donde $m_i >= n$ para todo i). La segunda consulta comparará los n puntos del patrón con los puntos 2 a (n *mod NumPuntos)*+ 1 de las imágenes a comparar. En imágenes en las que $m_i=n$, significará comparar los puntos 2 al 1 (de forma cíclica). En el resto de imágenes, significará comparar los puntos 2 a n+1. Así continuamos hasta efectuar

maxptos consultas, con lo que habremos probado todos los posibles lugares en los que puede hallarse el patrón de búsqueda en la imagen.

Por ejemplo, supongamos $n = 3$. Vamos a comparar este patrón (de 3 puntos característicos) con las imágenes de la base de datos que tengan 3 ó más puntos. Denominemos x al número de puntos de cualquier imagen de la base de datos, con $x \geq n$, es decir, $x \geq 3$. La primera consulta realiza la siguiente comparación:

$$1 \Leftrightarrow (0 \bmod x) + 1 = 1$$
$$2 \Leftrightarrow (1 \bmod x) + 1 = 2$$
$$3 \Leftrightarrow (2 \bmod x) + 1 = 3$$

donde el lado izquierdo muestra los puntos característicos de la imagen de entrada (patrón) y el lado derecho muestra los puntos característicos de la imagen comparada. Siguiendo con el ejemplo, la segunda consulta sería la siguiente:

$$1 \Leftrightarrow (1 \bmod x) + 1 = 2$$
$$2 \Leftrightarrow (2 \bmod x) + 1 = 3$$
$$3 \Leftrightarrow (3 \bmod x) + 1 = 4$$

suponiendo que $x > 3$. Si $x = 3$, entonces tendríamos:

$$1 \Leftrightarrow (1 \bmod 3) + 1 = 2$$
$$2 \Leftrightarrow (2 \bmod 3) + 1 = 3$$
$$3 \Leftrightarrow (3 \bmod 3) + 1 = 1$$

con lo que estaríamos comparando si el patrón termina (en la imagen comparada) en el primer punto (de forma cíclica). Por último, para concluir este ejemplo, mostramos la tercera consulta (aunque sabemos que existirán *maxptos* consultas):

$$1 \Leftrightarrow (2 \bmod x) + 1 = 3$$
$$2 \Leftrightarrow (3 \bmod x) + 1 = 4$$
$$3 \Leftrightarrow (4 \bmod x) + 1 = 5$$

suponiendo que $x > 4$. Si $x = 3$, entonces tendríamos:

$$1 \Leftrightarrow (2 \bmod 3) + 1 = 3$$
$$2 \Leftrightarrow (3 \bmod 3) + 1 = 1$$

$$3 \Leftrightarrow (4 \bmod 3) + 1 = 2$$

y si x = 4, entonces tendríamos:

$$1 \Leftrightarrow (2 \bmod x) + 1 = 3$$
$$2 \Leftrightarrow (3 \bmod x) + 1 = 4$$
$$3 \Leftrightarrow (4 \bmod x) + 1 = 1$$

Por otro lado, utilizamos el atributo difuso del tipo 1 *Distancia_real*, y no el atributo *Distancia*, puesto que nos interesa conocer el valor real de distancia, ya que las distancias relativas al número de puntos del patrón no podemos compararlas con las distancias relativas de imágenes, ya que éstas tendrán otro número de puntos total (generalmente mayor) y eso no debe influir en el parecido, porque solamente buscamos la inclusión del patrón en la imagen, y no la igualdad completa entre ambas. Así que no tenemos más remedio que utilizar los valores de distancia absolutos (medidos en número de píxeles del contorno). Esto implica que no podemos contar (en esta búsqueda por patrón) con la *invariabilidad al tamaño*, propiedad que era posible gracias a las distancias relativas.

Además, podemos apreciar que la consulta FSQL no incluye, en el último de los puntos característicos analizados, la comparación de la distancia real de éste respecto al primero. La razón es que, si lo hiciéramos, estaríamos diciendo que el último punto del patrón tiene como siguiente punto al primero de dicho patrón en la imagen comparada, cuando de lo que se trata es de comprobar si el patrón aparece en una imagen, no si ambas imágenes son iguales.

6. MANUALES DE INSTALACIÓN Y DE USUARIO

6.1. Manual de Instalación

En esta sección vamos a mostrar los pasos necesarios para poder instalar la aplicación *FuzzyFinder 1.0* y todos los componentes adicionales necesarios para su funcionamiento. Los siguientes apartados mostrarán secuencialmente dichos pasos a seguir para completar la instalación de la aplicación.

6.1.1. Instalación del S.G.B.D. *Oracle 8i* y el Servidor FSQL

Será necesario disponer de una copia del sistema gestor de bases de datos (en adelante, SGBD) *Oracle 8i*. La instalación está compuesta de dos discos compactos: el primero incluye los archivos de instalación y el segundo los archivos de ayuda. Realizaremos la instalación *típica* de *Oracle8i Enterprise Edition 8.1.7.0.0*. Podemos consultar en el libro [Pere02] para más información sobre la misma.

Antes de realizar la instalación del servidor FSQL, será recomendable crear una nueva cuenta de usuario con su contraseña para que nuestra aplicación funcione correctamente (será el propietario de la base de datos del programa). Podemos realizar esto a través de la aplicación *DBA Studio* o directamente a través de *SQL*Plus*. Para este usuario, debemos incluir el rol *CONNECT* y los siguientes privilegios de sistema: *CREATE ANY PROCEDURE*, *CREATE ANY TABLE*, *EXECUTE ANY PROCEDURE*, *SELECT ANY TABLE* y *UNLIMITED TABLESPACE*, todos ellos sin necesidad de activar la opción de administración (opción *With Admin. Option*).

Para instalar el servidor FSQL puede utilizarse el usuario *SYS*, que es un *DBA* que automáticamente crea *Oracle*. Por tanto, debemos distinguir entre propietario del servidor (*SYS*) y propietario de la base de datos. Hay que recordar el nombre y contraseña de éste

último para proporcionársela posteriormente a la instalación de la base de datos y a la aplicación en el menú *Configuración*, opción "*Conexión a Oracle*".

Descomprimimos en una carpeta el archivo *FSQLfiles.zip*, que contiene los ficheros necesarios para la creación del servidor FSQL y la base de datos del programa. Para obtener información sobre la instalación, existe el archivo "*Files.txt*" en la carpeta "*Servidor FSQL*" del CD-ROM que se acompaña a la memoria. En él se reflejan los pasos y los ficheros necesarios para instalar el servidor FSQL. De todas formas, vamos a explicar estos pasos:

1) Entramos como administradores (*SYS*) en *SQL*Plus*. Pulsaremos en el menú desplegable "*Fichero*" y, después, en la opción "*Abrir*"; localizaremos el archivo *FIRSTins.sql* y pulsaremos "*Abrir*". Desde este momento, la ruta por defecto será la de este archivo. Desde la consola introduciremos @*FIRSTins*, que ejecutara el script de instalación de FIRST (tablas, permisos...).

2) A continuación, también como administradores (*SYS*), instalamos el servidor FSQL que se corresponde con el archivo *SFSQLins.sql*. La misma operación, salvo que ya no hace falta buscar este archivo, ya que se encontrará en la misma carpeta que el anterior. Simplemente, introducimos en la consola @*SFSQLins*.

3) A continuación, debemos pulsar "*exit*" y salir del entorno *SQL*Plus*, para volver a entrar en él como usuario no administrador o bien, de manera más sencilla, usando *connect* para conectarnos como otro usuario desde la misma consola. Ahora es el momento de recordar el nombre de usuario y contraseña que creamos anteriormente, ya que debemos introducir estos datos en la ventana inicial de *SQL*Plus*. Instalamos el script de nuestra base de datos: fichero *script.sql* escribiendo @*script*. Tras esto, tecleamos '*exit*' y abandonaremos el entorno *SQL*Plus*.

Si nos encontrásemos con algún error durante la instalación, podemos desinstalar cada uno de los sistemas mediante la ejecución de los ficheros *FIRSTdes.sql*, *SFSQLdes.sql* y *uscript.sql*, aunque el orden debe ser el inverso, primero nuestra base de datos, después el servidor *FSQL* y, por último, *FIRST*, teniendo en cuenta los usuarios apropiados para cada sistema.

Para más información sobre la instalación, consultar el fichero "*Files.txt*", donde se especifican muchos otros detalles sobre cada uno de los ficheros.

6.1.2. Instalación de la Aplicación *FuzzyFinder 1.0*

Una vez instalado el SGBD *Oracle 8i*, el servidor FSQL y la base de datos de nuestro programa, solamente nos queda instalar la aplicación *FuzzyFinder 1.0* para *Windows*. Para este fin, no tendremos más que ejecutar el archivo "*Instalar.exe*" y seguir las instrucciones que nos irá mostrando la aplicación. Por defecto, la instalación nos creará un acceso directo en nuestro escritorio denominado *Ffinder*, mediante el cual podremos ejecutar la aplicación.

Una vez que estemos dentro de *FuzzyFinder*, lo primero que debemos hacer es entrar en la ventana "*Conexión a Oracle*". Introduciremos el nombre de usuario y la contraseña del propietario de la base de datos (que debíamos recordar), y que ya utilizamos en la instalación del servidor FSQL. También tendremos que introducir el nombre de la base de datos (introducido durante la instalación de *Oracle 8i*); si no lo recordamos, podemos verlo fácilmente mediante la aplicación *DBAStudio* de *Oracle*, que nos muestra en la ventana principal (en concreto en su lado izquierdo) el nombre de las bases de datos existentes en este ordenador. Solamente, tendremos que anotar el nombre de la base de datos que hemos utilizado e introducirlo en el campo "*Database*" de la ventana "*Conexión a Oracle*".

Una vez realizados todos estos pasos, la aplicación *FuzzyFinder 1.0* estará lista para ser utilizada. Para obtener información sobre la uso de la aplicación, hemos de consultar la siguiente sección.

6.2. Manual de Usuario

En esta sección vamos a mostrar toda la información relativa al uso de la aplicación *FuzzyFinder 1.0*. Iremos analizando ventana por ventana, indicando las características de

cada una de ellas. Además, también está disponible una ayuda en formato *HTML* [GaDi01] accesible desde el propio menú principal y desde todas y cada una de las ventanas de la aplicación, mediante el botón de Ayuda.

Hay que recordar que en esta sección no se abordan temas como ¿qué es un punto característico? o ¿cómo funciona la técnica de *1-Mejora*? Debemos dirigirnos a otros capítulos de la memoria para obtener esta información. Simplemente, se trata de mostrar a cualquier usuario las posibilidades y utilidades de la aplicación de manera sencilla.

6.2.1. Menú Principal

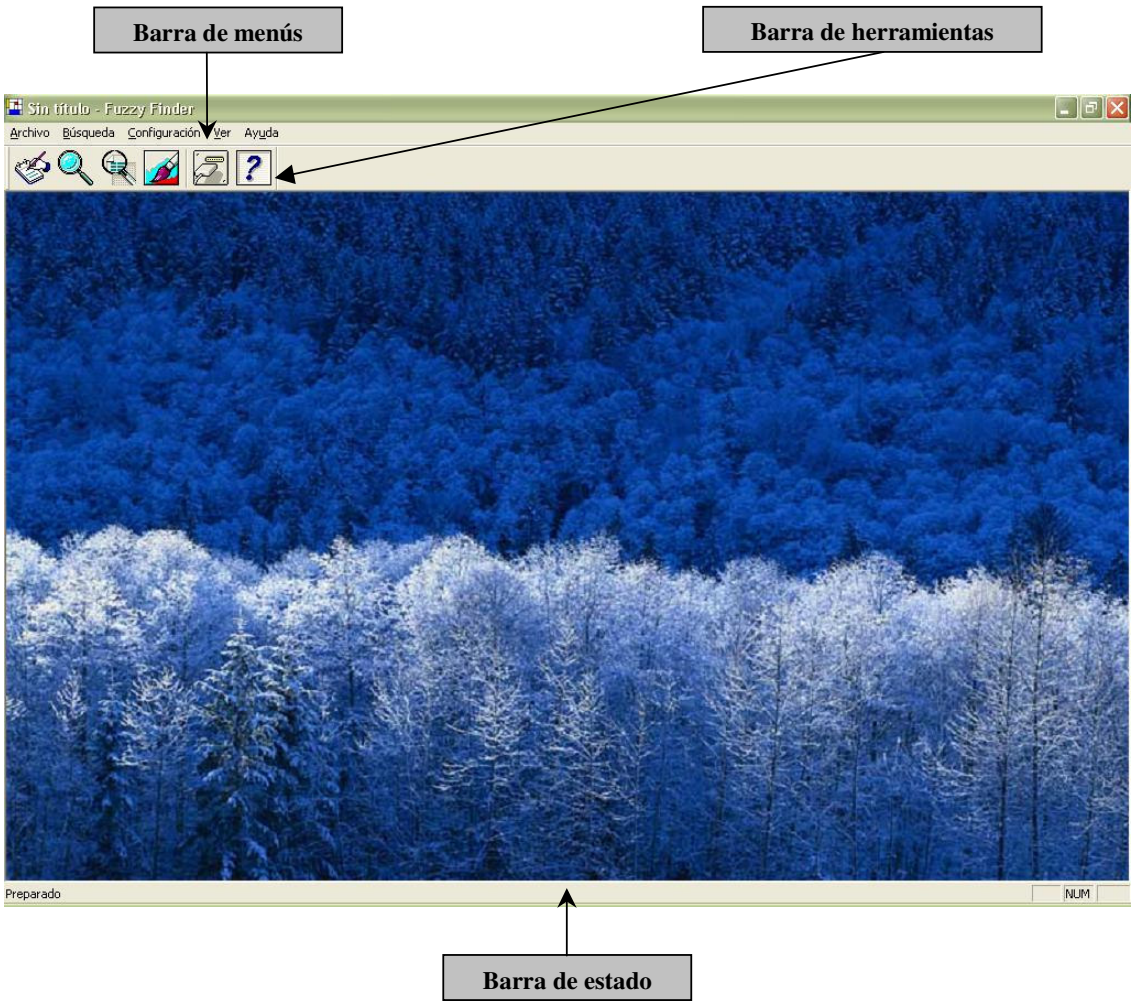


Figura 6.1: Aspecto de la ventana principal de la aplicación *FuzzyFinder 1.0*.

Podemos distinguir tres secciones claramente diferenciadas en el cuerpo de la ventana principal de la aplicación, que están indicadas en la Figura 6.1:

Barra de menús

La barra de menús, con cinco opciones: *Archivo*, *Búsqueda*, *Configuración*, *Ver* y *Ayuda*. Al hacer clic sobre ellas nos aparecen los correspondientes menús desplegables.

- ***Menú Archivo***

  Podemos seleccionar entre dos opciones:

  - *Modificar BD*.

    A través de ella podremos modificar el contenido de la base de datos, añadiendo y eliminando imágenes. Ver *Ventana Modificar Base de Datos*.

  - *Salir*.

    Abandona el programa regresando al entorno del sistema operativo.

- ***Menú Búsqueda***

  Podemos seleccionar entre dos opciones:

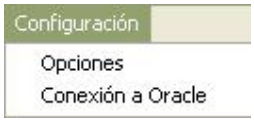
  - *Búsqueda por imágenes*.

    Nos permite especificar la imagen de entrada y determinadas opciones de configuración, así como buscar las imágenes más parecidas de la base de datos con respecto a la de entrada. Ver *Ventana Búsqueda por Imágenes*.

  - *Búsqueda por datos*.

    Nos permite realizar una búsqueda introduciendo los datos de una supuesta imagen sin necesidad de que exista dicha imagen; simplemente introduciendo los valores de sus puntos característicos. Ver *Ventana Búsqueda por Datos*.

- ***Configuración***

  Dos opciones estarán disponibles:

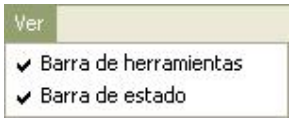
  - *Opciones*.

    Nos permite modificar los parámetros de la aplicación y así optimizar las búsquedas sobre la base de datos. Ver *Ventana Configuración*.

- *Conexión a Oracle*.

  Nos permite establecer los parámetros de conexión con la base de datos *Oracle* a través de un sencillo formulario denominado "*Conexión a la Base de Datos Oracle*".

- ***Ver***



  Podemos seleccionar entre dos opciones:

  - *Barra de Herramientas*.

    Si aparece marcada con el símbolo √, la barra de herramientas permanecerá visible. Si no aparece marcada, la barra de herramientas permanecerá invisible.

  - *Barra de Estado*.

    Si aparece marcada con el símbolo √, la barra de estado permanecerá visible. Si no aparece marcada, la barra de estado permanecerá invisible.

- ***Ayuda***

  Podemos seleccionar entre dos opciones:

  - *Contenido*.

    Nos permite acceder a la ayuda en línea de la aplicación. Aparece en formato *HTML*.

  - *Acerca de...*

    Nos permite acceder a una ventana que muestra los datos de la aplicación.

Barra de herramientas

La barra de herramientas supone una forma directa de acceder a las opciones de los menús con un simple clic de ratón. Cada opción va acompañada de un dibujo identificativo de su funcionalidad. La Figura 6.2 nos muestra el aspecto general de la barra de herramientas de la aplicación.

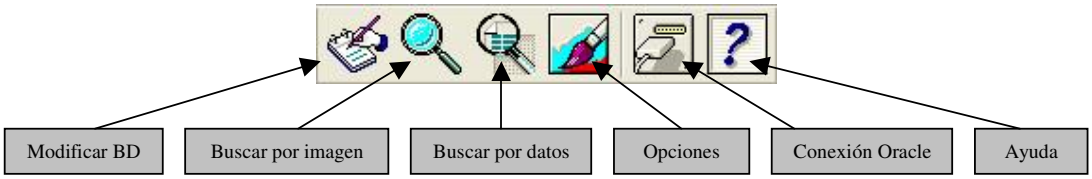


Figura 6.2: Aspecto de la barra de herramientas de la aplicación *FuzzyFinder 1.0*.

Barra de estado

La funcionalidad de la barra de estado, en esta aplicación, se va a limitar a mostrarnos información sobre las opciones que pueden seleccionarse. Por ejemplo, si posicionamos el puntero del ratón sobre el icono "*Modificar BD*" en la barra de herramientas, se informará en la barra de estado sobre la utilidad de esta opción ("*Modificar los elementos de la Base de Datos*"). Tendremos el mismo efecto si posicionamos el puntero del ratón sobre el comando "*Modificar BD*" del menú "*Archivo*". Véase Figura 6.3.

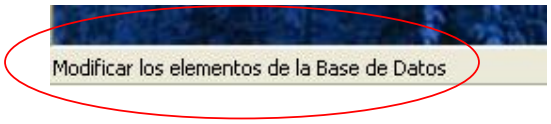


Figura 6.3: Aspecto de la Barra de Estado.

Además de las tres barras indicadas, podemos pulsar el botón de minimizar, maximizar y cerrar que desempeñan las funciones ya conocidas.

6.2.2. Ventana Modificar Base de Datos

Esta ventana nos permite añadir y eliminar grupos de imágenes, así como añadir y eliminar imágenes en dichos grupos. Además, podemos consultar toda la información almacenada en la base de datos, es decir, todas las imágenes y sus propiedades (número total de puntos característicos, longitud del contorno, etc.) y cada uno de los puntos característicos de cada imagen, así como los atributos de todos ellos (curvatura, distancia al siguiente punto característico y el signo de la curvatura).

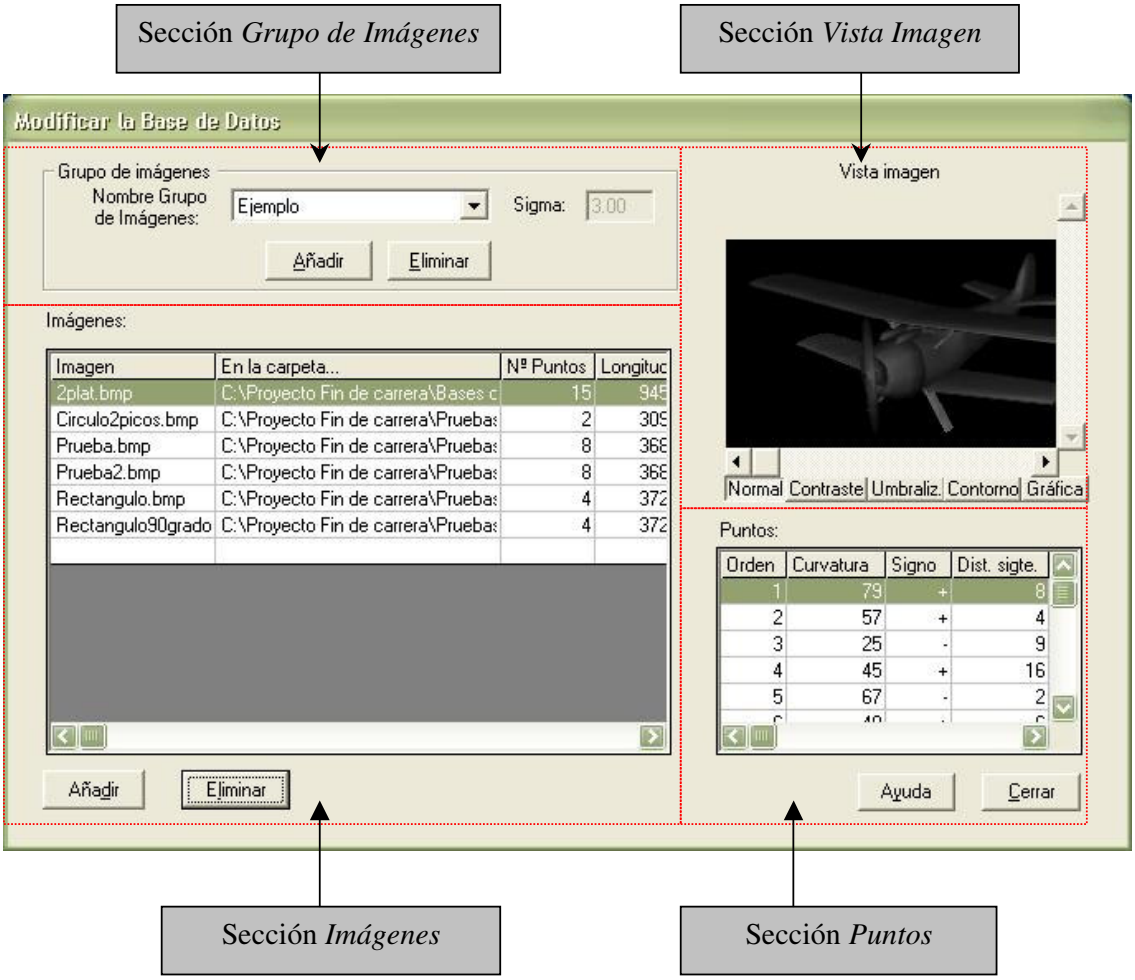


Figura 6.4: Aspecto de la ventana *Modificar Base de Datos* de la aplicación

Podemos pulsar en la lista desplegable titulada "*Nombre Grupo de Imágenes*" para seleccionar el grupo de imágenes que deseemos. Automáticamente, obtendremos el valor de *sigma* correspondiente a dicho grupo. Además, se cargarán todas las imágenes de ese grupo en la rejilla (o tabla) titulada "*Imágenes*" y, a su vez, se cargarán todos los puntos característicos de la primera imagen del grupo en la rejilla titulada "*Puntos*", así como la vista de dicha imagen en la sección "*Vista imagen*".

