UNIVERSIDAD NACIONAL AUTONOMA DE MÉXICO

FACULTAD DE ESTUDIOS SUPERIORES ACATLÁN

PROCESAMIENTO DE IMÁGENES DIGITALES DEL IRIS PARA SU IDENTIFICACIÓN A TRAVÉS DE PROGRAMAS EN C++ Y LA LIBRERÍA LIBRE SDL

TESIS

QUE PARA OBTENER EL GRADO DE

LICENCIATURA EN MATEMÁTICAS APLICADAS Y COMPUTACIÓN

PRESENTA:

JAFET MALVÁEZ LÓPEZ

ASESOR: DR. LÓPEZ GÓMEZ ANGEL

MÉXICO 2008

A MI MADRE porque siempre me impulsó a seguir adelante y es un ejemplo de que siempre se puede llegar a donde uno desea si se tienen las ganas.

A MINHA MENINA (Coral) porque gracias a ella pude madurar en otros aspectos y es la fuente de mi inspiración para llegar a ser una gran persona. Porque gracias a ella he visto que mi propia fuerza es más grande de lo que imaginé.

A MI ASESOR porque me ayudo a seguir adelante en esto aun cuando parecía que se complicaría.

A MI ABUELO porque siempre confía en nosotros y cree en nuestra capacidades para llegar lejos.

A MIS AMIGOS porque están ahí cuando uno menos se lo espera, por su apoyo y porque todo este tiempo me han dicho que he llegado a ser un ejemplo porque siempre termino lo que empiezo.

A TODOS MIS DEMÁS FAMILIARES que siempre confiaron y se sienten orgullosos de mí.

A JAFET porque sin él no habría podido ser quien siempre he querido ser.

Índice

Resumen

En este trabajo se presenta la programación de la metodología aplicando morfología matemática difusa a la configuración de códigos biométricos del iris del ojo humano como mecanismo de identificación y reconocimiento haciendo uso del lenguaje de programación C++ en su versión de WxDevcpp 10.1.

El Capítulo 1, presenta el marco teórico sobre el que se sustenta el planteamiento general del trabajo. El Capítulo 2 desglosa la metodología propuesta de integración al algoritmo para su aplicación en el reconocimiento del iris. El Capítulo 3 describe la metodología de proceso de la imagen del ojo hasta obtención del número binario y el código de barras. El Capítulo 4 presenta la aplicación práctica del algoritmo extendido, incluyendo diferentes combinaciones de las condiciones iniciales que caracterizan el comportamiento del algoritmo. Al final se presentan las conclusiones y trabajos futuros.

Introducción

La seguridad se ha convertido en un tema de interés popular. Conforme la tecnología avanza, las medidas de seguridad se hacen más estrictas, eficientes y necesarias. Pero así como se van volviendo parte de nuestra vida diaria, se requiere buscar alternativas para que toda la gente pueda tener acceso a las diferentes formas para proteger sus intereses y datos personales.

Incluso las empresas hacen uso de sistemas de seguridad para restringir a sus trabajadores el acceso a áreas o información. Y no solo eso, sino que también para controlar sus entradas y salidas para comprobar que efectivamente llegan a la hora adecuada y cumplen con sus turnos. Ya sea mediante el uso de tarjetas de identificación o alguno de los sistemas que verifican la morfología de los empleados.

La seguridad a través de métodos que permiten verificar la biometría de las personas es uno de los campos que hace que la seguridad a este nivel tenga un costo muy elevado, ya que los dispositivos biométricos resultan onerosos y/o resulta imposible la implementación libre ya que están patentados o protegidos por alguna ley de autoría. Es por ello que este trabajo retoma y amplía la "Propuesta Alterna del Reconocimiento del Iris Usando Procesamiento Morfológico Borroso" del ahora maestro "Vicente Magaña González", [MAGAÑA, 2005] para desarrollar un programa que da uso a esta alternativa tecnológica.

En [MAGAÑA, 2005] se presenta un análisis del iris y su reconocimiento por medio de simples imágenes y aplicando morfología matemática difusa. Haciendo uso del trabajo inicial expuesto por Daugman [DAUGMAN, 2003] ofrece una propuesta alterna a la descrita salvando los derechos de patente que conserva Daugman. En contraste el trabajo de investigación que se presenta describe la utilización del método general resuelto a través del uso de un leguaje de programación de uso general como es C y evitando de este modo la versión donde se utilizó Matlab como herramienta de simulación.

A partir del análisis, utilizando como herramienta de programación el entorno de desarrollo integrado (IDE) wxDevC++ y la librería SDL para permitir el procesamiento continuo, aplicando morfología matemática difusa a imágenes digitalizadas del iris es posible la obtención de un código binario de 87 bits para identificación y reconocimiento biométrico.