A medida que vayamos pulsando en cada una de las imágenes de la tabla "*Imágenes*", se irá actualizando la imagen de "*Vista Imagen*" y la rejilla "*Puntos*", cargando los puntos característicos correspondientes a esta imagen seleccionada.

Después de haber mostrado el funcionamiento general de la ventana, podemos analizar este formulario por secciones:

- Sección *Grupo de imágenes* (Figura 6.5)

  Un grupo de imágenes permite englobar un conjunto de imágenes de la base de datos que comparten un mismo *sigma* en el cálculo de su curvatura, esto es, todas aquellas imágenes añadidas sobre un mismo grupo serán calculadas con el mismo valor de *sigma*, que no podrá ser modificado en este formulario. Si lo permitiésemos, deberíamos, para mantener la congruencia, volver a añadir todas las imágenes de ese grupo con el nuevo valor de *sigma*, con el coste temporal que ello conlleva.

Figura 6.5: Aspecto de la Sección *Grupo de Imágenes*.

Además del funcionamiento explicado anteriormente, nos encontramos con dos botones en esta sección:

- Botón "*Añadir*" (grupo de imágenes)

  Al pulsar sobre este botón, se mostrará la ventana "*Añadir Grupo de Imágenes*" que se explica posteriormente en la Sección 6.2.7. Su función es clara: crear un nuevo grupo para albergar imágenes.
- Botón "*Eliminar*" (grupo de imágenes)

  Al pulsar sobre este botón, el grupo de imágenes será eliminado de la base de datos, así como todas las imágenes de dicho grupo.

- Sección *Imágenes*

  Esta sección esta constituida por una tabla con información de todas las imágenes incluidas en el grupo de imágenes seleccionado mediante la lista desplegable justo encima.

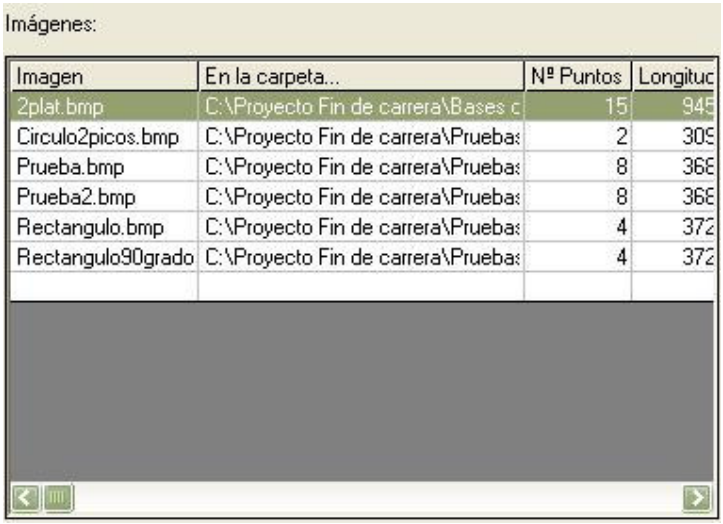


Figura 6.6: Aspecto de la rejilla (o tabla) *Imágenes*

En la rejilla de la Figura 6.6 observamos todas las imágenes que han sido añadidas a la base de datos. Podemos observar cuatro columnas en la tabla:

- La columna "*Imagen*" muestra el nombre de la imagen añadida.
- La columna "*En la carpeta...*" muestra el directorio (carpeta) en el que se halla la imagen. Si concatenamos esta columna con la anterior, obtendremos la dirección completa de la imagen.
- La columna "*Nº Puntos*" muestra el número de puntos característicos que se obtuvieron tras aplicar el proceso de caracterización sobre el contorno del objeto de la imagen.
- La columna "*Longitud*" muestra la longitud en número de píxeles que tiene el contorno, es decir, el perímetro de la forma considerada.

Además, las columnas de esta tabla pueden ser modificadas en su anchura por si la que existe por defecto para cada una no es suficiente para mostrar sus datos al completo.

Debajo de la tabla "*Imágenes*" tenemos dos botones ("*Añadir*" y "*Eliminar*"). Sus funciones son muy claras. El primero de ellos nos permite añadir una imagen a la base de datos. Tras su pulsación, nos aparecerá una ventana (Figura 6.7) que nos pide

que busquemos el archivo *.bmp* de la imagen que deseamos añadir. Pulsaremos abrir y se procederá a la adición de dicha imagen.

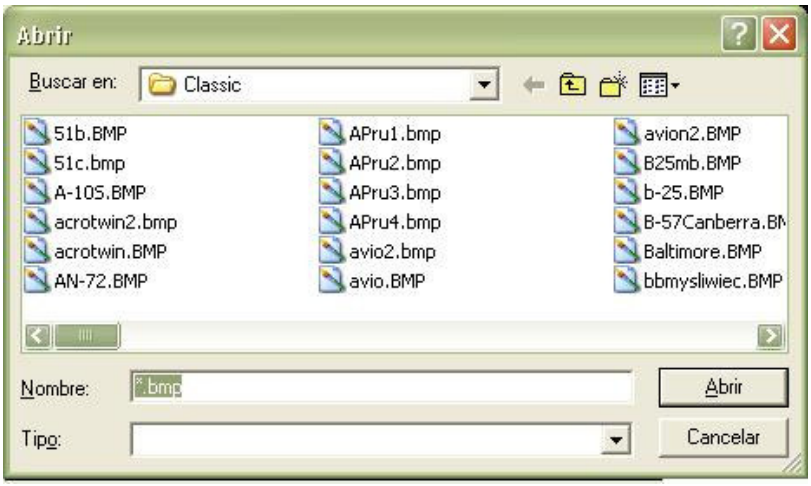


Figura 6.7: Aspecto de la ventana *Abrir* de la aplicación

Si la imagen es correcta, aparecerá un mensaje informativo indicándonos que todo concluyó satisfactoriamente (Figura 6.8).

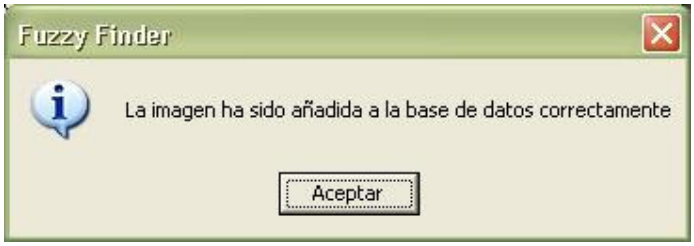


Figura 6.8: Aspecto de la ventana que concluye la adición de la imagen.

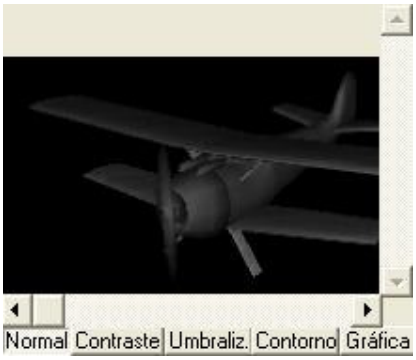
Añadir una imagen supone que se realizará sobre ella una serie de transformaciones y la obtención final de los puntos característicos, que son realmente los valores que almacenaremos en la base de datos, no la imagen propiamente dicha. Además, durante este proceso pueden sucederse algunos errores (el fichero no tiene el formato *.bmp*, el número de puntos característicos es demasiado grande, etc). La Sección 6.2.9 amplia esta información.

El segundo botón es "*Eliminar*", que elimina de la base de datos la imagen seleccionada en la tabla "*Imágenes*". Esta opción no borra el fichero de la imagen, sino

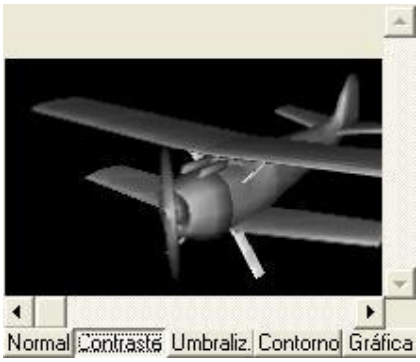
toda la información sobre la misma que existe en dicha base de datos. También aquí se mostrará un mensaje preguntándonos si estamos seguros de realizar la operación.

- Sección *Vista Imagen*

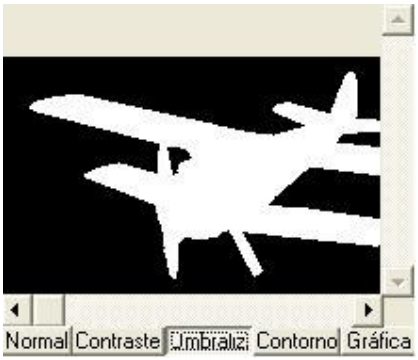
Esta sección está constituida por la vista de la imagen y cinco botones titulados *Normal*, *Contraste*, *Umbraliz.*, *Contorno* y *Gráfica*. La vista dispone de dos barras de desplazamiento que permiten navegar por toda la extensión de la imagen y poder apreciar, así, todos y cada uno de los detalles importantes.



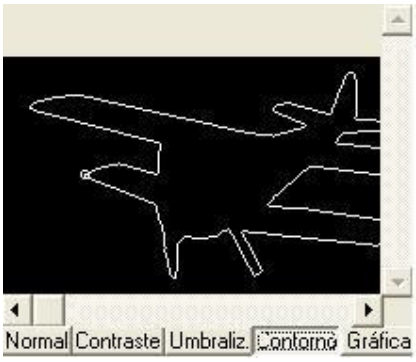
Si pulsamos el botón "*Normal*", la imagen que se mostrará en la vista será la original, mediante la cual se calculó todo el proceso, es decir, la imagen indicada en la tabla "*Imágenes*" a través de las columnas "*En la Carpeta...*" más "*Imagen*", suponiendo que no haya sido eliminada desde fuera del programa, con posterioridad a su adición.



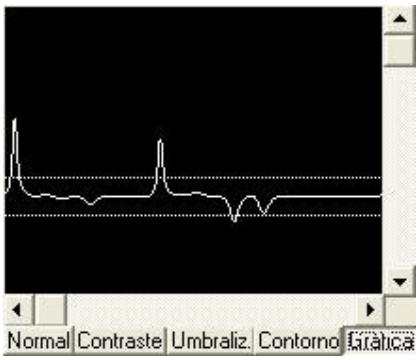
Si pulsamos el botón "*Contraste*", la imagen que se mostrará en la vista será el resultado de aplicarle a la original el proceso de *aumento de contraste*. Si la imagen de entrada es muy oscura, este proceso la aclarará, y viceversa, si la imagen es muy clara, la oscurecerá.



Si pulsamos el botón "*Umbralizar*", la imagen que se mostrará en la vista será el resultado de aplicarle a la imagen contrastada el proceso de umbralización, es decir, el objeto con color blanco (255) y el fondo de color negro (0).



Si pulsamos el botón "*Contorno*", la imagen que se mostrará en la vista será el resultado de aplicarle a la imagen umbralizada el proceso de extracción de contorno. El círculo que aparece en la parte izquierda de la imagen nos indica dónde se localiza el punto característico seleccionado en la tabla "*Puntos*".



Si pulsamos el botón "*Gráfica*", la imagen que se mostrará en la vista será la gráfica resultado de aplicarle a la imagen del contorno el proceso de obtención de la curvatura de dicho contorno. Las líneas horizontales discontinuas indican los umbrales superior e inferior.

Es necesario indicar que las vistas mostradas anteriormente, salvo la etiquetada como "*Normal*", se almacenan en un directorio especial (*Directorio de Vistas*) que puede ser modificado en la ventana "*Configuración*" de la Sección 6.2.5. Estos ficheros de imagen son generados por la aplicación para mostrar los resultados obtenidos en distintos puntos del proceso y no deben ser alterados; la propia aplicación se encarga de gestionarlos.

Si alguna de las vistas no está disponible (por ejemplo ha sido borrada desde fuera del programa, o bien no ha sido almacenada porque así se ha indicado en la ventana de *Configuración*) se mostrará una imagen especial que indica este hecho:

En el ejemplo de la izquierda (Figura 6.9), se ha seleccionado la vista "*Contraste*" de una imagen añadida a la base de datos, pero no fue guardada cuando se calculó y por eso, no puede ser dibujada. Por ello, se muestra esta imagen alternativa indicando tal hecho.

Figura 6.9: Vista NO Disponible

- Sección *Puntos*

Esta sección incluye una tabla con los puntos característicos obtenidos de la imagen seleccionada en la tabla "*Imágenes*".

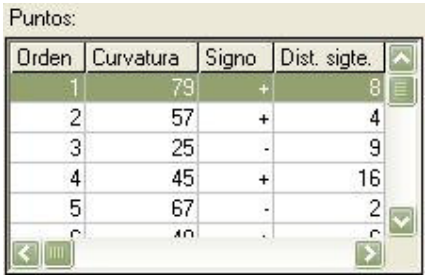


Figura 6.10: Aspecto de la tabla *Puntos*

Las cuatro columnas existentes en esta tabla (ver Figura 6.10) son las siguientes:

- La columna "*Orden*" indica el orden en el que se han hallado los puntos característicos de esa imagen. Habrá tantas filas en esta rejilla como número de puntos en la tabla "*Imágenes*".
- La columna "*Curvatura*" indica el valor de curvatura, en tanto por ciento respecto a π, del punto característico en cuestión.
- La columna "*Signo*" indica el signo de la función de curvatura para el punto característico en cuestión. Si es positivo estamos hablando de una curvatura del tipo convexo en ese punto. Si es negativo estaremos ante una curvatura del tipo cóncavo.
- La columna "*Dist. Sigte*" indica el valor de distancia de un punto característico respecto al siguiente. En el caso del último punto, será la distancia respecto al primero (de forma cíclica). El valor se muestra en tanto por ciento respecto a la longitud total del contorno (valor que puede consultarse en la tabla "*Imágenes*", en la columna "*Longitud*").
- La columna "*Dist. Real*" (que no se aprecia en la Figura 6.10) nos indica, al igual que la columna anterior, la distancia de un punto característico al siguiente, pero en valores reales (número de píxeles del contorno).

Por último, existen dos botones más en el formulario:

- Botón "*Ayuda*"

  Nos permite abrir la ayuda en línea, directamente en la página *HTML* "*Formulario Modificar Base de Datos*".

- Botón "*Cerrar*"

  Nos permite cerrar la ventana y volver al menú principal.

6.2.3. Ventana Búsqueda por Imágenes.

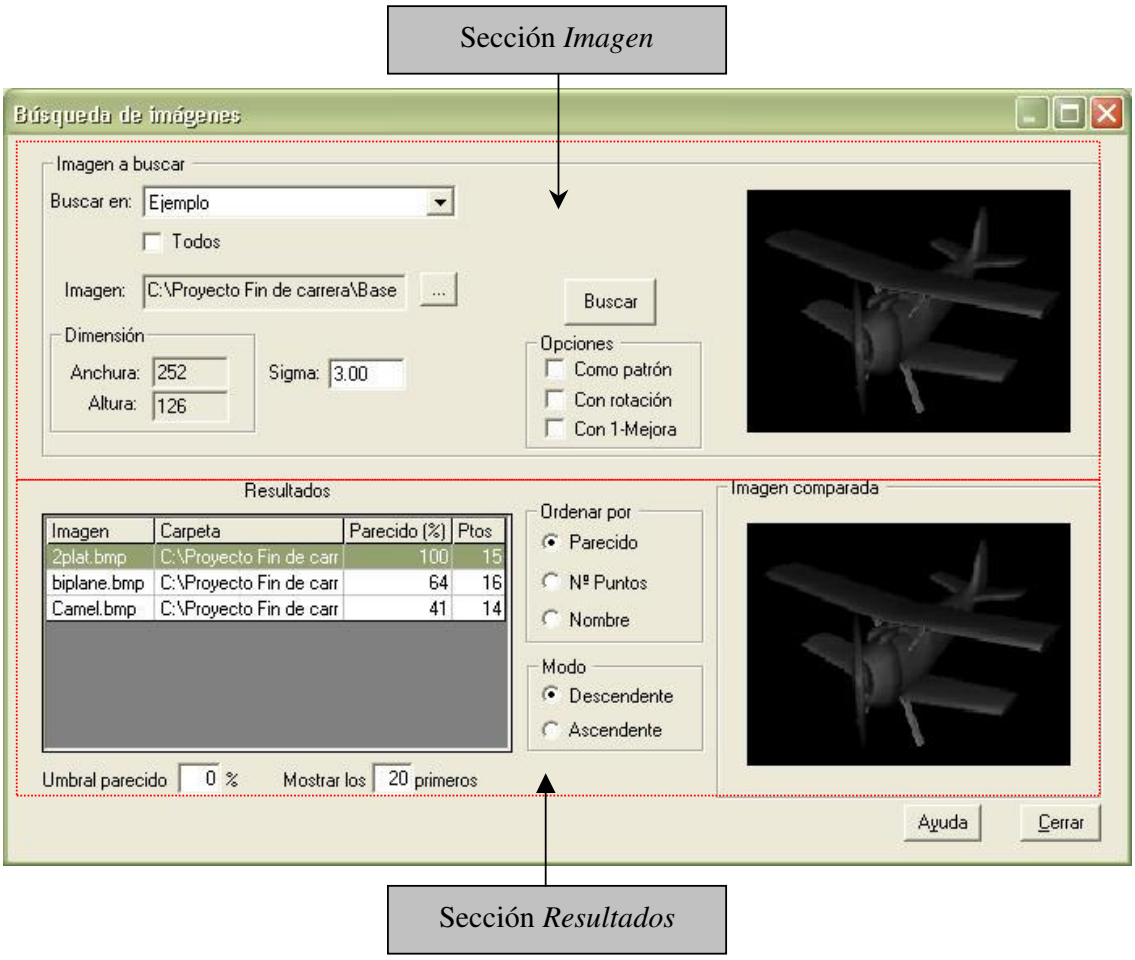


Figura 6.11: Aspecto de la ventana *Búsqueda de Imágenes*

Mediante esta ventana (Figura 6.11) podemos realizar una búsqueda de imágenes en la base de datos a través de una imagen de entrada que podemos elegir.

Analicemos el problema por secciones. Primero actuaremos sobre la sección "*Imagen*" en la que seleccionaremos la imagen de entrada y las opciones de búsqueda deseadas. Después, pulsaremos el botón "*Buscar*" para comenzar el proceso de búsqueda. Por último, modificaremos los valores de la sección "*Resultados*" para organizar los resultados obtenidos.

- Sección *Imagen*

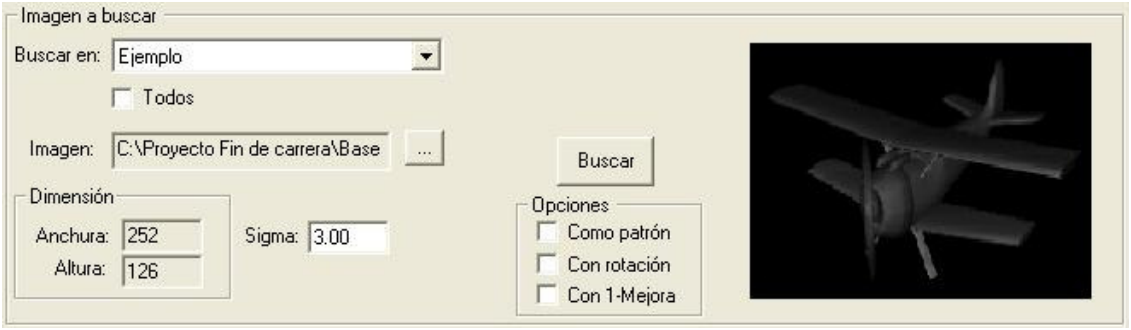


Figura 6.12: Aspecto de la sección *Imagen* de la ventana *Búsqueda de Imágenes*.

En primer lugar, podemos seleccionar el grupo de imágenes sobre el que deseamos realizar la búsqueda, haciendo clic directamente sobre la lista desplegable. Inmediatamente a este hecho, se mostrará en la caja de texto titulada "*Sigma*" el valor de *sigma* correspondiente a ese grupo de imágenes. Este valor puede alterarse, pero no suele ser conveniente, teniendo en cuenta que todas las imágenes de un mismo grupo han sido calculadas con un mismo valor de *sigma* y, entonces, podríamos obtener incongruencias en los resultados.

Si deseamos buscar en todos los grupos de imágenes, marcaremos la casilla titulada "*Todos*". En este caso, el valor de *sigma* se establecerá en *3.00*, que es el valor por defecto.

En segundo lugar, procedemos a establecer la imagen de entrada. Pulsamos el botón titulado "..." y nos aparece una ventana que nos pide que le introduzcamos la imagen a

buscar. Esta ventana tiene el aspecto de la Figura 6.7, que también se utilizó en el formulario "*Modificar Base de Datos*". Una vez seleccionada la imagen, pulsamos el botón "*Abrir*" de la ventana. Automáticamente volveremos al formulario principal, donde nos encontraremos con los siguientes cambios:

- La caja de texto titulada "*Imagen*" contendrá el nombre completo de la imagen seleccionada.
- Las cajas de texto tituladas "*Dimensión*" contendrán las dimensiones (anchura y altura, en número de píxeles) de la imagen seleccionada (Ver Figura 6.12).
- Aparecerá una vista de la imagen seleccionada en la parte derecha de esta sección, ajustada a un recuadro predefinido, por lo que no es raro que la imagen aparezca alargada o estrechada por alguno de sus lados.

El tercer paso es configurar las opciones de búsqueda. Podemos jugar con tres parámetros y todas las combinaciones permitidas:

- *Como patrón*

  Si la casilla está marcada, activaremos el *modo de búsqueda por patrón*, mediante el cual buscaremos el patrón (imagen de entrada) en cada una de las imágenes implicadas en la búsqueda, es decir, no una comparación directa sino buscando que en la imagen comparada aparezca, en alguna sección de la misma, el patrón de entrada.
- *Con rotación*

  Si la casilla está marcada, activaremos el *modo de búsqueda con rotación*, mediante el cual se tendrá en cuenta que las imágenes de la base de datos pueden hallarse rotadas con respecto a la imagen de entrada.
- *Con 1-Mejora*

  Si la casilla está marcada, activaremos el *modo de búsqueda 1-Mejora*, mediante el cual obtendremos una optimización para la búsqueda de las imágenes de la base de datos que tengan un punto característico más y uno menos que la imagen de entrada, eliminando el punto que maximice el parecido final (probamos a eliminar todos los puntos posibles).

Obsérvese que las casillas "*modo de búsqueda con rotación*" y "*modo de búsqueda 1-Mejora*" no pueden seleccionarse conjuntamente. Por otra parte, si seleccionamos el *modo de búsqueda por patrón*, se ignorarán el estado del resto de las casillas.

Tras haber configurado la imagen de entrada y las opciones de búsqueda no tendremos más que pulsar el botón titulado "*Buscar*" para comenzar el proceso. Observaremos unas barras de progreso que nos irán indicando la duración del mismo. Además, el botón "*Buscar*" se habrá convertido en "*Cancelar*". Pulsando sobre él, se cancelará la búsqueda y el botón retomará su aspecto inicial.

- Sección *Resultados*

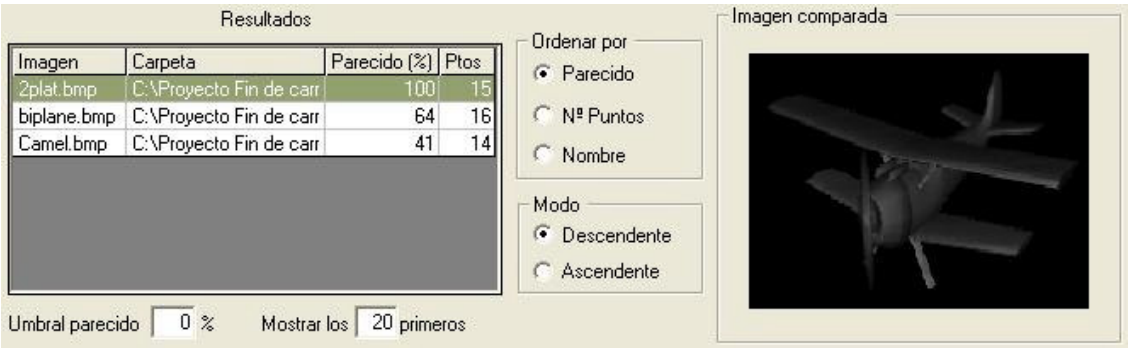


Figura 6.13: Aspecto de la sección *Resultados* de la ventana *Búsqueda de Imágenes*.

Una vez realizada la búsqueda, esta sección tiene como objetivo mostrar los resultados de forma ordenada.

En primer lugar, se aprecia una tabla o rejilla titulada "*Resultados*" con las imágenes de la base de datos obtenidas tras el proceso de búsqueda. Esta tabla se compone de cuatro columnas:

- La columna "*Imagen*" nos muestra el nombre de la imagen encontrada.
- La columna "*Carpeta*" nos muestra el nombre de la carpeta o directorio en el que se encuentra la imagen original (la ruta para encontrar la imagen).
- La columna "*Parecido*" nos muestra, en tanto por ciento, el parecido de la imagen de entrada con la imagen en cuestión: 100% indica el máximo parecido y 0% indica parecido nulo.

- La columna "*Ptos*" nos muestra el número de puntos característicos de la imagen en cuestión.

Si pulsamos en cada una de las imágenes resultado, podremos ver en el lado derecho de esta sección, una vista de la imagen seleccionada. Al igual que en todos los formularios donde se muestran vistas de imágenes, si alguna de las imágenes resultado no se encontrara en la ruta indicada, se mostraría la imagen de la Figura 6.9.

A continuación, analizamos un grupo de botones de opción denominado "*Ordenar por*". Podemos elegir tres opciones: "*Parecido*", "*Nº Puntos*" y "*Nombre*". Su función es la de ordenar los resultados obtenidos por el valor de la columna "*Parecido*", "*Ptos*" e "*Imagen*" respectivamente.

Justo debajo de este grupo de botones, existe otro denominado "*Modo*", con dos opciones: "*Descendente*" y "*Ascendente*". Si está seleccionado el primero de ellos, los resultados se ordenarán de mayor a menor y viceversa, si está seleccionado el segundo, los resultados se ordenarán de menor a mayor, siempre teniendo en cuenta el criterio de ordenación establecido en el grupo de botones "*Ordenar por*". Esta ordenación será numérica para los casos de las columnas "*Parecido*" y "*Ptos*" y, alfanumérica para los casos de las columnas "*Imagen*".

Para acabar esta sección, comentar dos cajas de texto situadas justo debajo de la tabla "*Resultados*" (Figura 6.13). La primera de ellas (de izquierda a derecha) se denomina "*Umbral Parecido*". En ella debe introducirse un número entre 0 y 100, que indica el valor de parecido mínimo que debe cumplir cada imagen resultado para ser mostrada en la rejilla. Si por ejemplo, establecemos este umbral en 100, sólo aparecerán las imágenes idénticas a la de entrada en la rejilla. Por el contrario, si lo establecemos en cero, mostraremos todas las imágenes implicadas en la búsqueda.

La segunda de ellas se denomina "*Mostrar los ... primeros*". En ella debe introducirse un número mayor que cero. Indica el número máximo de imágenes que se mostrarán en la rejilla, en función del criterio de ordenación seleccionado en "*Ordenar por*" y el modo de ordenación seleccionado en "*Modo*".

Por último, los botones de "*Ayuda*" y "*Cerrar*" de la parte inferior de la ventana:

- Botón "*Ayuda*"

  Nos permite abrir la ayuda en línea, directamente en la página *HTML* "*Formulario Búsqueda por Imágenes*".

- Botón "*Cerrar*"

  Vuelve al menú principal.

6.2.4. Ventana Búsqueda por Datos.

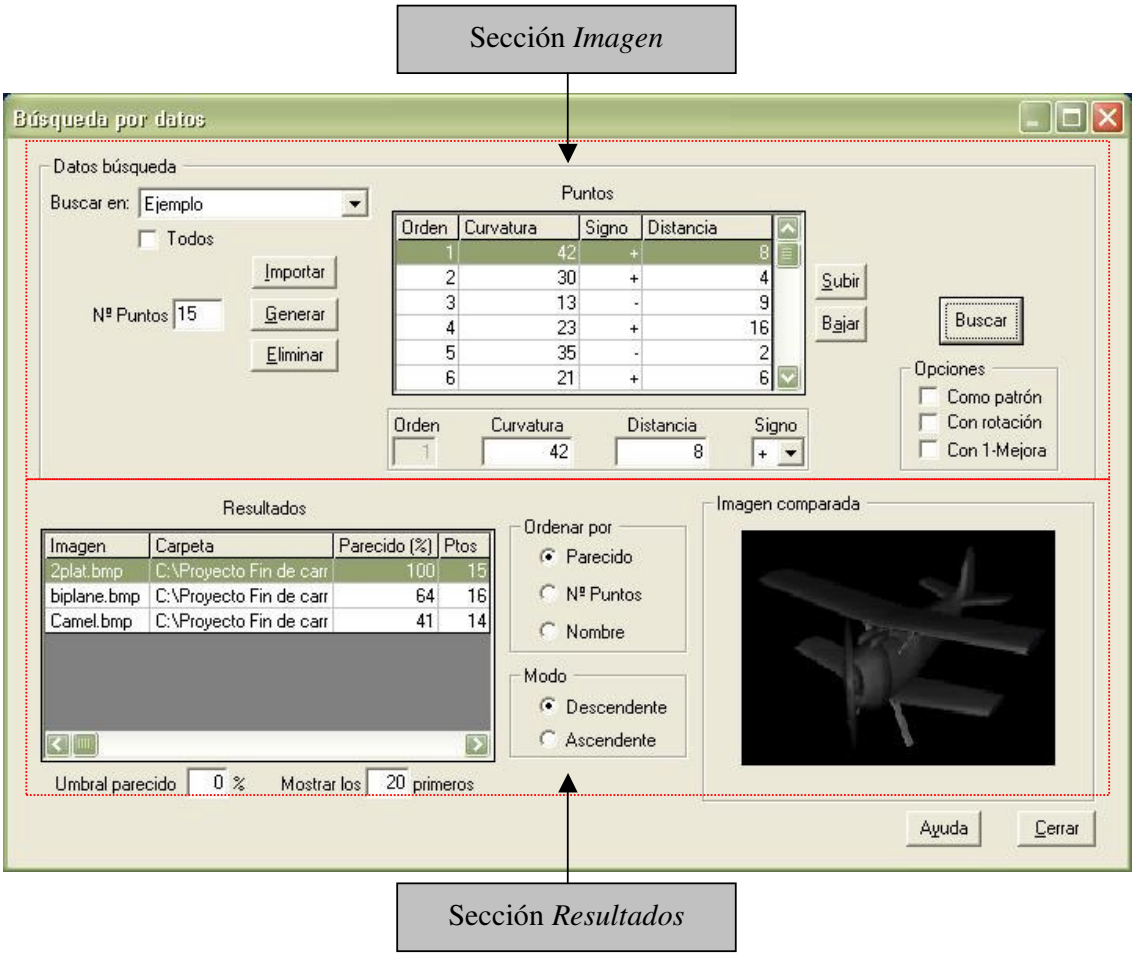


Figura 6.14: Aspecto de la ventana *Búsqueda por Datos*

Esta ventana (Figura 6.14) tiene la misma función que la anterior, "*Búsqueda por Imágenes*", es decir, buscar imágenes en la base de datos, pero con la salvedad de que, en este formulario, nosotros mismos creamos "artificialmente" las características de la imagen, en vez de ser extraídas a partir de una imagen real, esto es, a partir de un fichero *.bmp*.

La utilidad principal de esta ventana consiste en poder realizar comparaciones de imágenes más o menos parecidas a otras ya existentes en la base de datos, sin tener que dibujarlas en algún programa de retoque fotográfico, sino simplemente modificando alguno de los valores de sus puntos característicos.

Es por esto que añadimos una utilidad para esta ventana, el botón "*Importar*", que nos permitirá añadir los puntos característicos de imágenes ya existentes en la base de datos a nuestra tabla de búsqueda. Esto nos evita, en el caso de que deseemos comparar entre sí imágenes de la propia base de datos, tener que ir copiando uno a uno todos los datos de cada punto característico de una imagen. Una vez pulsado este botón, nos aparecerá la ventana "*Importar*" (ver Sección 6.2.8).

A continuación, vamos a analizar por secciones el funcionamiento del formulario "*Búsqueda por Datos*" (Figura 6.15).

- Sección *Imagen*

Figura 6.15: Aspecto de la sección *Imagen* de la ventana *Búsqueda por Datos*

En primer lugar, podemos seleccionar el grupo de imágenes sobre el que deseamos realizar la búsqueda, haciendo clic directamente sobre la lista desplegable. Si deseamos buscar en todos los grupos de imágenes, marcaremos la casilla titulada "*Todos*".

Lógicamente, en esta ventana no aparece la caja de texto "*Sigma*", puesto que no es necesario extraer los puntos característicos de la imagen de entrada, ya que somos nosotros mismos directamente los que le proporcionamos a la aplicación los puntos característicos.

En primer lugar, vamos a analizar la tabla o rejilla titulada "*Puntos*". Tenemos cuatro columnas:

- La columna "*Orden*" indica el orden en el que se disponen los puntos característicos de la supuesta imagen.
- La columna "*Curvatura*" indica el valor de curvatura, en tanto por ciento respecto a π.
- La columna "*Signo*" indica el signo de la función de curvatura para el punto característico en cuestión. Si es positivo estamos hablando de una curvatura del tipo convexo en ese punto. Si es negativo estaremos ante una curvatura del tipo cóncavo.
- La columna "*Distancia*" indica el valor de distancia de un punto característico respecto al siguiente. En el caso del último punto, será la distancia respecto al primero (de forma cíclica). El valor se muestra en tanto por ciento respecto a la longitud total del contorno, en el caso de no utilizar el modo de *búsqueda por patrón*. Si utilizamos ese modo, el valor representará el número total de píxeles del contorno desde el punto característico en cuestión hasta el siguiente.

Es por esto que debemos cuidar el hecho de que, si no usamos el modo de *búsqueda por patrón*, la suma de todos los valores de distancia debe ser 100. En otro caso, las distancias reflejan valores reales y habrá que introducirlas en valores absolutos.

A medida que vayamos seleccionando un punto característico de la tabla, se irá cargando la siguiente estructura con los datos del mismo para permitir su modificación, ya que cualquier cambio en dicha estructura también quedará reflejado recíprocamente en la tabla.

Figura 6.16: Aspecto de la estructura con los datos del punto seleccionado en la tabla *Puntos*.

Como se puede apreciar, el punto característico seleccionado en la tabla "*Puntos*" es el que está reflejado en la Figura 6.16. Podemos realizar las modificaciones que consideremos oportunas. La caja de texto "*Curvatura*" y "*Distancia*" pueden modificarse con valores enteros, mientras que en "*Signo*" sólo puede seleccionarse el valor "+" o "-".

En cuanto al "*Orden*", puede ser alterado pero no sobre la estructura, sino mediante los botones "*Subir*" y "*Bajar*", que permitir intercambiar puntos característicos hacia arriba y hacia abajo, después de seleccionarlos.

Figura 6.17: Aspecto del botón *Generar* y la caja de texto *Nº Puntos*.

La función del botón "*Generar*" y de la caja de texto denominada "*Nº Puntos*" (Figura 6.17) es establecer el número de puntos característicos que va a tener la imagen a buscar. No tenemos más que introducir el valor deseado en la caja de texto y pulsar, posteriormente, el botón "*Generar*". Inmediatamente a este hecho, la tabla "*Puntos*" pasará a tener el número de filas indicado, con todas las casillas en blanco (salvo "*Orden*") para ser rellenadas por el usuario. La columna "*Orden*" tendrá los valores (de arriba abajo) desde 1 hasta "*Nº Puntos*".

Existen dos maneras de eliminar puntos característicos de la tabla. La primera (que sólo sirve para eliminar los últimos puntos de la rejilla) es reducir el valor existente en "*Nº Puntos*" y pulsar "*Generar*". Pero la manera más adecuada es utilizar el botón "*Eliminar*". Simplemente, seleccionamos el punto que deseemos quitar de la tabla y pulsamos este botón. La aplicación restablecerá automáticamente la ordenación de los puntos característicos.

Por último, indicar que el botón "*Importar*" abre la ventana con el mismo nombre, que nos va a permitir recuperar los puntos característicos que deseemos de imágenes ya añadidas a nuestra base de datos, por si queremos buscar imágenes que son, realmente, pequeñas variaciones de otras ya existentes en la misma. Para mayor información sobre el formulario "*Importar*", hay que dirigirse a la Sección 6.2.8.

El botón de "*Buscar*" y sus opciones ya fueron explicados en la ventana "*Búsqueda por Imágenes*", al igual que la sección "*Resultados*"; ambas son exactamente iguales que la de dicha ventana.

Por último, los botones de "*Ayuda*" y "*Cerrar*":

- Botón "*Ayuda*"
  Nos permite abrir la ayuda en línea, directamente en la página *HTML* "*Formulario Búsqueda por Datos*".
- Botón "*Cerrar*"
  Cierra esta ventana y vuelve al menú principal.

6.2.5. Ventana Configuración.

Esta ventana nos permite configurar las opciones de la aplicación, permitiendo adaptarse a los tipos de imágenes que se pretenden buscar y optimizar los procesos según cada caso. Podemos configurar cuatro grupos de opciones: *opciones del servidor FSQL*, *opciones de búsqueda*, *opciones de imagen* (en general) y *opciones sobre las vistas*.

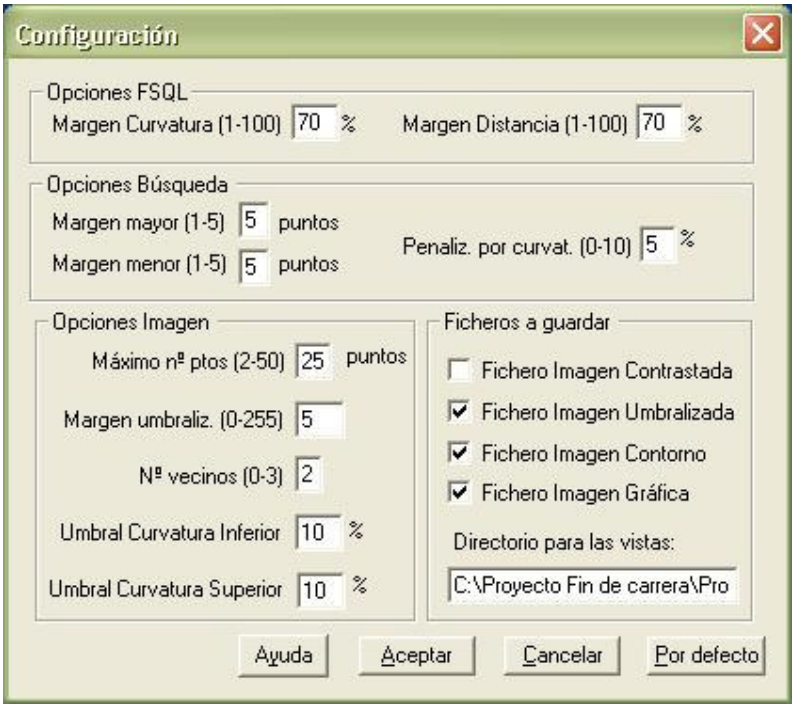


Figura 6.18: Aspecto la ventana *Configuración*

- Opciones FSQL

Son las únicas opciones que, al abandonar la aplicación, permanecerán guardadas en la base de datos y, por tanto, al volver a ejecutar el programa se mantendrán iguales. El resto de opciones que veremos posteriormente volverán a ciertos valores por defecto establecidos desde el diseño:

- "*Margen Curvatura*" nos permite ampliar o reducir el rango de curvaturas que se consideran parecidas a la estudiada en cada punto característico. Un margen de curvatura amplio aumentará el parecido entre las imágenes, mientras que un margen pequeño las distinguirá más. Su valor puede estar entre 1% y 100%. El valor por defecto es 70%.
- "*Margen Distancia*" nos permite ampliar o reducir el rango de distancias que se consideran parecidas a la estudiada en cada punto característico. Un margen de distancia amplio aumentará el parecido entre las imágenes, mientras que un margen pequeño las distinguirá más. Su valor puede estar entre 1% y 100%. El valor por defecto es 70%.

- Opciones de Búsqueda

Estas opciones afectarán a la búsqueda de imágenes y no a la adición de imágenes a la base de datos:

- "*Margen Mayor*" y "*Margen Menor*" nos permiten decidir qué imágenes de la base de datos serán comparadas con la de entrada y cuáles no. En concreto, se analizarán solamente aquellas imágenes cuyo número de puntos característicos se encuentre en el siguiente rango:

$$[n - mrg_{menor} , n + mrg_{mayor}]$$

  donde n es el número de puntos característicos de la imagen de entrada, mrg_{menor} es el margen menor y mrg_{mayor} es el margen mayor.

  Sus valores pueden establecerse entre 1 y 5. El valor por defecto para cada una de ellas es 5. Es importante señalar que una disminución de estos parámetros provocará que se analicen menos imágenes de la base de datos pero, a la vez, mejorará notablemente el rendimiento de la búsqueda.

- "*Penaliz. por curvat.*" nos permite especificar la importancia que va a tener la curvatura en la eliminación de un punto característico en la comparación de imágenes. Concretamente, el valor de este parámetro indica la penalización que provocará (en tanto por ciento) la eliminación de un punto característico con curvatura máxima (100%); si la curvatura es inferior, la penalización será también proporcionalmente inferior. Por tanto, si establecemos un valor alto de este parámetro, los puntos eliminados en el proceso de comparación conllevarán una penalización mayor en la imagen que si establecemos un valor pequeño.

  Sin embargo, además de esta penalización, existe la denominada *penalización base*, que "castiga" a aquellas imágenes que tengan un

número de puntos característicos diferente a la imagen de entrada. Esta penalización dependerá del número de puntos de diferencia con respecto a la imagen de entrada y con respecto al número total de puntos característicos de la misma.