La versión localizada en [MAGAÑA, 2005] se encuentra dividido en varios módulos los cuales no están conectados entre si, es decir, se tiene que iniciar cada una de las etapas consecutivas manualmente. Esto origina que el proceso sea lento y desde el punto de vista práctico es inoperante lo cual lo hace inconveniente. Mediante una programación en C++ tenemos los módulos integrados unificados en un solo programa lo que da oportunidad de realizar todos estos cálculos de manera continua, lo que agiliza el proceso.

Básicamente el método consiste en tomar una imagen del ojo y digitalizarlo (las imágenes que utiliza este método son en escala de grises), esto mediante una cámara digital (nuestro trabajo base propone una cámara CCD de 2.1 mega píxeles[1]). El siguiente proceso es hacer el análisis para encontrar el centro de la pupila (encontrar tanto la línea de píxeles horizontal como la vertical más grande) y remover esa sección incluyendo toda la basura[2] que se encuentre. Se prosigue encontrando el límite exterior del iris, posteriormente se necesita remover toda el área externa, esto por medio de un proceso similar al anterior. Lo que se obtiene con estos procesos es una imagen del iris (una imagen en forma de dona, ver detalles en Capítulo 3), sin ningún elemento que la distorsione.

Al extraer únicamente el área del iris, lo que procede es la transformación de la imagen resultante de coordenadas polares a cartesianas, lo que nos produce una nueva imagen en forma rectangular. Esta imagen tendrá los mismos elementos y propiedades que su forma original (en el proceso se puede perder un porcentaje no representativo de la imagen). Esto facilita obtener un código de barras y su identificación por comparación con otros que se mantengan en una base de datos. El código binario final se obtiene al aplicar las ondeletas de Haar [SCIELO, 2006], es decir, dividir nuestra imagen transformada y validar el color que tiene (blanco = 1 y negro = 0).

En el capítulo 1, Procesamiento de imágenes, se localiza una introducción al manejo de imágenes, así como los distintos formatos usuales que se tienen (jpg, gif, etc.). Se incluye un apartado que nos habla de Biometría y los diferentes métodos de seguridad mediante un análisis morfológico de alguna de las partes del cuerpo, hasta llegar al reconocimiento del iris. Describe en forma general los distintos sistemas de seguridad que aplican morfología, análisis del rostro, palma de la mano, huella digital, voz, escritura entre otros.

El capítulo 2, Análisis del método propuesto, explica el método desarrollado por el Maestro Vicente. Para ello se hace uso de definiciones como morfología, lógica difusa y posteriormente morfología difusa. Estos son los principales fundamentos metodológicos y teóricos que forman la base de dicho método. También se explica el desarrollo del método paso a paso, como son extracción del iris, transformación, obtención de código.

En el capítulo 3, Planteamiento del problema, se explica las adecuaciones al análisis descrito en el capítulo 2 con la ventaja de obtener un mayor beneficio, desde el punto de vista de continuidad e integración al remplazar la codificación en Matlab por una en lenguaje C++ con presentación gráfica.

Finalmente el capítulo 4 detalla los resultados obtenidos al desarrollar este nuevo sistema, contrastando los resultados de ambas investigaciones y evaluando los tiempos de ejecución en términos generales.

---

[1] El píxel (del inglés *picture element*, o sea, "elemento de la imagen") es la menor unidad en la que se descompone una imagen digital, ya sea una fotografía, un fotograma de vídeo o un gráfico [WIKIMEDIA_P, 2007].

[2] Definimos como basura a todos los píxeles que no se encuentran relacionados a la sección deseada.

Contribución:

Mejorar el método para obtener una identificación de una persona a través del iris. Debido a que la forma en que fue desarrollado, en diferentes módulos, origina que el método no tenga una rapidez adecuada.

Lo expuesto en este trabajo es unir esos módulos y reutilizar estos para que trabajen en conjunto y aplicando una herramienta que no requiere licencia ni estar previamente instalado en la computadora donde es ejecutado, con esto no tendremos que esperar a que termine un proceso para inicializar el siguiente módulo, ya que de esta manera comenzará el siguiente proceso de forma automática. Lo que nos permitirá agilizar la salida de resultados y no depender de un ambiente de soporte ajeno al programa.

Capítulo 1

Conceptos generales de procesamientos de imágenes y biometría

El presente capítulo introduce al procesamiento de imágenes, así como a los diferentes tipos o formatos comunes que existen, un comparativo entre ellos y la unidad mínima para trabajar con las imágenes que es el píxel. De igual manera se hace mención a la biometría como una herramienta de seguridad, los diferentes sistemas biométricos más empleados, entre los cuales se encuentra el análisis del iris.

1.1. Procesamiento de imágenes

Con el uso de los sistemas digitales el hombre ha podido hacer uso de imágenes de diversos objetos para realizar análisis de fenómenos y elementos de una manera más detallada. Por ejemplo, el uso de imágenes satelitales para ver la trayectoria de alguna tormenta tropical, para observar alguna zona de desastre, para identificar un rostro, con distorsiones o incompletos, así como identificar las placas de un automóvil o algún otro evento de este tipo.

También el análisis de imágenes nos permite encontrar e identificar patrones, lo cual para este caso, nos resulta de utilidad ya que para el reconocimiento del iris buscaremos elementos individuales irrepetibles (se sabe que cada persona tiene un patrón distinto de iris) que nos permita obtener un código de barras u otro código basado en información binaria personalizada. El cual, como se ha mencionado es comparado contra otro existente para autenticar a un usuario (persona) como una de sus posibles aplicaciones.

1.1.1. Conceptos básicos

Una imagen puede almacenarse en un fichero o arreglo de bytes, reconociendo a éstos como una secuencia posicional de 8 elementos binarios, siguiendo diferentes formatos. Algunos de ellos resultan ser mas complejos que otros, también muchos utilizan compresión de datos lo que permite tener una buena resolución en un archivo que ocupe un menor espacio en disco, tal es el caso entre bmp y jpg. Cada uno tiene sus ventajas y sus desventajas, pero todos ellos tienen algunas características en común [SIEUN, 2007].

Todos los formatos llevan una cabecera en el fichero, lo cual sirve para identificar el tipo y formato de organización de fichero que se utiliza, y además contiene toda aquella información necesaria para poder leer el fichero, es decir, nos dice como es el tamaño de la imagen o el número de colores que tiene cada píxel entre otros detalles de la imagen.

Posteriormente dentro del fichero se encuentran comprimidos mediante algoritmo específico de ese formato, los datos de la imagen. Los cuales al ser descomprimidos ofrecen la información de los elementos constitutivos de una imagen digitalizada que se conocen como píxeles, identificando su posición como pareja ordenada del espacio que representa la imagen y sus características individuales como es color.

Dependiendo de las características de la imagen y el tipo de formato cada una de ellas tendrá más o menos colores, según lo que este representando. De esta forma entre más sea el número de colores mayor es el número de bits necesarios para su representación. Es decir, a mayor número de bits más amplia es la gama de colores. Por otro lado, variando la cantidad de parejas ordenadas que representan al conjunto de píxeles en cada uno de los ejes, tendremos distintas calidades o resoluciones en la imagen que se este observando.

Normalmente los colores (en computación) van de los 16, o 256, hasta los 16 millones, lo que requiere 4, 16 ó 24 bits por píxel [SIEUN, 2007]. En el caso de utilizar 16 ó 256 colores, debe especificarse la cantidad exacta que corresponde a cada uno de esos colores, necesitamos saber que cantidades de Rojo, Verde y Azul realizan la combinación exacta para poder obtener el color ideal a representar en pantalla. Para definir el color que buscamos existe la llamada paleta de colores, que no es más que una tabla la cual nos permite definir la cantidad de los colores Rojo, Verde y Azul que son necesarias para llegar al color por combinación de los colores primarios. Puede ser modificada en función de la imagen, por lo que es necesario guardarla en el fichero. En el caso particular de utilizar 16 millones de colores, no se utiliza paleta de colores, pues la relación entre número de color y cantidad de Rojo, Verde y Azul es implícita: De los 24 bits por píxel, 8 especifican la cantidad de Rojo, otros 8 la cantidad de Verde, y otros 8 la de Azul. En la figura 1.1 se presenta una forma ampliada de lo que constituye el concepto de pixel.

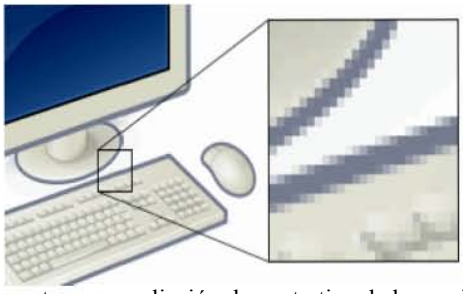

Figura 1.1 Representa una ampliación demostrativa de la apariencia de un píxel.

1.2. Imágenes y formatos

Existen diversos formatos para las imágenes y cada una tiene sus respectivas características y algoritmos de compresión, algunas llegan a comprimir más los datos que las otras, así como variación en la definición que reflejan como calidad de su presentación. Y cada uno de estos formatos se emplea mejor dentro de ciertas circunstancias que las otras (fotografías, gráficas, dibujos, etc.) y podemos estar de acuerdo en que siempre encontraremos el formato adecuado con el que podemos mostrar lo que deseemos.