El valor puede establecerse entre 0 y 10%, mientras que el valor por defecto será 5%. El caso concreto 0 implicaría que la penalización por la eliminación de un punto característico es independiente del valor de curvatura de dicho punto.

- Opciones de Imagen

Estos parámetros serán utilizados durante todo el proceso de extracción del contorno y obtención de puntos característicos:

- "*Máximo nº ptos*" nos permite especificar el máximo número de puntos característicos que podrá tener una imagen en la base de datos. En el caso de que una imagen intente ser añadida a la base de datos y supere esta cota, se cancelará el proceso y se mostrará un mensaje de error. Para más información sobre este error, consultar la Sección 6.2.9.

  Su valor puede estar entre 2 y 50 puntos. El valor tomado por defecto es 25.

- "*Margen umbraliz.*" nos permite establecer qué puntos formarán parte del fondo y cuáles del objeto, una vez calculado éste. Serían aquellos que estén en el intervalo [$color_{fondo}$ - $margen_{umb.}$, $color_{fondo}$ + $margen_{umb.}$].

  Si ponemos un margen demasiado pequeño o demasiado grande, podemos encontrar más entrantes y salientes en la forma.

El valor del "*Margen umbraliz.*" puede esta entre 0 y 255. El valor por defecto está fijado a 5.

- "*Nº Vecinos*" nos permite eliminar aquellos puntos de la imagen umbralizada que tengan un número de vecinos menor o igual que este valor. Su función, por consiguiente, es eliminar ruido puntual, es decir, puntos aislados y , además, suavizar los bordes de la forma (que no tenga muchos y pequeños entrantes y salientes) antes de calcular el contorno de la imagen.

  Su valor puede estar entre 0 y 3. El valor 0 implica eliminar únicamente píxeles aislados (sin vecinos), mientras que 3 permite eliminar ruido más grueso y redondear más la forma. El valor por defecto es 2.

- "*Umbral Inferior*" y "*Umbral Superior*" nos permiten establecer umbrales para el cálculo de puntos característicos. Sólo se tendrán en cuenta los puntos positivos que sean mayores que el umbral superior y todos los puntos negativos que sean menores que el umbral inferior.

  Por tanto, un aumento del umbral superior (disminución del umbral inferior) supondrá una reducción en el número de puntos característicos considerados y viceversa, una disminución del umbral superior (aumento del umbral inferior) supondrá un aumento del número de puntos característicos obtenidos.

  El valor de los umbrales puede estar entre 2 y 100%. El valor por defecto es 10%.

- Ficheros a guardar

Este grupo de opciones está destinado a organizar las vistas de cada una de las imágenes que se añaden a la base de datos. Estas vistas reflejan los resultados

intermedios que se van obteniendo durante el proceso de extracción de puntos característicos.

Las opciones "*Fichero Imagen Contrastada*", "*Fichero Imagen Umbralizada*", "*Fichero Imagen Contorno*" y "*Fichero Imagen Gráfica*" se refieren a las vistas de aumento de contraste, imagen umbralizada, contorno y gráfica de curvatura, respectivamente.

Si por ejemplo, la opción de "*Contorno*" está activada, cuando se calcule el contorno de cualquier imagen que se añada a la base de datos, se guardará el archivo correspondiente (*.bmp*) en disco. Igualmente para todas las opciones.

Por defecto, todas las opciones estarán marcadas salvo "*Fichero Imagen Contrastada*", debido a que es la que más espacio ocupará en el disco.

Falta decir en qué carpeta se almacenarán las vistas. Para ello contamos con el "*Directorio para las Vistas*". Debemos introducir aquí la carpeta en la que deseemos almacenar las vistas.

Al iniciar la aplicación, se comprobará si existe, en el directorio de trabajo de la misma, una carpeta denominada "*Vistas*". De no ser así, el programa la creará. En cualquier caso, este directorio será establecido como directorio de vistas.

Por último, debemos indicar el funcionamiento de los cuatro botones situados en la parte inferior:

- Botón "*Ayuda*"
  Nos permite abrir la ayuda en línea, directamente en la página *HTML* "*Formulario Configuración*".
- Botón "*Aceptar*"
  Valida los cambios realizados.
- Botón "*Cancelar*"

No guarda los cambios y vuelve al menú principal.

- Botón "*Por defecto*"

  Carga los valores por defecto de cada una de las opciones, excepto el directorio de vistas.

6.2.6. Ventana Conexión a *Oracle*.

Esta ventana (Figura 6.19) nos permite modificar la conexión al SGBD *Oracle 8i* y así, poder tener varios usuarios y varias bases de datos con imágenes y que el programa pueda conectarse a todas.

Figura 6.19: Aspecto la ventana *Conexión a la Base de Datos Oracle*

Los parámetros que podemos modificar son los siguientes:

- *Nombre de usuario*

  Identifica al usuario que va a utilizar la base de datos *Oracle*.
- *Contraseña*

  Identifica la contraseña utilizada por el usuario.
- *Database*

  Identifica la base de datos de *Oracle* a la que va a conectarse la aplicación.

Si pulsamos el botón "*Aceptar*", la aplicación tratará de realizar una conexión a la base de datos indicada, para el usuario y contraseña indicadas. En caso satisfactorio, se cerrará la ventana y la conexión quedará establecida. En caso contrario, se producirá un mensaje de error (ver Sección 6.2.9), indicando que no se ha podido realizar la conexión.

Si pulsamos el botón "*Cancelar*" no se realizará ningún tipo de cambio en la conexión a *Oracle*, y abandonaremos esta ventana.

Por último, el botón "*Ayuda*" nos permite abrir la ayuda en línea, directamente en la página *HTML* "*Formulario Conexión a Oracle*".

6.2.7. Ventana Añadir Grupo de Imágenes.

Esta ventana solamente puede ser ejecutada desde la ventana "*Modificar Base de Datos*", cuando se pulsa el botón "*Añadir*" de la sección "*Grupo de Imágenes*".

Figura 6.20: Aspecto la ventana *Añadir Grupo de Imágenes*

El objetivo de este sencillo formulario es añadir un nuevo grupo de imágenes a la base de datos, para poder organizar nuestras imágenes más cómodamente.

En primer lugar, debemos introducir el nombre del grupo de imágenes (no debe existir ya uno con el mismo nombre). Después, introduciremos el valor numérico *sigma* correspondiente (por defecto es *3.00*). Es importante elegir un buen *sigma* en este instante,

puesto que una vez creado el grupo y añadidas imágenes en él, será imposible modificar este valor, puesto que crearía incongruencias (unas imágenes con un valor de *sigma* y otras con otro valor distinto). Si realmente queremos modificar el valor de *sigma* de un grupo de imágenes, tendremos que borrar ese grupo y volverlo a crear de nuevo, esta vez con el nuevo valor de *sigma* que consideremos adecuado.

Si pulsamos el botón "*Aceptar*", regresaremos al formulario "*Modificar Base de Datos*", donde aparecerá el nuevo grupo. Si pulsamos el botón "*Cancelar*", regresaremos al formulario "*Modificar Base de Datos*" sin haber realizado ningún cambio.

Por último, el botón de "*Ayuda*" abre la ayuda en línea, directamente en la página *HTML* "*Formulario Añadir Grupo de Imágenes*".

6.2.8. Ventana Importar.

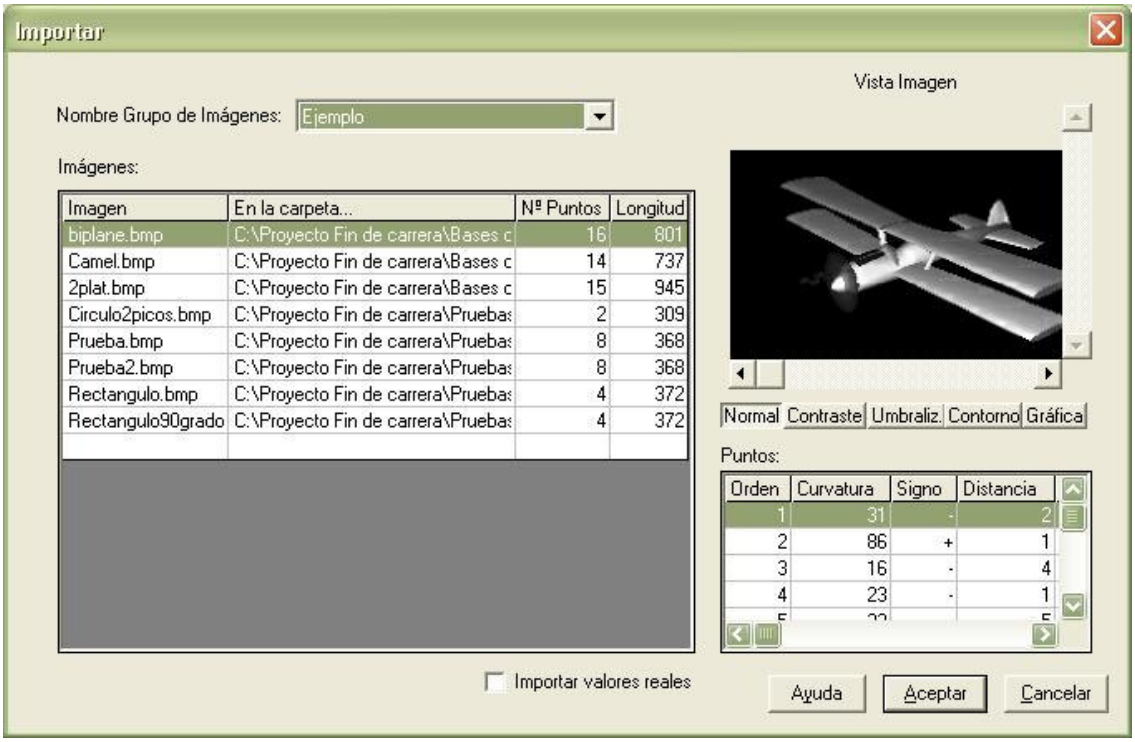


Figura 6.21: Aspecto de la ventana *Importar*

Esta ventana (Figura 6.21) solamente puede ser ejecutada desde la ventana "*Búsqueda por Datos*", cuando se pulsa el botón "*Importar*" de la sección "*Imagen*".

Su objetivo es recuperar ciertos puntos característicos de alguna de las imágenes de la base de datos para ser utilizados en el formulario "*Búsqueda por Datos*".

Al igual que el formulario "*Modificar Base de Datos*" (con el que guarda una similitud bastante grande), podemos seleccionar el grupo de imágenes que deseemos y se mostrarán las imágenes de ese grupo en la rejilla titulada "*Imágenes*". La diferencia radica en que no podemos, en este formulario, añadir grupo de imágenes, puesto que su utilidad es únicamente la de realizar consultas.

Cuando seleccionamos una imagen en la tabla "*Imágenes*", se carga la vista correspondiente, como en el formulario "*Modificar Base de Datos*" y se cargan los puntos característicos de la imagen seleccionada en la correspondiente rejilla "*Puntos*", que tiene el mismo aspecto que el mostrado en la Figura 6.10 (ventana "*Modificar Base de Datos*").

Para importar los puntos característicos que deseemos, no tenemos más que pinchar con el botón izquierdo del ratón en el punto característico inicial, arrastrar el puntero y soltarlo en el punto característico final que deseamos importar. Quedarán seleccionadas todas las filas desde este punto inicial hasta el punto final marcado, como muestra la Figura 6.22.

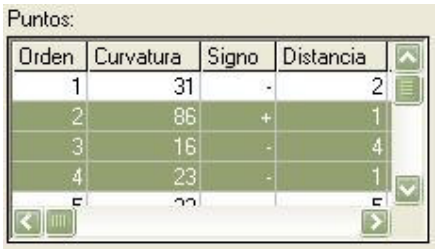


Figura 6.22: Muestra de una selección de puntos en la ventana *Importar*

En el ejemplo de la Figura 6.22, el punto característico inicial sería el de orden 2, y el punto final sería el de orden 4.

Ahora tenemos que decidir si queremos importar los valores relativos (búsquedas normales) o bien si deseamos importar los valores reales o absolutos para las distancias (búsquedas por patrón). Para el último caso, hay que marcar la opción "*Importar Valores Reales*". Además, esto provocará la selección automática, en el formulario "*Modificar Base de Datos*", de la opción "*Buscar como patrón*".

Una vez pulsado el botón "*Aceptar*", la ventana "*Búsqueda por Datos*" recuperará estos valores (relativos o reales) y los mostrará en la rejilla "*Puntos*" de esta ventana. Además, estos se insertarán justo delante del punto que estaba señalado en la rejilla antes de pulsar el botón "*Importar*", adoptando el valor de "*Orden*" adecuado.

Si pulsamos el botón "*Aceptar*" sin haber seleccionado ningún rango de puntos, simplemente no hará nada. Si pulsamos el botón "*Cancelar*", volveremos al formulario "*Búsqueda por Datos*" sin realizar ningún cambio.

Por último, contamos con el botón "*Ayuda*", que abre la ayuda en línea, directamente en la página *HTML* "*Formulario Importar*".

6.2.9. Errores.

En esta sección vamos a recopilar todos los errores controlados que pueden darse en la aplicación (ventana por ventana). Cada error constará del mensaje descriptivo del mismo (**Error**), la acción que se ejecutaba cuando se produjo el error (**Acción**) y la posible causa de éste (**Causa**).

Todos los errores serán mostrados en una ventana más o menos como la de la Figura 6.23, cuando se dispare el evento correspondiente.

Figura 6.23: Ejemplo de notificación de error

- **Ventanas "*Modificar Base de Datos*" y "Búsqueda por Imágenes"**

  - **Error**: "*Ya existe esa imagen en algún grupo de imágenes*" (Sólo en ventana "*Modificar Base de Datos*").
  - **Acción**: Se produce cuando tratamos de añadir una imagen a la base de datos.
  - **Causa**: No podemos introducir una imagen que ya existe en la base de datos, incluso aunque pertenezca a otro grupo de imágenes distinto.

  - **Error:** "*Error I/O: no se puede abrir el archivo. Utilice solo ficheros bmp*".
  - **Acción**: Se produce cuando tratamos de añadir una imagen a la base de datos o cuando tratamos de cargar una imagen para realizar su búsqueda.
  - **Causa:** Solamente se pueden abrir archivos *.bmp* en esta aplicación. Cualquier otro tipo de archivo provocará este error.

  - **Error:** " *Error I/O: no se puede leer el archivo. Fichero bmp corrupto*".
  - **Acción**: Se produce cuando tratamos de añadir una imagen a la base de datos o cuando tratamos de cargar una imagen para realizar su búsqueda.
  - **Causa:** El fichero *.bmp* introducido está corrupto o no tiene el formato adecuado.

  - **Error:** " *Error I/O: no se puede escribir en el directorio de vistas seleccionado (ver Configuración)*" (Sólo en ventana "*Modificar Base de Datos*").
  - **Acción**: Se produce cuando tratamos de añadir una imagen a la base de datos.

- **Causa:** El sistema no puede escribir los ficheros de vistas en el directorio indicado en la ventana de "*Configuración*". Puede ser que hayamos establecido como directorio de vistas un dispositivo en el que no pueda escribirse, como un CD, DVD o, simplemente, que el dispositivo esté protegido contra escritura. Otra razón puede ser que se haya ocupado el máximo espacio del dispositivo en el que se encuentra el directorio de vistas. Debemos modificar este directorio en la ventana "*Configuración*" para solventar el error.

- **Error:** "*Imagen no válida: No es de 256 colores o no es una gama de grises*".
- **Acción**: Se produce cuando tratamos de añadir una imagen a la base de datos o cuando tratamos de buscar una imagen en la base de datos.
- **Causa:** La aplicación solamente admite imágenes de 256 colores y que utilicen únicamente grises en su contenido, es decir, píxeles cuyas tres componentes *RGB* sean iguales. Por tanto, la imagen introducida no cumple alguna de estas condiciones.

- **Error:** "*Imagen no válida: no posee un fondo homogéneo*".
- **Acción**: Se produce cuando tratamos de añadir una imagen a la base de datos o cuando tratamos de buscar una imagen en la base de datos.
- **Causa:** Se produce cuando tratamos de añadir una imagen a la base de datos o en la fase previa a la búsqueda de esa imagen. La aplicación solamente admite imágenes con un fondo homogéneo para poder obtener el contorno. La aplicación detecta cuando estas imágenes no son válidas y lo avisa con este mensaje.

- **Error:** "*Contorno no válido: contorno vacío*".
- **Acción**: Se produce cuando tratamos de añadir una imagen a la base de datos o cuando tratamos de buscar una imagen en la base de datos.
- **Causa:** La imagen no posee ningún objeto, o bien es muy pequeño y ha sido eliminado por el filtro anti-ruido. Por tanto, ha sido incapaz de obtener un contorno y lo avisa con este mensaje.

- **Error:** "*Contorno no válido: número de puntos característicos demasiado grande*".
- **Acción**: Se produce cuando tratamos de añadir una imagen a la base de datos o cuando tratamos de buscar una imagen en la base de datos.
- **Causa:** El proceso de extracción de puntos característicos de la imagen ha resultado devolver un número de puntos mayor que el indicado en la ventana "*Configuración*". Tendremos que aumentar el parámetro "*Máximo nº ptos*" en dicha ventana si deseamos incorporar esta imagen a la base de datos, o bien tratar de disminuir, mediante los demás parámetros de la ventana "*Configuración*" el número de puntos característicos detectados (ver *Configuración*).

- **Error:** "*Contorno no válido: no hay puntos característicos*".
- **Acción**: Se produce cuando tratamos de añadir una imagen a la base de datos o cuando tratamos de buscar una imagen en la base de datos.
- **Causa:** El proceso de extracción de puntos característicos de la imagen ha resultado no devolver ningún punto. La aplicación lo desecha porque no arroja información alguna.

- **Error:** "*Contorno no válido: sólo posee un punto característico*".
- **Acción**: Se produce cuando tratamos de añadir una imagen a la base de datos o cuando tratamos de buscar una imagen en la base de datos.
- **Causa:** El proceso de extracción de puntos característicos de la imagen ha resultado devolver un único punto. Se desecha porque arroja muy poca información.

- **Error:** "*Error general: insuficiente memoria*".
- **Acción**: Se produce cuando tratamos de añadir una imagen a la base de datos o cuando tratamos de cargar o buscar una imagen en la base de datos.
- **Causa:** Se produce cuando tratamos de añadir una imagen a la base de datos o en la fase previa a la búsqueda de esa imagen. Si se produce este error, la aplicación será abortada y volveremos al sistema operativo.

- **Ventana "*Búsqueda por Datos*"**

  - **Error:** "*Algún valor de curvatura es incorrecto*".
  - **Acción**: Se produce cuando tratamos de buscar una imagen en la base de datos mediante la introducción de sus puntos característicos.
  - **Causa:** Los valores de curvatura deben situarse entre 0 y 100%, independientemente de si se va a utilizar una búsqueda por imagen o por patrón.

  - **Error:** "*Algún valor de distancia es incorrecto*".
  - **Acción**: Se produce cuando tratamos de buscar una imagen en la base de datos mediante la introducción de sus puntos característicos.
  - **Causa:** Los valores de distancia, en el caso de búsqueda por imagen, deben situarse entre 0 y 100. En el caso de búsqueda por patrón, deben ser números mayores o iguales a 0, puesto que la distancia debe estar en términos absolutos.

  - **Error:** "*Algún valor de signo no es válido*".
  - **Acción**: Se produce cuando tratamos de buscar una imagen en la base de datos mediante la introducción de sus puntos característicos.
  - **Causa:** Los valores de signo deben ser o bien "+" o bien "-". Cualquier otro valor supone un error en la aplicación.

- **Ventana "*Configuración*"**

  - **Error:** "*Margen de Curvatura incorrecto*".
  - **Acción**: Se produce al aceptar la configuración introducida.
  - **Causa:** Los valores posibles para esta opción deben estar en el intervalo [1,100].

  - **Error:** "*Margen de Distancia incorrecto*".
  - **Acción**: Se produce al aceptar la configuración introducida.

- **Causa:** Los valores posibles para esta opción deben estar en el intervalo [1,100].

- **Error:** "*Margen mayor incorrecto*".
- **Acción**: Se produce al aceptar la configuración introducida.
- **Causa:** Los valores posibles para esta opción deben estar en el intervalo [1,5].

- **Error:** "*Margen menor incorrecto*".
- **Acción**: Se produce al aceptar la configuración introducida.
- **Causa:** Los valores posibles para esta opción deben estar en el intervalo [1,5].

- **Error:** "*Penalización por curvatura incorrecta*".
- **Acción**: Se produce al aceptar la configuración introducida.
- **Causa:** Los valores posibles para esta opción deben estar en el intervalo [0,10].

- **Error:** "*Número máximo de puntos incorrecto*".
- **Acción**: Se produce al aceptar la configuración introducida.
- **Causa:** Los valores posibles para esta opción deben estar en el intervalo [2,50].

- **Error:** "*Existen imágenes con mayor número de puntos que el máximo introducido*".
- **Acción**: Se produce al aceptar la configuración introducida.
- **Causa:** Ya existen imágenes en la base de datos con más puntos que los indicados en esta ventana. Para evitar inconsistencias, el programa avisa. Podemos aumentar este parámetro o eliminar las imágenes que superen este valor de puntos característicos en la ventana "*Modificar Base de Datos*" para solventar este hecho.

- **Error:** "*Margen de umbralización incorrecto*".
- **Acción**: Se produce al aceptar la configuración introducida.

- **Causa:** Los valores posibles para esta opción deben estar en el intervalo [0,255].

- **Error:** "*Número de vecinos incorrecto*".
- **Acción**: Se produce al aceptar la configuración introducida.
- **Causa:** Los valores posibles para esta opción deben estar en el intervalo [0,3].