1.2.1. Formato JPG

Este formato (.jpg / .jpeg), a pesar de que consta de un complejo algoritmo de compresión sufre de perdida de información. Este algoritmo modifica ligeramente los datos para lograr la mejor compresión. Para imágenes escaneadas resulta ser imperceptibles, pero cuando se trata de dibujos, aparecen puntos los cuales si resultan ser visibles. Por ello este formato es bueno y útil para imágenes escaneadas pero no para dibujos. Tampoco nos permite definir colores como transparentes.

1.2.2. Formato PCX

Este es el formato conocido de Paintbrush. Este formato utiliza el algoritmo conocido como RLE. Este algoritmo se encarga de reemplazar las secuencias de N píxeles consecutivos y del mismo color por dos bytes, donde el primero nos dirá cual es la cantidad de N que tenemos y el segundo nos dirá de que color se trata. Si tenemos trabajando los 256 colores tendremos un píxel por cada color. Ahora necesitamos diferenciar cuando se nos indique el color o las repeticiones, para ello utilizamos el siguiente criterio, donde si el byte es inferior o igual a 192, se trata de un byte para color de un único píxel. Cuando el byte sea mayor a los 192, poniendo a 0 los dos bits más significativos nos indica que se trata del número de repeticiones de píxeles y el byte siguiente nos determinará el color (aun si resulta ser superior a los 192).

La compresión de este formato permite reducir el dibujo cuando este contenga varias secuencias iguales y las repeticiones de colores sean abundantes. Ya que tendremos muchas repeticiones de bytes. En dado caso que algún píxel de color sea menor que 192 (aislado), es reemplazado por los bytes 193 lo que puede repercutir en el aumento del fichero. Ejemplo de ello pasa cuando tenemos imágenes escaneadas, ya que resulta improbable que estas contengan píxeles consecutivos del mismo color.

1.2.3. Formato BMP

Este formato perteneciente a Windows (Windows BitMaP, Bitmapped File Format) resulta ser uno de los ficheros para imágenes más simples que existe. Aun cuando éste permite la compresión, no se usa. Consta básicamente de la cabecera y posteriormente de los valores de cada píxel. Esta descripción de píxeles empieza por el último y finaliza con el primero, además que se recorre de izquierda a derecha. Este parece ser el formato preferido por Bill Gates [SIEUN, 2007]. La real ventaja de este fichero es su enorme sencillez, pero su gran desventaja ante los otros es el gran tamaño que alcanzan los ficheros.

1.2.4. Formato GIF

Este formato de fichero para imágenes fue inventado por Compuserve. GIF se refiere a Graphic Interchange Format. Dicho formato utiliza un algoritmo que basa su estructura en programas de compresión convencional. De manera formal utiliza un algoritmo de compresión LZW modificado, el cual resulta ser mucho más complejo que el RLE. Este además de detectar las repeticiones en los colores, también se encarga de buscar todas aquellas secuencias mediante un diccionario, el cual se construye en ese momento. Para este formato su principal aplicación al igual que los PCX es en los dibujos, en los cuales se tienen series de puntos con el mismo color, en donde las series de colores tienden a

repetirse. Esto no significa que no se pueda utilizar para algún otro tipo de imágenes como fotos o imágenes escaneadas [SIEUN, 2007].

En la Tabla 1.1 se muestran en forma compacta las características que distinguen a los tres principales formatos.

Tabla 1.1 Establecen las diferencias de los formatos más usados para la web, jpg, gif y png.

JPG	GIF	PNG
• Número de colores: 24 bits color o 8 bits B/N • Muy alto grado de compresión. • Formato de compresión con pérdida. • No permite trasparencia • No permite animación.	• hasta 256 colores • Formato de compresión sin pérdida. • Carga progresiva • Permite transparencia • Animación simple	• Color indexado hasta 256 colores y no indexado hasta 48 bits por pixel. • Mayor compresión que el formato GIF (+10%) • Compresión sin pérdida. • Transparencia variable. • No permite animación.

1.2.5. Formato PNG

Este formato Portable Network Graphic resulta ser un formato de mapa de bits de libre distribución, como una alternativa para los formatos anteriores dentro de la web (GIF, JPG) que es valido tanto para PC como para MAC. Este formato nació de las manos de la empresa Unisys Corporation, la cual busca sustituir principalmente al GIF, ya que este nuevo resulta ser más simple, pero menos completo.

PNG utiliza un esquema de compresión sin pérdidas para reducir el tamaño del archivo, manteniendo intacta la calidad original de la imagen. Puede trabajar en modo Escala de Grises (con un canal alfa), en modo Color Indexado (8 bits, hasta 256 colores, paletas de colores) y en modo RGB (24 bits, 16,8 millones de colores y 48 bits, con 24 bits para canales alfa), por lo que admite 256 niveles de transparencia [MORENO, 2005].

Aun cuando permite el manejo de imágenes entrelazadas, es decir, de visualización progresiva y detección de errores, la real desventaja que tiene ante el formato GIF, es que PNG no soporta animaciones. Además los ficheros generalmente tienen un tamaño mayor que el de los formatos GIF y JPG, además de que inexplicablemente colores (fondos principalmente) se ven afectados, lo que resulta ser un aspecto negativo.

1.2.6. Otros

Además de los mencionados anteriormente existen otros tipos de formatos que son empleados para gráficos vectoriales e imágenes empleadas en animaciones como en Macromedia Flash (cuya extensión para este formato .swf, Shockwave Flash) y el w3c (.svg, Scalable Vector Graphic), los cuales han sido empleados dentro de páginas web desarrollando vistas con formato de muy alta calidad. Estos pueden ser banners animados, mapas y planos

1.3. Biometría

Demos un paso pequeño dentro de la historia de la biometría, la cual se remonta al occidente a finales del siglo XIX copiando métodos que ya se usaban en oriente desde el siglo XIV. Ya que los chinos empezaron a diseñar métodos para identificar entre los niños jóvenes. Recurrían a hacer estampados de la huella de la mano de los niños con tinta.

En occidente era común el uso de la "memoria fotográfica", hasta que el jefe de la policía de París (Alphonse Bertillon) desarrollo un sistema antropométrico conocido también como Bertillonage. Este se basaba en la anchura de ciertas partes del cuerpo, así como de la cabeza, de igual forma se tomaban registros de los tatuajes y cicatrices que tenían los criminales. Este método fue adoptado por mucho tiempo en occidente hasta que empezaron a aparecer errores debido a los distintos sistemas de medidas. Posteriormente la policía occidental empezó a utilizar el método de tomar la impresión de la huella dactilar (similar al que utilizaban los chinos).

En estos últimos años la biometría ha crecido desde usar simplemente la huella dactilar, a emplear muchos métodos distintos teniendo en cuenta varias medidas físicas y de comportamiento. Las aplicaciones de la biometría también han aumentado - desde sólo identificación hasta sistemas de seguridad y más [WIKIMEDIA, 2007].

1.3.1. Principios de la biometría

De una manera clásica podemos definir a la biometría como la ciencia que se dedica al estudio estadístico de las características cuantitativas de los seres vivos [TAPIADOR, 2005]. Recientemente dicho término está siendo usado para referirse a los métodos automáticos que permiten el análisis de las características humanas, cuyo objetivo es el de identificar a una persona.

Formalmente biometría es la técnica dirigida a obtener y tomar medidas individuales del hombre sobre todas las características para poder orientadas al uso de las tecnologías, sean: digitalizado, automatizada, robotizada, creando modelos matemáticos, funcionales e informáticos [DIMITRIE, 2005].

El contexto tecnológico de la palabra biometría se refiere a la aplicación automatizada de técnicas biométricas a la certificación, autentificación e identificación de personas en sistemas de seguridad. Las técnicas biométricas se utilizan para medir características corporales o de comportamiento de las personas con el objeto de establecer una identidad. Para diferenciar estos conceptos, organizaciones y autores han dado un nombre compuesto al contexto tecnológico como biometría informática y autentificación biométrica [GONZÁLEZ, 2005].

Podemos decir que la biometría forma parte del mundo de la criptografía y de la seguridad informática [TAPIADOR, 2005] y a su vez podemos clasificarla dentro de uno de los tres niveles en la que tradicionalmente se diferencian los sistemas de seguridad. Los cuales son:

1. Algo que el usuario sabe, como lo es una contraseña.
2. Algo que el usuario tiene, esto puede ser alguna llave o tarjeta personal.
3. Algo que el usuario es, se refiere a alguna forma de identificar a la persona a través de alguna seña particular, alguna característica física propia de esa persona.

Es necesario que las características físicas y conductuales cumplan con ciertos requisitos los cuales les permitirán ser utilizadas como elementos de identificación [TAPIADOR, 2005]:

- *Universalidad:* Ya que todas las personas deben contar esta característica. Como ejemplo podemos tomar las huellas digitales ya que todas las personas cuentan con sus propias huellas.
- *Singularidad:* Pues dos personas deben tener sus propias distinciones dentro de la misma característica, esta distinción debe ser suficiente para poder identificarlas.
- *Estabilidad:* Dicha característica debe ser bastante estable como para no modificarse durante el transcurso del tiempo y soportar las diversas condiciones ambientales.
- *Aceptabilidad:* Las personas deben aceptar que dicha característica puede ser parte de un sistema de identificación biométrico.
- *Rendimiento:* Se debe tener un alto nivel de exactitud para que la característica sea aceptada.
- *Usurpación:* Debe de ser capaz de determinar el grado en el que se puede disfrazar la identidad de una persona para sustituir a otra y violar la seguridad.