- **Error:** "*Umbral inferior incorrecto. Debe ser un valor entre 2 y 100*".
- **Acción**: Se produce al aceptar la configuración introducida.
- **Causa:** Los valores posibles para esta opción deben estar en el intervalo [2,100].

- **Error:** "*Umbral superior incorrecto. Debe ser un valor entre 2 y 100*".
- **Acción**: Se produce al aceptar la configuración introducida.
- **Causa:** Los valores posibles para esta opción deben estar en el intervalo [2,100].

- **Error:** "*Directorio de vistas incorrecto*".
- **Acción**: Se produce al aceptar la configuración introducida.
- **Causa:** Se produce cuando ponemos una cadena nula como directorio de vistas. Lógicamente no es correcto, puesto que debemos poner la ruta en la que se encuentra el directorio en cuestión.

- **Error:** "*El directorio de vistas introducido no existe*".
- **Acción**: Se produce al aceptar la configuración introducida.
- **Causa:** El directorio de vistas introducido no existe. El programa debe impedir este hecho y solicitar al usuario que introduzca una carpeta que exista realmente.

- **Ventana "*Conexión a Oracle*"**

- **Error:** "*El usuario, contraseña o base de datos introducidos no corresponden a ninguna cuenta Oracle*".

- **Acción**: Se produce al aceptar el grupo de imágenes a añadir.
- **Causa:** No existe en nuestro SGBD *Oracle* ninguna cuenta de usuario que tenga el nombre, contraseña y base de datos introducidos, es decir, al menos uno de los tres parámetros no corresponde a una misma cuenta.

- **Ventana "*Añadir Grupo de Imágenes*"**

  - **Error:** "*Introduzca un nombre de grupo correcto*".
  - **Acción**: Se produce al aceptar el grupo de imágenes a añadir.
  - **Causa:** El nombre de grupo que se ha introducido en esta caja de texto no es válido: debe contener un cadena de caracteres no vacía que no comience por un espacio en blanco, para que la aplicación lo valide. Este error simplemente se añade para evitar malentendidos con grupos de imágenes aparentemente iguales, pero distintos para el programa.

  - **Error:** "*Introduzca un sigma correcto*".
  - **Acción**: Se produce al aceptar el grupo de imágenes a añadir.
  - **Causa:** El valor de *sigma* que se ha introducido en esta caja de texto no es válido: debe contener un número válido.

  - **Error:** "*Imposible añadir ese valor de 'Nombre' o 'Sigma' en la base de datos*".
  - **Acción**: Se produce al aceptar el grupo de imágenes a añadir.
  - **Causa:** El nombre del grupo o el valor de *sigma* que se ha introducido en esta caja de texto no es válido. Puede ser, por ejemplo, porque el nombre de grupo introducido ya existía en la base de datos y ésta debe impedirlo para evitar inconsistencias.

7. ANÁLISIS Y RESULTADOS

En este capítulo vamos a mostrar y analizar resultados obtenidos con nuestra aplicación, para poder calibrar la capacidad de caracterización de las imágenes y de comparación de las mismas, indicando en cada caso las opciones que se utilizan para un funcionamiento óptimo. También realizaremos un pequeño estudio de la eficiencia de la aplicación y mostraremos los resultados obtenidos en una máquina real, con distintas imágenes y distintas opciones de configuración.

En general, los resultados de la obtención del contorno y la extracción de puntos característicos resulta satisfactoria, si bien hemos reducido mucho el tipo de imágenes que pueden utilizarse en la aplicación. Si se desea conocer más detalles sobre el comportamiento de la aplicación en esta fase, debemos dirigirnos al Capítulo 4. Por lo demás, si nos ceñimos a las imágenes que se han indicado que pueden ser utilizadas en la aplicación, difícilmente podemos encontrar aspectos desfavorables.

En cualquier caso, podemos examinar y comprobar cómo se ha realizado la obtención del contorno y la extracción de puntos característicos, simplemente comprobando en la ventana "*Modificar Base de Datos*" de la aplicación, las vistas que se han creado y los datos referentes a los atributos de cada uno de esos puntos característicos.

7.1. Pruebas

La comparación de imágenes, por otra parte, sí va a ser el centro de análisis de esta sección, que en definitiva, es el objetivo final del proyecto. Se mostrarán figuras y tablas que pretenden dar una idea (sin necesidad de utilizar la aplicación) del comportamiento del programa y su capacidad de encontrar parecidos entre imágenes.

Separamos las pruebas en dos grandes grupos: *pruebas de formas básicas* y *pruebas de formas complejas.*

El primer banco de pruebas trata de analizar cómo se comporta la aplicación ante figuras sencillas, con un número reducido de puntos característicos, como triángulos, cuadrados o rectángulos. El segundo banco de pruebas consiste en formas más complejas, con un número de puntos característicos generalmente mayor, y analizaremos los resultados obtenidos.

Como es lógico, las pruebas serán obtenidas directamente de la ejecución de la aplicación, pero combinando las imágenes de entrada, los parámetros de búsqueda y las opciones de configuración general del programa.

Antes de realizar las pruebas, podemos pensar qué propiedades deben cumplirse en la aplicación para que podamos considerar que funciona correctamente. Establecemos una función *Similitud(a,b)*, o de manera más abreviada, *S(a,b)*:

$$S(a,b) = p$$

Se dice que *a* y *b* son similares (parecidos) en un *p%*, es decir, indica que si *a* es una imagen de entrada y *b* es una imagen de la base de datos, entonces si buscamos *a*, encontramos que el parecido con *b* es del *p %*. Las propiedades que deben cumplirse son:

- *"Reflexividad"*: $S(a,a) = 100$

- *Diferenciabilidad*: Si dos imágenes son distintas (es diferente, al menos, un atributo de, al menos, uno de los puntos característicos de ambas imágenes), entonces se cumple que $S(a,b) \neq 100$.

- *"Simetría"*: $S(a,b) => S(b,a)$

- *Invariabilidad a traslación*: Si tenemos dos imágenes, cada una de ellas con dos formas iguales pero en distintas coordenadas, su parecido debe ser el mismo.

- *Invariabilidad a rotación*: Si tenemos dos imágenes, cada una de ellas con dos formas iguales pero una rotada respecto a la otra en el eje perpendicular al plano que define la pantalla, su parecido debe ser el mismo.

- *Invariabilidad al tamaño*: Si tenemos dos imágenes, cada una de ellas con dos formas iguales pero una está en una escala distinta a la otra, si mantiene las mismas proporciones de distancia y curvatura, su parecido debe ser el mismo.

- Si utilizamos las opciones "*Con Rotación*" o "*Con 1-Mejora*", el parecido debe ser al menos, el mismo que si no utilizamos dichas opciones.

$$S_{rotac}(a,b) \geq S(a,b) \text{ y } S_{1\text{-}mej}(a,b) \geq S(a,b)$$

Tras los siguientes resultados apreciaremos cómo muchas de estas propiedades (casi todas) se cumplen en todos los casos, pero otras como la *rotación* o *invariabilidad al tamaño*, pueden producir resultados un poco distintos a los esperados. Por supuesto, explicaremos en cada caso, las razones de esto.

7.1.1. Pruebas de Formas Básicas

Como hemos mencionado anteriormente, se trata de realizar una serie de pruebas de la aplicación utilizando diversas imágenes con formas geométricas sencillas y modificando los parámetros de búsqueda y configuración general de la aplicación.

Por otra parte, esta sección nos permitirá, además de comprender mejor la utilidad de cada uno de los parámetros que se usan, arrojar toda la información sobre el funcionamiento de la aplicación.

Vamos a utilizar los parámetros por defecto que establece la aplicación, pero con los siguientes cambios: *umbral de curvatura superior* a 4% y *umbral de curvatura inferior* a 4%.

La razón de esto es que las figuras básicas suelen tener pocos vértices y entonces, por pequeños que sean éstos, es conveniente localizarlos; de lo contrario podríamos clasificar

incorrectamente. Un ejemplo de un caso indeseable, sería obtener que la imagen de un pentágono solamente tiene cuatro vértices porque hay uno que no ha superado el umbral superior de curvatura; en este caso, el pentágono y algún cuadrado podrían llegar a ser prácticamente iguales, sin realmente serlo.

A continuación, añadimos a la base de datos varios *grupos de imágenes* (*Cuadrados*, *Rectangulos*, *Triangulos* y *Pentagonos*) con un valor de *sigma* de 3.00 (valor por defecto). Tras esto, añadimos las imágenes de la Tabla 7.1 (que podemos encontrarlas en la carpeta "*Imágenes*" del CD adjunto a esta memoria) a cada grupo correspondiente.

Tabla 7.1: Imágenes de formas básicas añadidas a la base de datos

Tipo (Grupo)	**Nombre (.bmp)**	**Imagen**	**Nº Puntos**	**Longitud**
Cuadrados	*Cuadrado*		4	272
	Cuadradopeq		4	176
	Cuadradotras		4	272
	Cuadrado45gr		4	217
Rectángulos	*Rectangulo*		4	360
	Rectangulogran		4	532
	Rectangulopeq		4	238
	Rectangulo90gr		4	360
	Rectangulo60gr		4	312

Triángulos	*Triangulo1*		2	204
	Triangulo2		3	208
	Triangulo3		3	208
	Triangulo4		3	238
	Triangulo5		3	232
	Triangulo6		3	210
	Triangulo7		3	192
	Triangulo8		3	204
	Triangulo4rot90gr		3	238
	Triangulo4rot45gr		3	263
	Triangulo4rot135gr		3	243
Pentágonos	*Pentagono*		5	224
	Pentagono90gr		5	224

Tabla 7.1 (Continuación): Imágenes de formas básicas añadidas a la base de datos

Antes de comenzar a realizar una búsqueda para comentar los resultados obtenidos, hemos de indicar que la imagen *triangulo1.bmp* solamente tiene 2 puntos característicos (ver Tabla 7.1), a pesar de ser un triángulo. La razón es que la curvatura de su vértice superior es más pequeña que 4%, que es el umbral que hemos establecido. Concretamente, el valor de curvatura de ese vértice es del 2%. Más adelante, se volverá a este punto.

Tras esto, vamos a realizar la búsqueda de la imagen *cuadrado.bmp* en la base de datos y vamos a mostrar los resultados obtenidos en la Tabla 7.2.

Imagen (.bmp)	**Nº Puntos**	**Parecido (%)**
Cuadrado	4	100
Cuadradotras	4	100
Cuadradopeq	4	100
Cuadrado45gr	4	95
Rectangulogran	4	93
Rectangulo	4	91
Rectangulo60gr	4	91
Rectangulo90gr	4	91
Rectangulopeq	4	87
Pentagono	5	71
Pentagono90gr	5	71
Triangulo4	3	59
Triangulo4rot45gr	3	51
Triangulo5	3	51
Triangulo4rot135gr	3	50
Triangulo4rot90gr	3	48
Triangulo3	3	35
Triangulo6	3	35
Triangulo8	3	16
Triangulo7	3	14
Triangulo2	3	4
Triangulo1	2	0

Tabla 7.2: Resultados obtenidos tras buscar la imagen *cuadrado.bmp*

En primer lugar, hay que comentar que la propia imagen *cuadrado.bmp* ha sido encontrada con un parecido (como cabía esperar) del 100%, cumpliendo así, la *reflexividad* en la relación, es decir, toda imagen deben tener un parecido del 100% consigo misma, habiéndose realizado los cálculos con los mismos parámetros.

Además, encontramos que la imagen *cuadradotras.bmp*, que no es más que la misma forma pero trasladada hacia la derecha y arriba, tiene un parecido con la imagen de entrada del 100%. Esto debe ser así puesto que la búsqueda es *invariante a la traslación*.

La imagen *cuadradopeq.bmp* representa el mismo cuadrado de la imagen de entrada, pero a diferente escala, aunque manteniendo las proporciones (sigue siendo un cuadrado y todos los lados miden igual). Puesto que el sistema preserva la *invariabilidad al tamaño* de la forma, el resultado es el esperado, es decir, 100% de parecido respecto a la imagen de entrada. Esto es así porque los valores de la distancia no están en valores absolutos (número de píxeles del contorno) sino relativos a la longitud total del contorno, es decir, al perímetro de la figura.

La imagen *cuadrado45gr.bmp* es la imagen de entrada rotada 45º. Sin embargo, apreciamos que no obtenemos un 100% sino un 95%. La razón estriba en que la rotación de la forma se produce a través de un programa de retoque fotográfico y éste, tras la rotación, no mantiene las distancias, sino que aproxima. Podemos apreciar esto en la ventana "*Modificar Base de Datos*", consultando los puntos característicos de la imagen *cuadrado45gr.bmp*. Veremos que las distancias no son 25% para todos los puntos, como cabe esperar en un cuadrado, sino que se diferencian ligeramente.

Ocurre lo mismo en el caso de la curvatura, ya que las esquinas no resultan exactamente iguales y tampoco producen un parecido total. Los 4 vértices de la imagen *cuadrado.bmp* tienen curvatura 12%. Sin embargo, en la imagen rotada ninguno llega a ese valor (entre 8% y 9%). En cualquier caso, queda constancia del parecido casi total entre ambas imágenes y, en líneas generales, de la *invariabilidad a la rotación*.

A continuación, apreciamos en la Tabla 7.2 que los rectángulos añadidos a la base de datos obtienen un parecido muy alto. Todos poseen, lógicamente, cuatro puntos al igual que los cuadrados y las curvaturas de las esquinas son de 90º al igual que éstos. La única diferencia, por tanto, se halla en las distancias, y por ello el parecido de cuadrados y rectángulos queda mostrado claramente. Así mismo, la propiedad de *diferenciabilidad* se cumple también, ya que aún siendo muy parecidos, mantienen (los rectángulos) un grado de diferencia con el cuadrado.

Hasta el momento, hemos obtenido como más parecidas, figuras con cuatro puntos característicos. Sin embargo, ahora nos encontramos con dos formas de cinco puntos característicos: los pentágonos. A pesar de que los triángulos (siguiente grupo a analizar) tienen también un punto característico de diferencia con los cuadrados, la aplicación considera a los pentágonos más parecidos que los triángulos, en función a la curvatura de sus vértices y a las distancias entre ellos, lo cual parece satisfactorio. Además, ambos pentágonos arrojan el mismo parecido (puesto que uno es la versión rotada 45º del otro).

Por último, los triángulos son las imágenes que menos parecido tienen con respecto al cuadrado, puesto que tienen un punto característico menos que éstos. El orden en el que se hallan, en función del parecido, estos triángulos también es importante. Vemos que los triángulos 4 (y sus rotaciones), 5, 3 y 6 son los más parecidos: parece bastante lógico.

Por otra parte, podemos apreciar que la imagen *triangulo1.bmp* tiene un parecido del 0%. La razón (como se indicó anteriormente) estriba en que posee, únicamente, dos puntos característicos, lo que la distingue aún más del cuadrado que el resto de triángulos. Si deseamos obtener los tres puntos del triángulo, hemos de modificar el *Umbral de Curvatura Superior* y ponerlo a 2%. Si realizamos de nuevo la búsqueda, la imagen *triangulo1.bmp* tendrá tres puntos característicos y un parecido del 36% respecto a *cuadrado.bmp*.

Sin embargo, observamos que resulta más parecida la imagen *triangulo8.bmp* que la imagen *triangulo7.bmp*, lo cual parece contrario al sentido común. La explicación sobre esto ya fue comentada en la Sección 4.1.3.

Podemos comprobar en estas búsquedas cómo la columna "*Nº Puntos*" nos muestra que los parecidos entre formas simples tiene mucho que ver con el número de puntos característicos, aunque puede haber imágenes con más (o menos) puntos característicos y tener mayor parecido a la imagen de entrada que otras con el mismo número de puntos. Se establece, por tanto, una ordenación por número de puntos característicos en la tabla (primero todas las imágenes de 4 puntos, después las de 5 y, finalmente, los triángulos de 3) en forma piramidal.

A continuación vamos a realizar otra búsqueda, cuya imagen de entrada es *rectangulo.bmp*. Con estos resultados vamos a poder obtener más conclusiones.

Imagen (.bmp)	**Nº Puntos**	**Parecido (%)**
Rectangulo	4	100
Rectangulogran	4	99
Rectangulopeq	4	96
Cuadrado	4	91
Cuadradopeq	4	91
Cuadradotras	4	91
Cuadrado45gr	4	88
Rectangulo60gr	4	83
Rectangulo90gr	4	83
Pentagono	5	71
Pentagono90gr	5	71
Triangulo4	3	56
Triangulo4rot45gr	3	54
Triangulo5	3	51
Triangulo4rot90gr	3	50
Triangulo4rot135gr	3	48
Triangulo6	3	38
Triangulo3	3	35
Triangulo8	3	19
Triangulo7	3	16
Triangulo2	3	4
Triangulo1	2	0

Tabla 7.3: Resultados obtenidos tras buscar la imagen *rectangulo.bmp*

Como podemos apreciar en la Tabla 7.3, los resultados obtenidos son muy parecidos a los de la tabla anterior, salvo ciertos aspectos.

Por supuesto, sigue cumpliéndose la *reflexividad*, y por consiguiente, la imagen *rectangulo.bmp* tiene un 100% de parecido con ella misma.

Además, podemos apreciar que se cumple la *simetría*. Si antes buscamos la imagen *cuadrado.bmp* y encontramos que *rectangulo.bmp* se parecía en un 91% a la primera, ahora es la imagen *cuadrado.bmp* la que se parece un 91% a *rectangulo.bmp* (que es la imagen de entrada).

La imagen *rectangulogran.bmp*, a pesar de ser mucho más grande, mantiene las proporciones casi exactamente, por lo cual el parecido es casi total (99%), manteniéndose la *invariabilidad al tamaño*. Podemos verificar la pequeña diferencia en la ventana "*Modificar Base de Datos*" con los datos que mostramos en la Figura 7.1.

rectangulo.bmp

Orden	Curvatura	Signo	Dist. sigte.
1	12	+	19
2	12	+	31
3	12	+	19
4	12	+	31

rectangulogran.bmp

Orden	Curvatura	Signo	Dist. sigte.
1	12	+	20
2	12	+	30
3	12	+	20
4	12	+	30

Figura 7.1: Puntos característicos de las imágenes *rectangulo.bmp* y *rectangulogran.bmp*

Se trata de que una de las dimensiones del rectángulo mide el 19% (cada arista) en *rectangulo.bmp* y un 20% en *rectangulogran.bmp* (ver Figura 7.1) y, por consiguiente, la segunda dimensión mide un 31% en el primero y un 30% en el segundo. Esta misma situación se da con la imagen *rectangulopeq.bmp*, aunque la diferencia es mayor, 16% la primera dimensión y 34% la segunda.

Por último, ya que el resto de resultados es igual que en el caso de *cuadrado.bmp* como imagen de entrada, puede parecer un poco extraño que los rectángulos rotados (*rectangulo60gr.bmp* y *rectangulo90gr.bmp*) se parezcan menos al rectángulo de entrada que todos los cuadrados. La razón es que, en estos dos casos, el punto característico primero a buscar no coincide con el primero de *rectangulo.bmp*. Por tanto, es necesario indicarle a la aplicación que se desea buscar una imagen rotada, pulsando la opción "*Con Rotación*". Así, el programa buscará la imagen más parecida, pero comenzando por todos los puntos característicos posibles, quedándose con el mejor, es decir, con el que produzca un parecido mayor.

Los resultados obtenidos (solamente mostramos los diez primeros) al buscar la imagen *rectangulo.bmp* con la opción "*Con Rotación*" marcada, son los que incluimos en la Tabla 7.4.

Imagen (.bmp)	Nº Puntos	Parecido (%)
Rectangulo	4	100
Rectangulo90gr	4	100
Rectangulogran	4	99
Rectangulo60gr	4	98
Rectangulopeq	4	96
Cuadrado45gr	4	94
Cuadrado	4	91
Cuadradotras	4	91
Cuadradopeq	4	91
Pentagono	5	71

Tabla 7.4: Resultados obtenidos tras buscar la imagen *rectangulo.bmp* con la opción "*Con Rotación*" activada

Como se aprecia, la imagen *Rectangulo90gr.bmp* es considerada exactamente igual que la original (*invariabilidad a la rotación*), a la vez que la imagen *Rectangulo60gr.bmp* es considerada también casi igual que la imagen de entrada, puesto que se ha encontrado la combinación que la hace más parecida a *rectangulo.bmp*.

También es curioso observar como la imagen *Cuadrado45gr.bmp* ha aumentado su parecido (de 88% a 94%). La razón es que ya vimos que no era realmente un cuadrado, ya que tenía unos lados más grandes que otros a causa de la rotación a través de un programa de retoque fotográfico y por esto, la aplicación (con la rotación activada) ha sido capaz de encontrar una combinación mejor para compararla con la imagen *rectangulo.bmp*.

Gracias a esto, obtenemos de nuevo como más parecidos al rectángulo de entrada, todos los rectángulos existentes, antes que los cuadrados. El resto de imágenes no se ven afectadas, puesto que no se hallan rotadas, sino que simplemente son distintas.

Es interesante comentar los resultados que se obtienen si modificamos las opciones, de la ventana "*Configuración*", *margen curvatura* y *margen distancia* (opción del servidor *FSQL*). Su función es indicar el parecido que se obtendrá por cada uno de los puntos característicos de una imagen de la base de datos con la imagen de entrada. Cuanto menor sea el margen de curvatura (o de distancia), más parecido se le exigirá a cada punto característico de la imagen buscada con respecto a la de entrada en función a la curvatura (o a la distancia), y viceversa. Los valores por defecto para estos parámetros es de 70.

Vamos a mostrar los resultados (diez primeros) de buscar la imagen *cuadrado.bmp* en la base de datos, con *margen curvatura* y *margen distancia* iguales ambos a 30 (en vez de 70).

Imagen (.bmp)	Nº Puntos	Parecido (%)
Cuadrado	4	100
Cuadradotras	4	100
Cuadradopeq	4	100
Cuadrado45gr	4	89
Rectangulogran	4	83
Rectangulo	4	80
Rectangulo90	4	80
Rectangulo60	4	80
Rectangulopeq	4	70
Pentagono	4	58

Tabla 7.5: Resultados obtenidos tras buscar la imagen *cuadrado.bmp* con márgenes FSQL igual a 30

Los resultados muestran que hemos diferenciado más las imágenes, como era de esperar. Por ejemplo, *cuadrado45gr.bmp* ya sólo se parece un 89% (antes un 95%), *rectangulo.bmp* se parecía antes un 91% y ahora sólo un 80%, o *pentagono.bmp* que, antes se parecía un 71%, ahora tiene un parecido del 58%.

Sin embargo, apreciamos perfectamente como las imágenes que tienen un 100% de parecido no ven alterado este parecido, evitando incumplir la propiedad de *reflexividad* de la relación.

Es por esto que, dependiendo de la rigidez que deseemos, podremos establecer unos valores de *margen curvatura* y *margen distancia* determinados, aunque el orden (de más a parecido a menos) no variará demasiado, salvo que realicemos cambios extremos en estas dos opciones.

El siguiente análisis es sobre los triángulos. Vamos a buscar la imagen denominada *triangulo4.bmp* en la base de datos y vamos a mostrar en la Tabla 7.6 los resultados obtenidos.

Imagen (.bmp)	Nº Puntos	Parecido (%)
Triangulo4	3	100
Triangulo5	3	84
Triangulo4rot135gr	3	83
Triangulo4rot45gr	3	82
Triangulo4rot90gr	3	79
Triangulo3	3	76
Triangulo6	3	66
Cuadrado	4	59
Cuadradotras	4	59
Cuadradopeq	4	59
Rectangulo90gr	4	59
Rectangulo	4	56
Rectangulogran	4	56
Rectangulopeq	4	54
Rectangulo60gr	4	52
Triangulo8	3	47
Cuadrado45gr	4	46
Triangulo7	3	45
Triangulo2	3	41
Triangulo1	2	38
Pentagono	5	32
Pentagono90gr	5	32

Tabla 7.6: Resultados obtenidos tras buscar la imagen *triangulo4.bmp*

Podemos observar el cumplimiento de la *reflexividad* y *simetría* (con *cuadrado.bmp* y *rectangulo.bmp*) en esta tabla.

Los triángulos 5, 4 (versiones rotadas), 3 y 6 aparecen parecidos al triángulo de entrada. Se aprecia que el parecido no es tan grande como en otras figuras similares, sin mencionar el caso menos deseable (*triangulo2.bmp*). La razón es doble:

- Por un lado, disponemos de muy poca información sobre estas figuras (únicamente tres puntos característicos). Cualquier pequeña variación en alguno de los puntos del triángulo, supone modificar el 33% de la información de éste.
- Sumado a lo anterior, cuando pasamos de *triangulo4.bmp* a *triangulo2.bmp* cualquier persona aprecia un estrechamiento de la figura, pero con casi total seguridad, señalaría un parecido mucho mayor que 41%. Sin embargo, en la

aplicación, este hundimiento supone la variación de la curvatura de cada punto característico y de la modificación de las distancias, por lo que este pequeño cambio para el ojo humano, supone un cambio notable en la caracterización de la figura por parte de la aplicación.

Podemos comparar las gráficas y apreciar esta diferencia en la caracterización de las formas. La Figura 7.2 muestra tal hecho.

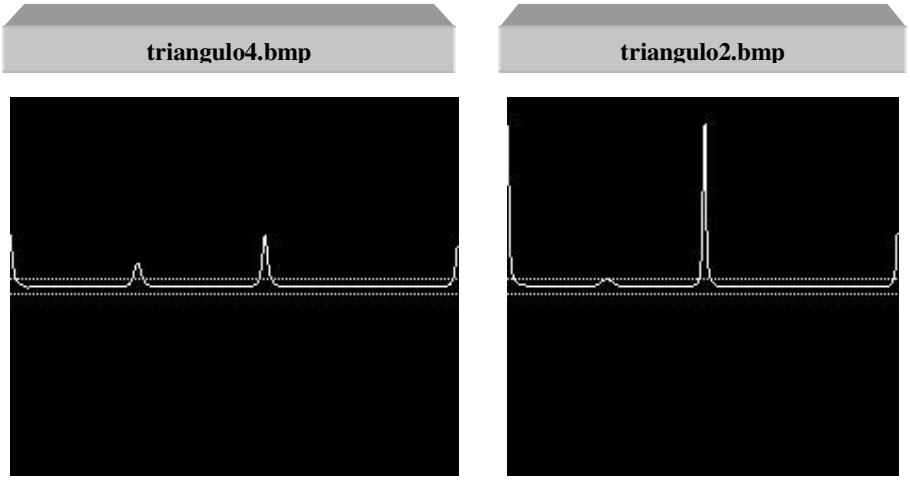


Figura 7.2: Gráficas de la función de curvatura de las imágenes *triangulo4.bmp* y *triangulo2.bmp*

Observando la Figura 7.2, apreciamos que los picos primero y tercero de la imagen *triangulo2.bmp* son mucho más agudos que los de la imagen *triangulo4.bmp*, mientras que el segundo es ligeramente más pequeño. Las distancias relativas también son distintas, pero no son las responsables del resultado, como puede apreciarse.

Los problemas obtenidos con la imagen *triangulo1.bmp* y entre *triangulo8.bmp* y *triangulo7.bmp* se explican claramente, como se ha mencionado con anterioridad, en la Sección 4.1.3.

Los pentágonos resultan ser las figuras que menos parecido tienen con el triángulo de entrada; es lógico puesto que posee dos puntos característicos más, lo que implica una penalización doble durante la búsqueda (hay que eliminar dos picos del pentágono para comparar, y eso supone una penalización).

Por supuesto, es lógico que los triángulos rotados tengan un parecido menor que la imagen *triangulo5.bmp*. Debemos marcar la opción "*Con Rotación*" si queremos tenerla en cuenta y obtendremos los siguientes resultados (sólo los cinco primeros) en la Tabla 7.7.

Imagen (.bmp)	**Nº Puntos**	**Parecido (%)**
Triangulo4	3	100
Triangulo4rot90gr	3	99
Triangulo4rot45gr	3	91
Triangulo4rot135gr	3	89
Triangulo5	3	88

Tabla 7.7: Resultados obtenidos tras buscar la imagen *triangulo4.bmp* con la opción "*Con Rotación*" activada

El hecho de que la *invariabilidad a la rotación* no se cumpla tan estrictamente como en los cuadrados o rectángulos, queda explicado por la razón doble expuesta más arriba.

En definitiva, y después de exponer todas estas pruebas, hay que decir que los resultados obtenidos con la aplicación (utilizando formas básicas) son satisfactorios, si bien los problemas más importantes señalan a la función del cálculo de la curvatura como la principal responsable, ya que en determinadas situaciones extremas (picos muy agudos) responde con valores de curvatura menores a los esperados, además del problema de picos dobles en donde debiera existir un único pico. Para más información, consultar la Sección 4.1.3.

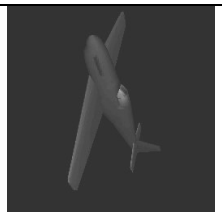
Además, la posibilidad de configurar la aplicación en función del tipo de figura a buscar o añadir, permite solucionar muchos de los pequeños problemas y poder así, adaptarse a cada tipo de imagen. Queda bien claro que es conveniente, para cada tipo de imagen, una configuración específica para obtener resultados óptimos, aunque no necesaria.

7.1.2. Pruebas de Formas Complejas

Después de haber realizado una serie de pruebas con imágenes sencillas, vamos a mostrar cómo se comporta la aplicación *FuzzyFinder* ante imágenes más complejas, con un número de puntos característicos mayor.

Vamos a mostrar una tabla (Tabla 7.8) con las imágenes a utilizar en esta sección, indicando el número de puntos característicos obtenidos y la longitud total de su contorno o perímetro, teniendo en cuenta que hemos utilizado las opciones por defecto salvo el *margen de curvatura superior* e *inferior*, que se han establecido en 12.

Tabla 7.8: Imágenes de formas complejas añadidas a la base de datos

Nombre (.bmp)	**Imagen**	**Nº Puntos**	**Longitud**
51b		8	691
51bpeq		8	472
51b90gr		8	688
51b270gr		8	689
51bmenos1p		7	646
51bmenos2p		6	675
51bmas1p		9	709
51bmas2p		10	714

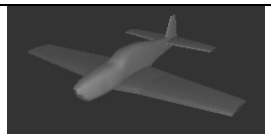
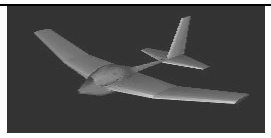
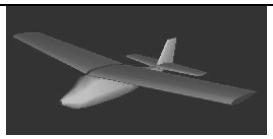
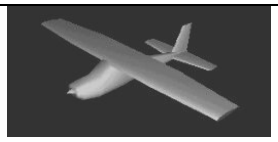
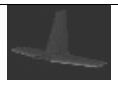
51bsinala		6	610
Extra-300		8	677
P-51M		8	673
Pimouss		8	689
cessnaliketrainer		8	707
Cessna_Card		7	635
cola		3	168

Tabla 7.8 (Continuación): Imágenes de formas complejas añadidas a la base de datos

Obsérvese que la imagen *51bmenos2p.bmp* tiene dos puntos característicos menos, debido a que esta imagen se ha generado a partir de ciertos cambios sobre la imagen original. En concreto, el ala derecha del avión ha sido redondeada, eliminando ese punto característico existente en *51b.bmp* y la parte superior de la cola ha sido también modificada (aunque no se aprecie en la imagen) para evitar que la curvatura de ese punto sobrepase el umbral de curvatura superior y genere un nuevo punto característico más.

Vamos a realizar la búsqueda de la imagen *51b.bmp* en la base de datos sin ninguna opción marcada, sin la "*1-Mejora*" y con la opción "*Con Rotación*" desactivada. Los resultados son los mostrados en la Tabla 7.9.

Imagen (.bmp)	Nº Puntos	Ninguna opción	Con 1-Mejora	Con Rotación
51b	8	100	100	100
51b90gr	8	99	99	99
Extra-300	8	90	90	90
P-51M	8	89	89	89
Cessnaliketrainer	8	81	81	81
51bpeq	8	77	77	77
51bmenos1p	7	77	**83**	77
51bmas1p	9	76	**86**	76
Pimouss	8	73	73	73
51bmenos2p	6	72	72	72
51bmas2p	10	65	65	65
51b270gr	8	48	48	**99**
Cessna_Card	7	48	**60**	48
51bsinala	6	22	22	22
Cola	3	0	0	0

Tabla 7.9: Resultados obtenidos tras buscar la imagen *51b.bmp* sin opciones, con la "*1-Mejora*" y con la opción "*Rotación*" activada

En primer lugar, la *reflexividad* se cumple ya que la imagen *51b.bmp* tiene un parecido del 100% consigo misma, como muestra la Tabla 7.9.

Además, la *invariabilidad a rotación* queda patente con el 99% de parecido de las imágenes *51b90gr.bmp* (rotada 90º a partir de la imagen original) y *51b270gr.bmp* (rotada 270º a partir de la imagen original). La primera no necesita marcar la opción "*Con Rotación*" para mejorar el resultado porque ésta y la imagen original comparten el mismo punto característico de inicio, mientras que en el caso de las imágenes *51b270gr.bmp* y la original no ocurre lo mismo, de ahí que necesitemos buscar el punto inicial que maximiza el parecido ("*Con Rotación*"). Ninguna otra imagen se ve mejorada en parecido por la rotación, puesto que ninguna se halla en una mejor disposición espacial que la actual respecto a *51b.bmp*.

En cuanto a la *invariabilidad al tamaño*, los resultados no son todo lo bueno que se desearía (77% para la imagen *51bpeq.bmp*), pero el resultado es lógico, puesto que aunque se mantienen los 8 puntos característicos, todos han visto modificados sus atributos (distancia y, sobre todo, la curvatura) ligeramente, lo que globalmente supone una diferencia notable. Cuántos más puntos, la aplicación tiene más dificultad para mantener la *invariabilidad al tamaño*. La solución a esto implica mejorar la función de curvatura, para que cada punto obtenga igual curvatura independientemente del tamaño.

También se aprecia como las imágenes *51bmas1p.bmp* y *51bmenos1p.bmp* que se obtienen a partir de la original, pero añadiendo (eliminando) un punto, tienen un parecido aproximado y mayor que las imágenes *51bmas2p.bmp* y *51bmenos2p.bmp*, que se obtienen añadiendo (eliminando) dos puntos característicos, lo que parece en un principio bastante lógico.

Si aplicamos la técnica de *1-Mejora*, observamos mejorías en las imágenes con un punto más (menos) que la imagen de entrada. Las imágenes *51bmenos1p.bmp*, *51bmas1p.bmp* y *Cessna_Card.bmp* se ven mejoradas por esta técnica, que es capaz de encontrar el punto que hay que eliminar de la imagen que tiene un punto más para maximizar el parecido.

Sin embargo, la imagen *51bsinala.bmp* es también una modificación de la imagen de entrada (perdemos 2 puntos característicos) y el parecido obtenido es del 22%. El motivo de esto es que los puntos característicos eliminados no son los que deberían suprimirse para maximizar el parecido en la comparación, pero la aplicación prescinde de los puntos de menor curvatura. Podría modificarse el algoritmo utilizado para optimizar este tipo de situaciones.

Esto mismo ocurre en las imágenes con un punto más (o menos) que la de entrada, pero con la técnica de *1-Mejora* se soluciona, ya que examina qué punto debemos suprimir para maximizar el parecido final. Sin embargo, la posibilidad de utilizar la "*2-Mejora*", es decir, eliminar los dos puntos que maximizan el parecido tiene una complejidad muy grande, por lo que no podemos solucionarlo (ver Sección 7.2, que nos muestra un pequeño estudio de la eficiencia del programa). Este es uno de los puntos débiles de la aplicación.

Las imágenes *Extra-300.bmp* y *P-51M.bmp* tienen un gran parecido con *51b.bmp*, incluso más que todas las versiones de ésta última a la que se le han quitado o añadido puntos. Aunque puede ser muy discutido este hecho, nadie puede dudar del parecido entre estas imágenes y la de entrada, al igual que *cessnaliketrainer.bmp* y *Pimouss.bmp* que aunque menos, también tienen un parecido considerable, y la aplicación así lo muestra.

La imagen *CessnaCard.bmp*, a pesar de ser muy similar a la imagen de entrada, se parece únicamente un 48% sin *1-Mejora* y un 60% con ella. La razón es que el punto característico que se indica en la Figura 7.3 tiene una curvatura del 11% y hemos puesto el umbral en el 12%.

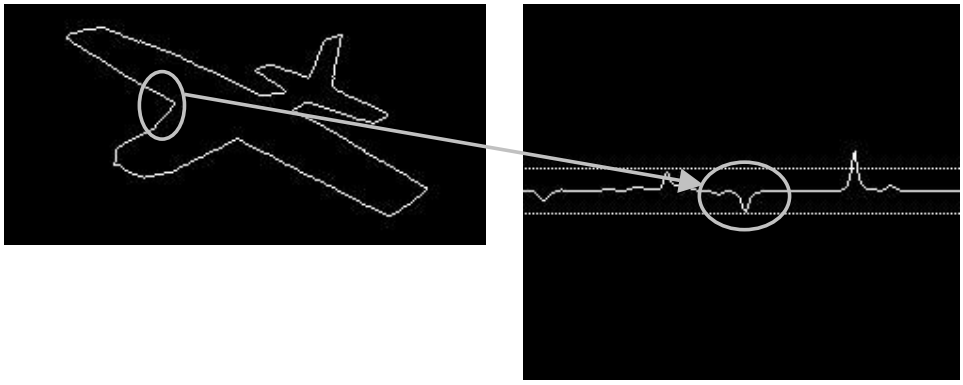


Figura 7.3: Vistas del contorno y gráfica de la función de curvaturas de la imagen *Cessna_Card.bmp*

Como se aprecia en la Figura 7.3, el punto cóncavo señalado por la primera circunferencia en la imagen del contorno, produce tras el cálculo de la función de curvaturas, el pico señalado por la segunda circunferencia. Este pico no llega a superar el umbral inferior establecido, por lo que no es considerado punto característico.

No tenemos más que cambiar este umbral (a 11%) y añadir de nuevo la imagen a la base de datos. El resultado obtenido es que la imagen añadida se parece a *51b.bmp* en un 75%, lo que tiene mucho más sentido.

Cada vez que la aplicación se encuentra con imágenes con más o menos puntos característicos que los que tiene la imagen de entrada, no tiene más remedio que eliminar los necesarios para poder realizar la comparación. Como es lógico, siempre se eliminan puntos de la imagen que tiene un número mayor de ellos, sea ésta la imagen de entrada o la imagen comparada. Por cada punto característico eliminado, la aplicación penaliza la imagen en un tanto por ciento de puntos eliminados respecto a puntos totales. Por ejemplo, si la imagen que busco tiene 6 puntos y la imagen comparada tiene 8, tendremos que eliminar 2 de los 8 puntos de ésta última para poder realizar la comparación, penalizándose con un 25% (2 de 8).

Sin embargo, existe otra penalización adicional: la *penalización por curvatura*. Si el punto eliminado posee una curvatura mayor, su penalización es mayor, y viceversa.

Si nos dirigimos a la ventana de "*Configuración*" y modificamos el parámetro *Penalización por curvatura* con el valor 0 y 10 (en vez de 5), los resultados de las imágenes que no tienen 8 puntos, es decir, un número de puntos distinto que la imagen de entrada son los que se muestran en la Tabla 7.10.

Imagen (.bmp)	**Nº Puntos**	**Parecido (%) (Penalización = 0%)**	**Parecido (%) (Penalización = 5%)**	**Parecido (%) (Penalización = 10%)**
51bmenos1p	7	78	77	76
51bmas1p	9	77	76	76
51bmenos2p	6	74	72	70
51bmas2p	10	67	65	65
Cessna_Card	7	49	48	47
51bsinala	6	24	22	20
Cola	3	0	0	0

Tabla 7.10: Resultados obtenidos tras buscar la imagen *51b.bmp* (8 puntos) con la opción "*Penalización por Curvatura*" establecida en 0, 5 y 10.

Observamos como, a medida que aumentamos la penalización, los parecidos disminuyen sutilmente. Además, las imágenes con un punto más (o menos) que la de entrada disminuyen en menor medida su parecido que aquellas que difieren en más puntos de la imagen de entrada. El resto de imágenes no se ve alterada en su parecido, razón por la que no se han incluido en esta tabla.

Si bien no nos encontramos ante un parámetro decisivo en la obtención de resultados de la aplicación, éste nos permite calibrar o perfeccionar aún más el proceso de búsqueda de imágenes.

A continuación, vamos a ver los resultados de *buscar por patrón* la imagen *cola.bmp* (Tabla 7.11) en la base de datos, es decir, tratamos de localizar todas aquellas imágenes que tengan en alguna parte de su contorno, la forma de la imagen *cola.bmp.* Como ya sabemos, buscaremos valores de distancia absolutos, no relativos a la longitud del contorno de cada

imagen, porque las distancias relativas a la longitud del patrón no coinciden con las distancias relativas a la longitud de la imagen de la base de datos, por consiguiente, la búsqueda no tiene más remedio que realizarse con los valores absolutos de las distancias.

Imagen (.bmp)	**Nº Puntos**	**Parecido (%)**
Cola	3	100
51b	8	93
51bmas1p	9	93
51bmas2p	10	93
51bsinala	6	93
51b90gr	8	78
51b270gr	8	78
Cessnaliketrainer	8	66
Cessna_Card	7	64
Extra-300	8	54
51bmenos1p	7	53
P-51M	8	53
51bpeq	8	39
51bmenos2p	6	33
Pimouss	8	26

Tabla 7.11: Resultados obtenidos tras buscar el patrón *cola.bmp* en las imágenes de la Base de Datos

De nuevo, se cumple la *reflexividad* para el caso de *búsqueda por patrón* con la imagen *cola.bmp*. Hay que decir, además, que en este modo de búsqueda, no utilizamos la penalización, porque no necesitamos eliminar puntos para poder realizar la consulta.

Además, la imagen *51b.bmp* y las que se obtienen a partir de ella tienen un parecido del 93%, salvo aquellas cuyos puntos característicos de la cola han sido modificados, disminuyéndose en ese caso el parecido.

En formas más complejas y desde un punto de vista general, es muy representativo apreciar como aparece una estructura piramidal de imágenes resultado, donde la cúspide la conforman imágenes con el mismo número de puntos característicos que la imagen de entrada, las imágenes con un punto más y un punto menos que la imagen de entrada aparecen en el segundo eslabón, y así sucesivamente. Es, por tanto, indiscutible la dependencia existente entre el número de puntos característicos y el parecido entre imágenes, según la

estrategia seguida en esta aplicación, mayor aún si tenemos imágenes con un número grande de puntos característicos. Esto no quiere decir que sea, ni muchísimo menos, el único factor que diferencie las imágenes, pero sí lo convierte en decisivo.

Los resultados obtenidos son los esperados en cualquier caso. Cuando utilizamos imágenes sencillas con un número reducido de puntos característicos, cualquier cambio en el contorno supone un gran cambio en el parecido final de la búsqueda, pero también un gran cambio en el aspecto general de la forma en cuestión, por lo que los resultados son muy buenos. Sin embargo, en imágenes con un número grande de puntos característicos, pequeños cambios en su contorno no suponen gran diferencia al ojo humano, pero sí una gran diferencia para la aplicación, cuyas curvaturas y distancias entre puntos característicos se ven alteradas de forma notable, con la consiguiente disminución del parecido entre las imágenes involucradas en el proceso.