1.3.1.2. Comparación de la biometría con otras técnicas

Lo que hace que la biometría sea más eficaz frente a otras tecnologías es que es algo que el individuo siempre es y por ende que es difícil se desprenda de él y que se pierda, en especial por extravío o por olvido, como suele suceder con las tarjetas de identificación, generadores de claves e incluso las mismas contraseñas.

Cuando se trata de objetos que nos dan el acceso suele suceder que se extravían, ya sea mientras se viaja o en el lugar donde se ha hospedado y algunas veces se olvidan en el último lugar de uso. Así mismo suele pasar que para mayor comodidad se llevan dichos dispositivos y tarjetas junto a maletines con documentos importantes y computadoras portátiles lo cual resulta en que si se extravían también perderemos nuestras llaves de acceso.

En lo que se refiere a contraseñas lo común es que se cambien las contraseñas las cuales se tiende a olvidarlas o hacer el uso de generadores de claves las cuales son poco usuales. Es por ello que la gran mayoría de las veces las personas tienden a usar claves que son muy fáciles de recordar como cumpleaños, fechas representativas y nombres que representen algo para dicha persona. Estas claves suelen ser descifradas con mayor facilidad por otros individuos que conozcan los gustos y cierto tipo de datos de otro. De igual manera existen programas denominados de fuerza bruta cuya función es encontrar la clave al hacer uso de diccionarios o combinaciones de letras al usar poderosos algoritmos para generar palabras hasta poder encontrar la clave correcta.

1.3.2. Sistemas biométricos

A través de las diferentes tecnologías que el hombre ha desarrollado, ahora somos capases de poder identificar a un individuo mediante algún signo corporal (entiéndase alguna parte del cuerpo o comportamiento), esto con el motivo de darle acceso a información o algún otro tipo de bien. Para ello se han diseñado sistemas biométricos que nos permiten mediante algoritmos matemáticos analizar patrones (como en iris y huellas digitales) para compararlos con otro haciendo uso de hardware que tomará la muestra.

1.3.2.1. Elementos de un sistema biométrico

Se entiende por tecnologías biométricas a todos aquellos métodos automatizados para la identificación y verificación de la identidad de un ser humano (viviente) que se basa en sus características fisiológicas, así como de comportamiento únicas (***Fisiológicos*:** Geometría de la mano, iris, retina, reconocimiento facial, huella digital ***Comportamiento o Dinámica*:** Firma, voz, dinámica de teclado [HOMINI, 2004]). Los rasgos fisiológicos, estables por naturaleza, incluyen huellas dactilares, silueta de la mano, patrón del iris y firma autógrafa. Las características del comportamiento son más una reflexión de la formación del individuo. Dentro de esta clasificación caen características tales como la voz y la firma caligráfica. Un sistema biométrico es aquel reconocimiento de patrones que lleva a cabo comparaciones de identidad a la vez que valida las características almacenadas de un individuo contra las que presenta en un ambiente en vivo [MERKATUM, 2007], es decir tenemos registrada una muestra de alguna de estas características, la cual es comparada contra otra muestra que sea tomada en el tiempo en que se intente acceder. La Figura 1.2 muestra un comparativo de los sistemas biométricos por características anatómicas y de comportamiento.

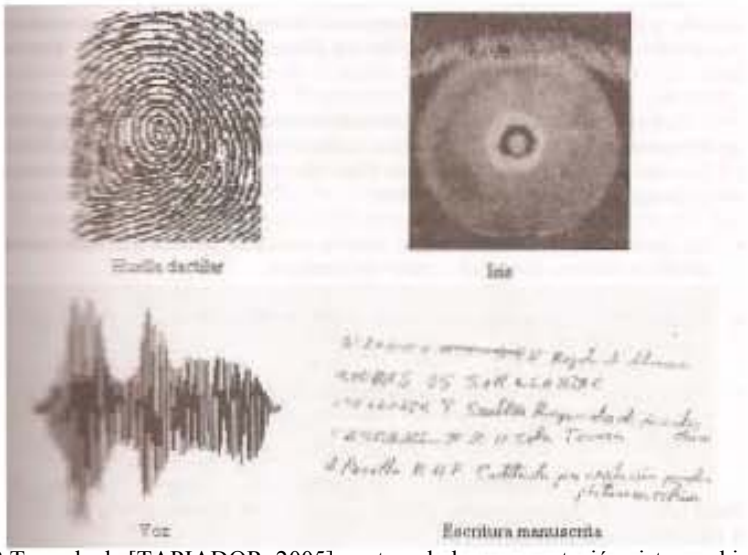

Figura 1.2 Tomada de [TAPIADOR, 2005] contempla la representación sistemas biométricos.