A todo lo anterior, debemos sumar que, a partir de dos puntos característicos más (o menos) que la imagen de entrada, el sistema elimina los puntos de menor curvatura, pudiendo estar suprimiendo, precisamente, los puntos característicos que más parecido generaban entre ambas formas, disminuyendo enormemente el parecido.

En el caso de imágenes con un punto característico más (o menos) que la imagen de entrada, la técnica de *1-Mejora*, nos permite estudiar qué punto debemos eliminar de la imagen para maximizar el parecido entre ambas imágenes, sin embargo, aplicar este proceso implica aumentar el número de consultas a realizar, con el consiguiente retraso en la búsqueda. Por este motivo, es imposible realizar este mismo proceso con imágenes con dos o más puntos característicos de diferencia con respecto a la imagen original, ya que las combinaciones se ven incrementadas enormemente. Para analizar este último punto, debemos dirigirnos a la sección que sigue.

7.2. Estudio de la Eficiencia

En este apartado vamos a tratar de realizar un pequeño estudio de la eficiencia de la aplicación. En primer lugar, mostraremos la complejidad algorítmica de las diferentes

técnicas utilizadas en el programa, además de una explicación de por qué no pueden utilizarse conjuntamente las técnicas de "Rotación" y "1-Mejora".

Tras esto, expondremos una serie de pruebas en una máquina real, e indicaremos los tiempos empleados para realizar diversas consultas con distintas características.

7.2.1. Complejidad Algorítmica de las Técnicas

En general, para buscar una imagen en la base de datos tenemos que distinguir dos fases claramente diferenciadas: primero debemos obtener los datos de la imagen de entrada y preparar los datos en tablas temporales para organizar las consultas y, segundo, realizar las consultas pertinentes para obtener los resultados necesarios cada vez.

En este estudio vamos a obviar la primera fase, ya que aunque puede tener valores temporales notables en determinadas consultas, siempre es mayor la segunda fase y, además, es difícil de estudiar por grupos de casos, mientras que el número total de consultas sí puede estudiarse fácilmente y además, es el punto realmente importante de la complejidad en la búsqueda de imágenes en la base de datos. Antes de comenzar el análisis, vamos a definir algunas variables y vamos a aclarar ciertos conceptos:

- Definimos *n* como el número de puntos característicos de la imagen de entrada.
- La búsqueda de imágenes con un número de puntos característicos igual o mayor que la imagen de entrada requiere una única consulta (si no utilizamos ni la técnica de rotación ni la técnica de 1-mejora).
- La búsqueda de imágenes con un número de puntos característicos menor que la imagen de entrada requiere tantas consultas como indique el *margen menor* (en la ventana "*Configuración*" de la aplicación) en el peor de los casos, ya que realizamos una búsqueda por cada conjunto de imágenes menores, cada uno de ellos con el mismo número de puntos característicos. Denominemos *m* al *margen menor* y al *margen mayor*, ya que vamos a considerarlos iguales, aunque pueden no serlo.
- El tiempo necesario para realizar una consulta depende considerablemente del número de puntos característicos de la imagen de entrada y, en menor medida, del número de imágenes existente en la base de datos.

- Vamos a suponer en este pequeño estudio el peor caso posible, es decir, existen realmente *m*-conjuntos de imágenes menores a la de entrada (si no existiera uno de ellos, se ignoraría esa búsqueda y sería tiempo que nos ahorraríamos). Además, siempre va a haber una imagen con un punto más que la de entrada (al igual que con un punto menos) para realizar la 1-Mejora en sus dos casos.

7.2.1.1 Complejidad Algorítmica para el Modo de *Búsqueda por Imagen*

En cuanto a la *búsqueda por imágenes* podemos analizar las siguientes configuraciones, en función de las técnicas de búsqueda disponibles en la aplicación:

1) *Sin Rotación*. *Sin 1-Mejora*.

$$N^{o}\, consultas = 1 + m$$

El "1" indica la consulta necesaria para realizar la búsqueda de imágenes iguales o mayores. Además, necesitamos *m* consultas más para las imágenes menores (con *n*-1, *n*-2, ..., *n*-*m* puntos), debido a que la aplicación realiza una consulta por cada conjunto de imágenes con el mismo número de puntos característicos. No olvidemos que estamos teniendo en cuenta el caso peor.

El número de consultas solamente depende del *margen menor* establecido en la aplicación, pero no depende del número de puntos característicos de la imagen de entrada, lo cual es una gran ventaja respecto a las otras combinaciones.

2) *Con Rotación*. *Sin 1-Mejora*.

$$N^{o}\, consultas = n + mn = (m+1)n$$

n consultas para las imágenes iguales y mayores, porque tenemos que realizar todas las combinaciones posibles para la rotación, es decir, comenzando desde el primer punto, desde el segundo y así, hasta el último punto, por tanto *n* en total. Además, para cada conjunto de imágenes con un número menor de puntos

característicos (de *n*-1 hasta *n*-*m*) hay que realizar una búsqueda con rotación también, por lo que tenemos *mn* consultas más. En total, (*m*+1)*n*.

Podemos apreciar en este caso, la dependencia de la búsqueda con el número de puntos característicos de la imagen de entrada, lo que incrementa el coste temporal de la misma, además de la dependencia ya existente con el *margen menor* establecido.

Podemos, en este punto indicar que, si bien el *margen mayor* tiene la misma utilidad práctica que el *margen menor* pero para las imágenes con un número mayor de puntos característicos que la imagen de entrada, puesto que la aplicación es capaz de realizar en una sola consulta (sin rotación) o en *n* consultas (con rotación) la búsqueda de imágenes mayores, este margen tiene escasa importancia en el estudio de la complejidad, mientras que el *margen menor* puede ser importante, ya que de él depende el número de conjuntos de imágenes menores que van a analizarse.

3) *Sin Rotación. Con 1-Mejora.*

$$N^{\circ}\,consultas = 1 + (n+1) + n + (m-1) = 2n + m + 1$$

El "1" indica la consulta de las imágenes iguales y mayores, salvo la que tiene *n*+1 puntos que es incluida en la 1-Mejora (ver Sección 4.2.1.1). El término (*n*+1) nos indica el coste de realizar la 1-Mejora en las imágenes con *n*+1 puntos, ya que hay que eliminar un punto de la imagen mayor (*n*+1) y comparar con la imagen menor (*n*+1 veces). El término *n* es el coste de realizar la 1-Mejora en las imágenes con *n*-1 puntos, ya que hay que eliminar un punto de la imagen mayor (*n*) y comparar con la imagen menor (*n* veces). El término (*m*-1) indica el número de consultas necesario para realizar la búsqueda de las imágenes menores, salvo la que tiene *n*-1 puntos que es incluida, al igual que la anterior, en la 1-Mejora y no debemos volverla a sumar.

En general, la 1-Mejora solamente afecta a dos grupos de imágenes (las que tienen un punto menos y las que tienen un punto más que la imagen de entrada), por lo que el número de consultas tiene menor complejidad que la rotación. Para el caso de *m*=5 (por defecto) y, por supuesto, siguiendo en el caso peor posible, la rotación

presentaría un número de consultas igual a 6*n*, mientras que la 1-Mejora tendría un número de consultas igual a 2*n*+ 6.

4) *Con Rotación*. *Con 1-Mejora* (no permitida en la aplicación).

$$N^{o}\,consultas = n + (n+1)n + n(n-1) + (m-1)n = 2n^2 + mn$$

donde la *n* refleja el número de consultas necesario para realizar la búsqueda de imágenes con un número de puntos mayor o igual que la imagen de entrada, pero teniendo en cuenta la rotación, es decir, probando todos los puntos de inicio posibles. El término (*n*+1)*n* es el resultado de aplicar la 1-Mejora mayor y, para cada opción (*n*+1 en total), calcular la rotación (*n* veces), mientras que el término *n*(*n*-1) es el resultado de aplicar la 1-Mejora menor y, también para cada opción (ahora *n* en total), calcular la rotación (*n*-1 veces). Con el resto de imágenes menores, tendríamos una única consulta por grupo de imágenes con el mismo número de puntos si no estuviera la rotación activada (*m*-1 consultas). Pero, al estarlo, tenemos que probar todas las combinaciones posibles, es decir, *n*-veces (*m*-1), o sea, (*m*-1) *n*.

De este último punto se desprende que, utilizar ambas combinaciones conjuntamente durante una búsqueda, puede aumentar enormemente el coste temporal de la misma, ya que nos encontramos con un término al cuadrado. Analicemos los resultados obtenidos en la Tabla 7.12, donde mostramos el número de consultas necesarias para realizar una búsqueda de una imagen con *n* puntos característicos, cuando se ha establecido un *margen mayor* y un *margen menor* en *m*.

Técnicas utilizadas	**Nº Consultas**					
	***n* = 5 *m* = 5**	***n* = 10 *m* = 5**	***n* = 15 *m* = 5**	***n* = 5 *m* = 2**	***n* = 10 *m* = 2**	***n* = 15 *m* = 2**
Sin Rotación - Sin 1-Mejora	6	6	6	3	3	3
Con Rotación - Sin 1-Mejora	30	60	90	15	30	45
Sin Rotación - Con 1-Mejora	16	26	36	13	23	33
Con Rotación - Con 1-Mejora	75	250	525	60	220	480

Tabla 7.12: Número total de consultas utilizando las cuatro combinaciones y distintos valores de *n* y *m*

En primer lugar, comentar que por mucho que modifiquemos el valor de *n*, la opción "*Sin Rotación - Sin 1-Mejora*" no varía, como se indicó anteriormente. Además, comprobamos como la opción con la 1-Mejora tiene siempre menor número de consultas que la opción con rotación, en el peor caso posible. También podemos observar como la opción que utiliza las dos técnicas es la que tiene (con diferencia) el mayor número de consultas.

Por supuesto, una reducción del valor de *m*, en todos los casos, supone realizar un menor número de consultas, ya que consideramos menos imágenes candidatas a ser parecidas a la imagen de entrada, como podemos apreciar en la Tabla 7.12.

Esta última técnica, incluso en el caso más ideal de la Tabla 7.12, necesita 60 consultas para realizar una búsqueda de una imagen de 5 puntos (por ejemplo, un pentágono) e incluir en la búsqueda únicamente las imágenes que tengan de 3 a 7 puntos característicos ([5-2,5+2]). En el peor de los casos, para $n = 15$ y $m = 5$ necesita 525 consultas. Si tenemos en cuenta que cada consulta, por ejemplo, tarda en realizarse 1'5 segundos, la cuarta opción tardaría 1 minuto y 30 segundos en el mejor de los casos y nada menos que 13 minutos y 7 segundos en el peor de los casos reflejados en esta tabla.

Es por esto que, en la aplicación, impedimos que puedan seleccionarse conjuntamente ambas técnicas, ya que supone un incremento sustancial en el tiempo de respuesta y no es una gran mejora en el resultado final.

El hecho de hacer hincapié en que estamos analizando el peor caso posible es debido a la dependencia del tiempo de respuesta con las imágenes de la base de datos implicadas en la búsqueda. Si por ejemplo, realizamos una búsqueda con 1-Mejora de una imagen de 8 puntos, si no existe en la base de datos ninguna imagen de 7 y 9 puntos, el número total de consultas necesarias para la 1-Mejora es 0, porque no puede aplicarse. Otro caso es que, en una búsqueda con rotación para la misma imagen anterior, no existan imágenes con menos de 8 puntos, por lo que el número de consultas se reduce a *n* (hemos quitado *mn*), es decir, únicamente nos quedan las imágenes con un número de puntos característicos igual o mayor a la imagen de entrada.

7.2.1.2 Complejidad Algorítmica para el Modo de *Búsqueda por Patrón*

En cuanto a la *búsqueda por patrón*, ignoramos si la rotación o la 1-Mejora se hallan activadas, puesto que solamente existe una configuración:

$$N^{o}\, consultas = maxptos$$

donde *maxptos* nos indica el número de puntos de la imagen de la base de datos con mayor número de ellos y que participa en la búsqueda. Si buscamos en toda la base de datos, *maxptos* será relativa a todas las imágenes de la misma. Si buscamos en un grupo de imágenes concreto, *maxptos* será relativa a las imágenes de dicho grupo.

Esto es así porque el patrón que buscamos puede tener su máxima coincidencia con la imagen comparada en el primer punto de la imagen comparada, o en el segundo ... o en el punto *maxptos*-ésimo.

Por tanto, el número de consultas en la *búsqueda por patrón* es independiente del número de puntos característicos de la imagen de entrada y del valor del *margen menor* establecido en la aplicación.

7.2.2. Pruebas sobre el Tiempo de Respuesta en la Búsqueda

En el apartado anterior, hemos estudiado el número de consultas necesario para realizar una búsqueda en función de las técnicas seleccionadas. Además, hemos explicado el porqué de obviar la fase de obtención de datos de la imagen de entrada y de organización de las imágenes previa a las consultas, ya que no suponen un acceso intensivo a la base de datos (verdadero cuello de botella).

Pero, también adelantamos que una consulta con 3 puntos característicos no tarda el mismo tiempo en ejecutarse que una consulta con 15, ya que el S.G.B.D tiene que realizar muchas más comparaciones. Tampoco tardará lo mismo una consulta si tenemos 10 imágenes en nuestra base de datos, que si tenemos 100.

Por ello, vamos a mostrar los tiempos de respuesta obtenidos con distintas imágenes en la base de datos y con distintos valores de *m*. La máquina utilizada es un *Pentium IV Mobile* a 1'6 GHz con 256 Mb RAM.

Vamos a tener dos casos: base de datos con 52 imágenes (13 conjuntos de 4 imágenes cada uno). Cada conjunto se caracteriza por tener un número determinado de puntos característicos (3, 4, .., 15), es decir, tendremos en total 4 imágenes de 3 puntos, 4 imágenes de 4 puntos, ... y 4 imágenes de 15 puntos. El otro caso tiene 26 imágenes y cada conjunto tiene 2 imágenes. El tiempo de respuesta que se muestra en la Tabla 7.13 se mide desde que pulsamos el botón "*Buscar*" hasta que obtenemos los resultados, y su formato es *minutos:segundos,décimas*. Además se indica qué técnicas se han utilizado, así como el nombre de la imagen de entrada y su número total de puntos característicos.

Imagen (.bmp)	Nº Puntos	Con Rotación	Con 1-Mejora	Tpo. Respuesta 52 imágenes (m = 5)	Tpo. Respuesta 26 imágenes (m = 5)	Tpo. Respuesta 52 imágenes (m = 2)
P-82	5	-	-	0:06,9	0:05,7	0:04,3
P-82	5	√	-	0:17,3	0:16,0	0:16,9
P-82	5	-	√	0:18,6	0:16,4	0:17,4
51b	8	-	-	0:12,6	0:11,2	0:07,5
51b	8	√	-	1:20,2	1:09,2	0:41,3
51b	8	-	√	0:41,4	0:38,5	0:40,2
b-25	10	-	-	0:16,8	0:15,3	0:08,3
b-25	10	√	-	1:57,6	1:47,2	0:59,2
b-25	10	-	√	1:01,0	0:54,0	0:52,4
Baltimore	13	-	-	0:17,2	0:15,5	0:10,1
Baltimore	13	√	-	3:02,2	2:50,8	1:39,5
Baltimore	13	-	√	1:29,1	1:18,3	1:19,8

Tabla 7.13: Tiempo de respuesta en la búsqueda de imágenes. *m* indica el margen mayor y menor

Analizando la Tabla 7.13 en general, podemos apreciar como en todos los casos, la técnica simple arroja los mejores resultados en tiempo de respuesta, ya que realiza un número menor de consultas a la base de datos. Siguiendo a esta, aparece la técnica de 1-Mejora y, por último, la técnica de rotación. Sin embargo, en el caso de la imagen *P-82.bmp* (de 5 puntos), nos encontramos con que el tiempo de rotación es ligeramente inferior en todos los casos a los resultados obtenidos con la 1-Mejora. La razón es que, al no existir imágenes de menos

de 3 puntos, la técnica de rotación se ahorra 3 de los 5 grupos de imágenes a estudiar (los de 0, 1 y 2 puntos), sin embargo, la técnica de 1-Mejora no tiene más remedio que estudiar las imágenes de 4 y 6 puntos (que sí existen), lo que conlleva más tiempo.

Además, a medida que aumentamos *n*, es decir, el número de puntos característicos de la imagen de entrada, el tiempo de respuesta se incrementa paulatinamente, ya que la consulta resulta más compleja y el S.G.B.D necesita más tiempo para procesarla.

La segunda columna de *Tiempo de Respuesta* nos demuestra que el incremento de imágenes en la base de datos no supone un gran incremento en el tiempo de respuesta final, ya que las consultas a realizar son las mismas, aunque utilicen un número inferior de imágenes. Esto se aprecia comparando en la tabla las columnas primera y segunda de tiempo de respuesta. Hay que mencionar que Oracle© es uno de los SGBD comerciales más eficientes (si no el que más) del mercado.

Sin embargo, la tercera columna de *Tiempo de Respuesta* sí nos demuestra que si disminuimos el *margen mayor* y el *margen menor* de la aplicación, tendremos que estudiar un número inferior de combinaciones, por lo que el tiempo final de respuesta disminuye notablemente en el caso de rotación y en el resto de casos, el tiempo de respuesta se mantiene. Insistimos en que es el *margen menor* el responsable de este hecho, y no el *margen mayor*, porque su importancia es prácticamente nula, ya que las imágenes mayores van englobadas en una única consulta (o varias con rotación), mientras que las imágenes menores necesitan tantas consultas como número de puntos característicos tengan. Por esto, obtenemos mejores tiempos en la tercera columna de *Tiempo de Respuesta* que en la segunda (en el caso sin rotación y sin 1-mejora). En el caso de *m*=2 y 52 imágenes en la base de datos, analizaremos un total de 5*4 = 20 imágenes, mientras que en el caso de *m*=5 y 26 imágenes en la base de datos, analizaremos 11*2 = 22 imágenes.

La razón de que la rotación disminuya con un valor de *m* menor, pero la 1-Mejora no lo haga es porque la primera se ahorra analizar varias combinaciones, ya que solo analizamos las imágenes cuyo número de puntos se halle en el intervalo [n-2,n+2], mientras que la 1-Mejora sigue teniendo que analizar las imágenes con un punto más y con un punto menos que la imagen de entrada.

8. CONCLUSIONES Y LÍNEAS FUTURAS

En general, los resultados obtenidos en el capítulo anterior nos indican que, a pesar de la escasa información de descripción de la figura con la que contamos, la aplicación es capaz de comparar formas (sobre todo básicas) de manera bastante acertada. Sin embargo, sabemos que la aplicación podría mejorarse en muchos aspectos. En este capítulo, trataremos de resumir los aspectos de la aplicación más importantes y las posibles direcciones futuras a tomar en este campo de investigación, para lograr una recuperación de imágenes más sensible y real que la implementada con este proyecto, pero siempre a partir de la idea de este programa: recuperar imágenes de una base de datos utilizando atributos difusos y el lenguaje FSQL (*Fuzzy SQL*) para efectuar las consultas.

8.1 Características Principales de *FuzzyFinder 1.0*

En esta sección vamos a comentar aquellos aspectos más relevantes de este proyecto, con el fin de analizar lo que la aplicación nos permite hacer y los resultados obtenidos.

Como se ha mencionado anteriormente, los resultados son bastante buenos teniendo en cuenta la escasa información con la que se cuenta. Otros sistemas de recuperación de imágenes utilizan, además de la forma, otros descriptores como el color, la textura, la excentricidad, el área que ocupan.... Sin embargo, este sistema, caracterizando la figura a través únicamente de su forma, obtiene resultados interesantes como los mostrados en la Sección 7.1.

En cuanto a la segmentación de la imagen, únicamente comentar que hemos restringido el tipo de imágenes válidas para ser utilizadas en la aplicación. Como se indicó en su momento, el objetivo de este proyecto no es tanto la segmentación de la imagen, es decir, el aislamiento efectivo de la forma del fondo, sino más bien, que la aplicación sea capaz de identificar la imagen con ciertos puntos de su contorno que la caractericen de la mejor manera posible y que sirvan de descriptores para la comparación con otras imágenes que hayan sufrido este mismo proceso.

La característica fundamental de este proyecto es la utilización de bases de datos difusas como mecanismo de tratamiento de la imprecisión, puesto que la comparación de imágenes es un concepto claramente difuso e impreciso. Es por ello que las consultas tradicionales pasan a ser consultas difusas realizadas sobre una base de datos difusa, de manera que una comparación de imágenes se traduce a una o varias consultas FSQL. Esto conlleva las siguientes ventajas:

- No tenemos que realizar una comparación por cada par de imágenes, sino que, al representarla como una consulta en lenguaje FSQL, únicamente poniendo los valores de los atributos difusos de la imagen de entrada en la consulta, ésta nos devolverá el parecido de todas las imágenes de la base de datos con respecto a dicha imagen de entrada.
- Se utiliza un SGBD tradicional (Oracle©), sobre el que está implementado el servidor FSQL [Gali99]. Este servidor es el que nos permite efectuar las consultas difusas de manera similar a como se ejecutan en SQL. Por tanto, no necesitamos instalar un nuevo SGBD difuso, sino que únicamente necesitamos el servidor FSQL sobre un SGBD tradicional.

En general, la utilización del lenguaje de consultas difuso FSQL hace más *simple* y *eficiente* el proceso de comparación de imágenes.

Además, las imágenes son almacenadas en la base de datos difusa no como mapas de bits, sino que únicamente almacenamos los atributos de todos los puntos característicos de cada imagen, puesto que es la única información utilizada para establecer el parecido entre las imágenes. Esto ahorra espacio ya que ocupan menos las características utilizadas que la imagen propiamente dicha.

Por otro lado, existen una serie de propiedades deseables para cualquier sistema de recuperación de imágenes:

- **Reflexividad**. Si comparamos una imagen consigo misma, el resultado obtenido debe ser el máximo (es decir, 100%). La aplicación siempre es reflexiva, como muestran los resultados.
- **Diferenciabilidad**. Si dos imágenes son distintas, la comparación de ambas debe devolver un parecido menor que el máximo, es decir, menor que el 100%. La aplicación siempre cumple, por la forma de calcular los parecidos, la propiedad de diferenciabilidad.
- **Simetría**. Si al buscar una imagen *a* en la base de datos, obtenemos que se parece a *b* (una de las imágenes de la base de datos) en un parecido determinado, si buscamos ahora *b* (como imagen de entrada) y la imagen *a* está en la base de datos, entonces el parecido de *b* con *a* debe ser el mismo. La aplicación es siempre simétrica en todos los casos, por la forma de obtener el parecido de las imágenes, inclusive con la utilización de las técnicas de rotación y 1-Mejora.
- **Invariabilidad a traslación**. Aunque la figura se halle situada en la imagen en unas coordenadas distintas que otra imagen con la misma figura, si éstas son iguales, deben producir el máximo parecido. Esto se cumple siempre en la aplicación, gracias al proceso de obtención del contorno que localiza el primer punto del objeto en su primera fase.
- **Invariabilidad a tamaño**. Aunque la figura sea modificada de tamaño, si mantiene las mismas proporciones entre las distancias de los puntos característicos y los mismos valores de curvatura, el parecido debe ser el mismo. La aplicación, aunque no siempre devuelve el máximo parecido, siempre responde en estos casos con parecidos muy altos. La razón de por qué no obtiene el máximo parecido estriba en que la función de curvatura (ante cambios de escala) no devuelve siempre el mismo valor de curvatura en cada punto característico, mientras que las distancias sí las mantiene proporcionalmente iguales. A su vez, la función de curvatura no devuelve el mismo parecido ante cambios de escala por la propia naturaleza de una curva digital. Si, por ejemplo, reducimos el tamaño de una imagen, se eliminan puntos, de manera que perdemos información sobre la forma y, por tanto, exactitud en el contorno de la misma. Por ello, la función de curvatura devuelve resultados parecidos pero no siempre iguales.
- **Invariabilidad a rotación**. Si dos imágenes incluyen una misma figura, pero una de ellas se encuentra rotada cierto ángulo respecto a la otra, el parecido debe ser el

mismo, siempre que haya sido marcada esta opción en la búsqueda. De nuevo, aunque con valores altos de parecido, la aplicación no siempre devuelve el máximo por la misma razón de antes: la función de curvatura y, en última instancia, la propia naturaleza de una curva digital. Al encontrarse únicamente con ángulos entre puntos múltiplos de 45º, nos encontramos con ruido de cuantización que dicha función no solventa completamente, aunque, como ya se ha dicho y se ha mostrado en el capítulo anterior, se consiguen resultados bastante aceptables.

En cuanto al proceso de caracterización, la función de curvatura ha demostrado representar de forma acertada el contorno de la forma. Cumple las siguientes propiedades deseables:

- **Invariabilidad**. Si dos contornos son iguales, sus representaciones deben ser iguales.
- **Unicidad**. Si dos contornos son distintos, deben tener representaciones distintas.
- **Estabilidad**. Un pequeño cambio en el contorno debe producir un pequeño cambio en la representación.

Por otra parte, la aplicación *FuzzyFinder 1.0* tiene un alto grado de configuración, permitiendo adaptarse adecuadamente al tipo de información que deseamos extraer de cada imagen. Los siguientes parámetros son modificables desde el programa:

- *Sigma*, que nos permite obtener una función de curvatura más o menos suavizada, lo que nos permite establecer el tipo de puntos característicos obtenidos en etapas posteriores. Si, por ejemplo, la función es suavizada mucho, solamente permanecerán aquellos picos que correspondan a esquinas importantes de la forma, es decir, aquellas que impliquen a un número de píxeles del contorno considerablemente grande. Sin embargo, aquellas esquinas (aunque muy agudas) que estén formadas por un número reducido de píxeles del contorno, serán obviadas. En el caso contrario, si no suavizamos apenas la función de curvatura, cualquier pequeña esquina del contorno será considerada, en etapas posteriores, como punto característico de la forma. Este último caso podría sernos útil si deseamos obtener cualquier detalle de la forma por pequeño que sea.
- *Umbrales de Curvatura*, cuya configuración implica de manera directa el número de puntos característicos a obtener en la fase de *Extracción de Puntos Característicos*.

- Los *márgenes de curvatura y distancia* (necesarios para poder definir atributos difusos en FSQL) pueden ser modificados desde la aplicación, de manera que podemos concretar qué significa "aproximadamente igual" cuando utilizamos el comparador difuso FEQ. Un valor pequeño de estos márgenes nos distinguirá en mayor medida los valores de curvatura y distancia, mientras que un valor grande de estos márgenes hará más parecidos valores más diferentes de curvatura y distancia.

Además de estos, existen más parámetros configurables en la aplicación, como muestra la Figura 6.18.

Uno de los principales inconvenientes de la aplicación es que necesitamos igualar el número de puntos característicos de las imágenes que deseamos comparar. La eliminación de aquellos puntos con menor curvatura no es una solución óptima en todos los casos, puesto que puede que estemos eliminando precisamente aquellos puntos que tienen en común ambas imágenes. Para solventar esto, la técnica de 1-Mejora (véase Sección 4.2.1.1) nos permite decidir con un mejor criterio qué puntos debemos suprimir para maximizar el parecido, siendo únicamente válida (esta técnica) para imágenes con un punto característico de diferencia con respecto a la imagen de entrada, sabiendo que la técnica *d*-Mejora no puede ser implementada por razones de eficiencia (véase Sección 4.2.1.1).

Por otro lado, la *búsqueda por patrón* fue implementada como ejemplo del gran número de posibilidades que conlleva la utilización de comparaciones difusas en un sistema de recuperación de imágenes, no solamente con la utilización de otros comparadores difusos distintos de FEQ (ver siguiente sección), sino también eligiendo qué aspectos de la imagen deseamos comparar. En el caso de la búsqueda por patrón, no buscamos calcular la igualdad directa entre dos imágenes sino comprobar si el patrón que representa la imagen de entrada, aparece en algún lugar del contorno de alguna imagen de la base de datos. Un ejemplo de esto puede ser "buscar qué avión de los que existen en la base de datos tiene una cola parecida a la imagen de entrada".

La mayor parte de los resultados menos satisfactorios de este proyecto se deben al principal problema con el que se encuentra esta aplicación: **su fuerte dependencia al número de puntos característicos de las imágenes implicadas en la búsqueda**. Si, por

ejemplo, en cuanto a la función de curvatura hablábamos del problema de picos dobles o si, en cuanto a la extracción de puntos característicos, hablábamos de qué valores de umbrales proporcionar a la aplicación, era porque un error en alguno de estos aspectos nos lleva a obtener un número total de puntos característicos de la imagen distinto al esperado o al obtenido en otras imágenes similares, disparando la distinción entre ellas y arrojando, en definitiva, resultados menos buenos. Es decir, sabemos que el número de puntos característicos supone el mecanismo que nos permite establecer qué imágenes pueden ser comparadas con la de entrada y cuáles no pueden serlo. Además, si queremos comparar dos imágenes (que son muy parecidas) pero que difieren en el número de puntos característicos, esto supone un descenso de la sensibilidad de la aplicación, que establece como primer filtro que el número de puntos característicos sea similar.

8.2 Líneas Futuras de Desarrollo

En general, hemos dividido esta sección en dos apartados para poder distinguir entre mejoras puntuales (aunque puedan ser más o menos costosas en tiempo y esfuerzo) sobre algún módulo concreto de la aplicación (*Mejoras de la Aplicación FuzzyFinder 1.0*) y cambios notables sobre la base o el planteamiento de la recuperación de imágenes en general, aunque basándonos en las consecuencias del desarrollo de la aplicación *FuzzyFinder 1.0* (*Líneas Futuras de Desarrollo*).

8.2.1 Mejoras de la Aplicación FuzzyFinder 1.0

Vamos a comentar ciertas modificaciones que pueden realizarse en la aplicación *FuzzyFinder 1.0* para mejorar su funcionalidad. Algunas mejoras tienen suficiente complejidad como para ser, por sí mismas, otro proyecto como este o, incluso, en algunos aspectos son líneas de investigación abiertas en la actualidad.

- Permitir en la aplicación imágenes con más (o menos) de 256 colores de resolución.
- Permitir la utilización de imágenes en color en la aplicación.
- Utilizar un algoritmo de segmentación más potente que permita obtener el contorno de formas en imágenes con fondos no homogéneos y, en consecuencia, permitir

aumentar el conjunto de imágenes válidas en la aplicación. La opción más interesante puede ser aplicar en primer lugar el *Algoritmo de Canny* [Cann86] y, posteriormente, un algoritmo que complete los contornos no cerrados que devuelva dicho *Algoritmo de Canny*. De esta manera, evitaríamos la imposición de un fondo homogéneo.

- Utilizar una función de curvatura *invariable al tamaño* de las formas y que responda correctamente ante picos muy agudos, para evitar los problemas de picos dobles o picos con curvatura menor a la esperada que terminen produciendo un número de puntos característicos inesperado.
- Utilizar un módulo de cálculo de la función de curvatura que sea capaz, tras varias ejecuciones, de establecer el valor ideal de *sigma* para cada imagen. Lógicamente, este valor debiera ser consistente para las imágenes de la misma naturaleza, es decir, si por ejemplo un rectángulo se calcula con un *sigma* muy bajo, puede ocurrir que no se obtenga ningún punto característico y, otro rectángulo, calcularse con otro valor de este parámetro y obtener los cuatro puntos. Una solución, por ejemplo, podría ser la utilización del *espacio de escalas de la curvatura de una imagen* propuesto por Mokhtarian y Mackworth en [MoMa92].
- Modificar el módulo de cálculo de la función de curvatura para permitir extraer los puntos característicos sin basarse en los umbrales de curvatura o, al menos, no únicamente, es decir, que sea capaz, mediante algún tipo de análisis basado por ejemplo en la anchura de los picos, de determinar cuáles son los picos más representativos de la forma, sin tener en cuenta únicamente el valor de curvatura.
- Tener en cuenta a la hora de caracterizar una imagen, la existencia de huecos en ella, es decir, zonas con color del fondo pero embebidas en el interior del objeto. Podemos caracterizar cada hueco como si de un objeto se tratara, y por tanto nos encontraríamos con más atributos difusos (*NumHuecos* y todos los puntos característicos que describen cada hueco, esto es, *curvaturah*, *signoh* y *distanciah*).
- Considerar como opcional también la invariabilidad a traslación y a tamaño, por si deseamos buscar la forma con la misma localización y escala que la imagen de entrada.
- Podríamos realizar una modificación en la búsqueda por patrón. Cuando realizamos este tipo de búsqueda, pueden surgir puntos que no son del patrón, pero que aparecen al cortar el patrón de la imagen. Estos puntos podrían ser detectados y no considerados, aumentando notablemente la funcionalidad de este tipo de búsqueda.

8.2.2 Líneas Futuras de Desarrollo a partir de *FuzzyFinder 1.0*

Aunque también está basada en los resultados de la aplicación *FuzzyFinder 1.0*, esta sección trata de proponer determinadas alternativas que requieren muchas más modificaciones y, lo más importante, que suponen un cambio radical en el planteamiento de la recuperación de imágenes por parte de la aplicación.

- Utilizar otros descriptores (además de los ya existentes) a la hora de caracterizar la imagen (área, color, esqueleto de la forma, etc.). En general, no habría sólo que implementar todos estos descriptores (*nuevos atributos difusos*), sino además ponderar cada uno de los atributos difusos, según la importancia que éstos reflejen. Por ejemplo, el color de la forma debe ser un atributo menos importante a la hora de comparar imágenes que el contorno de la misma (una bombona y una naranja son del mismo color y no se parecen en nada).
- Diseñar un algoritmo utilizando la técnica de programación de *Ramificación y Poda* (como variante de la técnica de *Back-Tracking*) que nos permita comparar pares de imágenes de forma mucho más intensiva y detallada. Trataríamos de construir una de las figuras a partir de la otra (a partir de la figura con más puntos, tratar de construir la figura con un número menor de ellos), estableciendo como cota inferior una similitud (o parecido) determinado (*umbral de parecido*). Este umbral iría decrementándose paulatinamente hasta encontrar una combinación que incluya a todos los puntos de la imagen menor con un parecido mayor o igual al del umbral o hasta que este umbral sea igual a 0 (parecido igual a 0).

  La gran ventaja que se obtendría mediante esta técnica es que supondría una mayor exactitud en la comparación de imágenes y una menor dependencia con el número total de puntos característicos, pues esta técnica sabría encontrar la mejor combinación de puntos posible para maximizar el parecido, penalizándose en cualquier caso, la diferencia de puntos de forma adecuada.

Sin embargo, esta técnica desplazaría los atributos difusos de la base de datos a la aplicación, es decir, ya no necesitaríamos una base de datos difusa, sino que sería la propia aplicación la que compararía estos atributos difusos mediante algún procedimiento creado para tal efecto. La base de datos se utilizaría únicamente para almacenar los puntos característicos, y estos serían recuperados de una vez (una única consulta) y cargados en una estructura en memoria principal.

Además, con esta técnica, no podríamos utilizar un lenguaje de consultas difusas (como FSQL) que permite facilitar el proceso de acceso a la información sobre las imágenes, haciendo menos escalable aún la adición de nuevos descriptores de la forma, como se ha mencionado en el punto anterior.

Otro inconveniente de esta técnica es que, comparando imagen por imagen, los tiempos de respuesta dependerían fuertemente del número de imágenes implicadas en la búsqueda, lo que augura tiempos de respuesta bastante pobres.

- Respecto al uso de FSQL, se podrían utilizar otros comparadores además de *FEQ*, como *NFEQ* y podrían usarse distintos umbrales para cada tipo de atributo. Así se puede, por ejemplo, dar más importancia a la curvatura que a la distancia entre puntos, o viceversa.

REFERENCIAS

Referencias bibliográficas

[ArGa98]	M.C. Aranda, J. Galindo. "*Clasificación de Imágenes de una Base de Datos utilizando información de su forma*". IV Jornadas Internacionales de Informática. Las Palmas de Gran Canaria (España). 1998.
[BaFu97]	J.R. Bach, C. Fuller et al. "*The Virage Image Search Engine: An Open Framework for Image Management*". SPIE Storage and Retrieval for Image and Video Database. 1997
[Benz76]	J.P. Benzécri et coll. "*L`analyse des données*"; Tomo I: "*La Taxinomie*"; Tomo II : « *L`analyse des correspondences* » . Paris, Dunod. 1976**.**
[Cann86]	J. Canny. "*A computational approach to edge detection*". IEEE Transactions on PAMI, 8(6): 679-698. 1986.
[Ceba98]	Fco. Javier Ceballos. "*Visual C++: Aplicaciones para Win32*". Ed. Ra-Ma 1998.
[Dowe93]	J. Dowe. "*Content-based Rretrieval in Multimedia Imaging*". SPIE Storage and Retrieval for Image and Video Database. 1993
[DuPr80]	D. Dubois, H. Prade. "*Fuzzy Sets and Systems: Theory and Applications*". Academic Press, NY. 1980.
[DuPr85]	D. Dubois, H. Prade. "*Fuzzy Number. An Overview, the Analisis of Fuzzy Information*". J.C. Bezdek CRS Press, Boca Raton F1. USA.1985.
[DuPr88]	D. Dubois, H. Prade. "*Possibility Theory. An Approach to Computerized Processing of Uncertainty*". Plenum Press, NY. 1988.
[DuPr95]	D. Dubois, H. Prade. "*A Review of Fuzzy Set Aggregation Connectives*". Information Sciences, 36, pp. 85-121. 1995.
[ElNa00]	R. Elmasri, S.B. Navathe. "*Fundamentals of Database Systems*". Third Edition. Addison-Wesley. 2000.
[Esco03]	Calixto Escobar Rodriguez. "*Software para Control Difuso de todo tipo de Sistemas*". Proyecto Fin de Carrera. Universidad de Málaga. 2003
[GaDi01]	G. Galeano, P. Díaz, J.C. Sánchez. "*Manual imprescindible de HTML 4*". Ed. Anaya Multimedia. 2001.
[Gali99]	J. Galindo. "*Tratamiento de la Imprecisión en Bases de Datos Relacionales: Extensión y Adaptación de los SGBD Actuales*". Tesis Doctoral. Universidad de Granada. 1999
[GaMe00]	J. Galindo, J.M. Medina, J.M. Rodríguez. "*Comparadores para Bases de Datos Difusas: Definiciones, Clases y Relaciones*". X Congreso Español sobre Tecnologías y Lógica Fuzzy (ESTYLF'00), pp. 187-192. España. 2000
[GaMe98]	J. Galindo, J.M. Medina, A. Vila, O. Pons. "*Fuzzy Comparators for Flexible Queries to Databases*". Iberoamerican Conference on Artificial Intelligence, IBERAMIA'98. Lisboa (Portugal), pp. 29-41. 1998.
[GoWo96]	R. González, R. Woods. "*Tratamiento Digital de Imágenes*". Ed. Addison-Wesley/Diaz de Santos. 1996.
[KrGe94]	R. Kruse, J. Gebhardt, F. Klawonn. "*Foundations of Fuzzy Systems*". John Wiley & Sons. 1994.
[McTh94]	F.M. McNeill, E. Thro. "*Fuzzy Logic: A practical approach*". AP professional. 1994.
[Medi94]	J.M. Medina. "*Bases de Datos Relacionales Difusas. Modelo Teórico y Aspectos de su Implementación*". Tesis Doctoral. Universidad de Granada. 1994.
[MoMa86]	F. Mokhtarian, A. Mackworth."*Scale-Based description and recognition of planar curves and two-dimensional shapes*". IEEE Trans. Pattern Analysis and Machine Intelligence, 8. 1986.
[MoMa92]	F. Mokhtarian, A. Mackworth."*A Theory of Multiscale, Curvature-Based Shape Representation for Planar Curves*". IEEE Trans. Pattern Analysis and Machine Intelligence, 8. 1992.
[MoVa93]	J. Mohammd, N. Vadiee, T.J. Ross. "*Fuzzy Logic and Control. Software and Hardware Applications*". Eaglewood Cliffs, NJ:PTR. Prentice Hall. 1993.

[NgSe96]	R. Ng, A. Sedighian. "*The QBIC Project: Querying Images by Content Using Color, Texture and Shape*". SPIE Storage and Retrieval for Image and Video Database. 1994
[PeFe93]	N. Pérez de la Blanca, J. Fernández Valdivia. "*Characterizing Planar Outlines*". Pattern Recognition Letters 14. pp. 489-497. 1993
[PeGo98]	W. Pedrycz. F. Gomide "*An Introduction to Fuzzy Sets: Analysis and Design*". Ed. The MIT Press. 1998.
[Pere02]	César Pérez. "*Oracle 9i: Administración y Análisis de Bases de Datos*". Ed. Ra-Ma. 2002.
[Petr96]	F.E. Petry. "*Fuzzy Databases: Principles and Applications*" (with chapter contribution by Patrick Bosc). International Series in Intelligent Technologies. Ed. H.-J. Zimmermann. Lkuwer Academic Publishers (KAP). 1996.
[PrTe84]	H. Prade, C. Testemale. "*Generalizing Database Relational Algebra for the Treatment of Incomplete/Uncertain Information and Vague Queries*". Information Sciences, 34, pp. 115-143. 1984.
[RuHu99]	Y. Rui, T.S. Yuang, S. Chang. "*Image Retrieval: Current Techniques, Promising Directions and Open Issues*". Journal of Visual Communication and Image Representation 10, 39-62. 1999.
[Scha89]	R. Schalkoff. "*Digital Image Processing and Computer Vision*". Ed. John Wiley & Sons, Inc. 1989.
[ScSk83]	B. Schweizer, A. Sklar. "*Probabilistic Metric Spaces*". North-Holland. 1983.
[SoHl99]	M. Sonka, V. Hlavac, R. Boyle. "*Image Processing, Analysis and Machine Vision*" Ed. Thompson Computer Press, London. 1999.
[Tril79]	E. Trillas. "*Sobre Funciones de Negación en la Teoría de Conjuntos Difusos*". Stochastica, Vol. 3 Nº 1, pp. 47-59. 1979.
[Ullm82]	J.D. Ullman. "*Principles of Database Systems*". Computer Science Press, 2nd edition. 1982.
[Yage80]	R.R. Yager. "*On a General Class of Fuzzy Connectives*". Fuzzy Sets and Systems, e, pp. 235-242. 1980.
[Yage91]	R.R. Yager. "*Connectives and Quantifiers in Fuzzy Sets*". Fuzzy Sets and Syst., 40, pp. 39-76. 1991.
[YaOv87]	R.R. Yager, S. Ovchinnikov et al. "*Fuzzy Sets and Applications: Selected Pappers by L. A. Zadeh*". Wiley Intersc. 1987.
[Zade65]	L. Zadeh. "*Revista científica Information and Control*". Artículo Fuzzy Sets. 1965.
[Zade65b]	L.A. Zadeh. "*Fuzzy Sets*". Information Control, 8, pp. 338-353. 1965.
[Zade75]	L.A. Zadeh. "*The Concept of a Linguistic Variable and Its Application to Aproximate Reasoning*". Information Sci., 8, pp. 199-248, pp. 301-357, 9, pp. 43-80. 1975.
[Zade78]	L.A. Zadeh. "*Fuzzy Sets as a Basis for a Theory of Possibility*". Fuzzy Sets and Systems, 1, pp. 3-28. 1978.
[Zade92]	L. Zadeh. "*Knowledge Representation in Fuzzy Logic*". Cap. 1. Ed. Kluwer Academic Publisher. 1992.
[Zimm91]	H.-J. Zimmermann. "*Fuzzy Set Theory and its Applications*". 2nd Edition. Ed. Kluwer Academic Publishers (KAP), 1991.