Además de ello existe la *multi-biometría* la cual utiliza más de un una característica fisiológica para poder determinar si el individuo es el indicado para tener acceso. Básicamente existen tres de estos:

- *Multi-modal*: en el cual es capaz de hacer uso de más de una característica para identificar la identidad, como huellas dactilares y la cara, huellas e iris, cara e iris, y uno de los más completos son cara, iris y huellas, etc.
- *Multi-algoritmo*: éste permite unir más de un método para definir la identidad a través de una sola característica fisiológica (dos métodos distintos para identificar la huella digital).
- *Multi-instancia*: de esta forma seremos capaces de identificar más de un mismo rasgo, como verificar la huella de cuatro dedos o el iris de ambos ojos.

Por ello podemos decir que esta tecnología es la más segura ya que a través de la identificación de alguna parte única en las personas podemos cerciorarnos de quien se trata.

1.3.2.2. Diferenciación por características anatómicas

Podemos clasificar a los sistemas biométricos de acuerdo a la característica física o de comportamiento, mediante algún hardware que nos permita (como ya se mencionó con anterioridad) registrar la nueva muestra que es comparada contra otra almacenada anteriormente.

Para este trabajo, nos interesan los sistemas biométricos que utilizan alguna parte del cuerpo para hacer la comparación (característica física), de esta manera obtenemos una clasificación como se muestra a continuación:

- Huellas Digitales
- Geometría de la mano
- Termografía
- Análisis del iris
- Análisis de retina
- Venas del dorso de la mano
- Reconocimiento Facial

Ahora se realiza una pequeña descripción de algunos métodos biométricos y se mencionan dentro de estos algunos de los aparatos (hardware) que nos permiten realizar los diferentes análisis para proporcionar un acceso.

Huella dactilar

Esta técnica es conocida como la dactiloscopia la cual corresponde a uno de los métodos detección biométrica por excelencia [TAPIADOR, 2005]. Esto es debido a que se ha convertido en un método bastante aceptado, lo cual lo ha hecho muy popular además de que resulta fácil su adquisición. Este método cumple ampliamente con las reglas básicas que debe tener un sistema biométrico que en este caso es que es una característica que no varía con el tiempo y que existe una amplia variedad de huellas que permite que cada ser humano tenga sus propias huellas.

Nuestras huellas digitales ya habían sido descubiertas por las antiguas civilizaciones, pero no fue sino hasta 1665 que el anatómico Marcelo Malpighi (1628-1964) realizara un estudio detallado de las *crestas papilares* y sentara las bases para este método. Cabe

mencionar que este método no fue tomado como tal hasta 1823 pues el fisiólogo Johannes Purkinje revelara que las huellas aparecen al sexto mes de gestación y que estas se mantienen inalterables a través del tiempo. Finalmente quien le dio nombre a esta técnica fue Sir Francis Galton, quien realizo diferentes trabajos donde muestra que la inexistencia de coincidencias entre las huellas dactilares [TAPIADOR, 2005] a través de métodos estadísticos, es decir que mostró que no pueden existir dos huellas iguales. Posteriormente se realizo un método mediante el cual se puede clasificar las huellas digitales, este creado por un inspector jefe de la policía de Bengala, el cual hace uso de las formas de las crestas para ordenarlas de una forma lógica. Todos los demás avances dentro de esta rama siguen todos estos estudios de identificación de huellas.

Entre los dispositivos que actualmente destacan haciendo uso del método de verificación por la huella dactilar tenemos:

Sensores de campo eléctrico**:** El sensor de campo eléctrico funciona con una antena que mide el campo eléctrico formado entre dos capas conductoras. Esta tecnología origina un campo entre el dedo y el semiconductor adyacente que simula la forma de los surcos y crestas de la superficie epidérmica.

Sensores capacitivos**:** El método capacitivo, genera una imagen de las crestas y valles de la huella en la superficie de un circuito integrado de silicona.

Dentro de la Figura 1.3 se puede apreciar distintos dispositivos para la lectura e identificación de la huella dactilar.

Figura 1.3 Diferentes dispositivos de reconocimiento e identificación de huellas dactilares.

Sensores termales**:** El sensor termal utiliza un sistema único para reproducir el dedo completo "arrastrándolo" a través del sensor. Durante este movimiento se realiza una serie de toma sucesiva y se pone en marcha un software especial que reconstruye la imagen del dedo.

Sensores ópticos**:** Los sensores ópticos se basan en una extracción de puntos de la imagen que se genera de la huella. Es el método de identificación biométrica más común, por su coste y su facilidad de uso.

Iris

Este método se ha empezado a volver popular por su veracidad y fiabilidad. El precursor de esta técnica es el profesor John G. Daugman de la Universidad de Cambridge. Más adelante se muestra un pequeño resumen de lo que es el iris, además se menciona las ventajas de su uso.

La idea básica de identificar a las persona mediante la comparación del patrón del iris fue del oftalmólogo Frank Burch en 1936. Esto hasta el momento parecía cosa de ciencia ficción hasta 1987 [TAPIADOR, 2005] cuando Leonard Flom y Aran Safir patentaron este concepto. Estos dos oftalmólogos carecían de lo necesario para poder desarrollar el sistema, lo cual los orilló a contactar a Daugman (cuando éste se encontraba en la Universidad de Harvard). Los algoritmos de Daugman fueron patentados en 1994 por el mismo. Estos algoritmos son en los que se basan en su gran mayoría los sistemas de reconocimiento de iris. Al fundar entre Flom, Safir y Daugman la empresa IriScan Corp. Empezaron a vender la licencia para que otras empresas pudieran desarrollar sistemas y que exportaran productos para el reconocimiento del iris. Un ejemplo de estas compañías en Sensar que ha creado cámaras que facilitan la extracción de imágenes del iris, la cual ha sido diseñada para ser usada en cajeros automáticos [TAPIADOR, 2005]. Básicamente lo sistemas de identificación de iris siguen los siguientes pasos:

- Localización del iris dentro de la imagen.
- Detección de los bordes[1]. Extracción del límite interno y externo del iris.
- Eliminar las partes de la imagen no deseadas.
- Se realiza una compensación en el tamaño del iris ya que la distancia de la persona a la cámara puede variar y también se debe considerar la dilatación o contracción de la pupila por la fuente de iluminación.
- Utiliza algún método conocido para la identificación del iris.

Para realizar la captura de una imagen del iris necesitamos tener en cuenta lo siguiente [TAPIADOR, 2005]:

- Se debe de usar una cámara que de la suficiente resolución.
- Se debe de aprovechar toda la resolución de la cámara para captar solamente la imagen del ojo de la persona en cuestión.
- El sujeto debe sentirse cómodo con la cámara, esto lo logramos al colocarla a una distancia aceptable donde podamos obtener una buena imagen.
- La distancia de la cámara al sujeto no debe de deformar la imagen.

Entre los métodos reconocidos para la extracción de características tenemos a la extracción mediante filtros de Gabor, de igual manera tenemos a la extracción por transformada *Wavelet* (Ondeleta), extracción por circunferencia y por corona circular.

Para realizar una autenticación del iris existen varios métodos como los ya mencionados que nos permiten realizar dicha verificación. Uno de ellos y el más comercial junto con su componente para la extracción es el siguiente:

[1] Los bordes son la frontera con la esclerótica y la pupila. El orden de la extracción puede variar.

Este tipo de reconocimiento biométrico consiste en identificar a los usuarios mediante una cámara que captura los distintos detalles que contiene el iris de usuario: Criptas, Surcos radiales, Borde pigmentado, Zona ciliar y Collarín.

El proceso de autenticación:

Se captura una imagen digital del iris.
Se prepara y procesa la imagen para el análisis.
Se crea una plantilla de 512 bytes, de la textura y patrones del iris.
Se usa la plantilla IrisCode® para la autentificación.
El código que genera la imagen de iris tiene un tamaño aproximado de 512 bytes [MERKATUM, 2007].

Toda la información transmitida entre la cámara y cualquier componente del sistema está siempre encriptado (3DES y Blowfish).El sistema permite hacer identificación y verificación. Un ejemplo de estos dispositivos encargados de realizar el análisis se muestra en la Figura 1.4.

Más adelante del trabajo se presenta un método diferente al mencionado arriba, que es el que desarrollaremos para trabajar con las imágenes del iris y obtener resultados positivos sin necesidad de usar métodos patentados (IrisCode®).

Figura 1.4 Ejemplo de dispositivo para análisis del iris.

Rostro

El reconocimiento facial es un sistema biométrico que realiza, mediante la captura de una cámara, un patrón de la cara del usuario. Normalmente las capturas son imágenes en 2 dimensiones. Aunque también hay una solución que permite el reconocimiento en 3 dimensiones. En el sistema de reconocimiento facial 3D la imagen generada tiene un tamaño que oscila de 0,55 mm a 1 mm [MERKATUM, 2007]. Se permite reproducir gráficos y combinar imágenes 2D con 3D. En la Figura 1.5 se puede apreciar una parte del proceso de identificación del rostro.

El proceso de autenticación:

Se genera un grafico 3D de la cara del usuario.
El vector binario del grafico se almacena en la base de datos.
El algoritmo permite un proceso rápido de identificación.
En entorno de red trabaja de 5 a 15 Frames por segundo.
La imagen se captura en 8 milisegundos.
El usuario puede colocarse hasta 2,5 m de la cámara [MERKATUM, 2007].

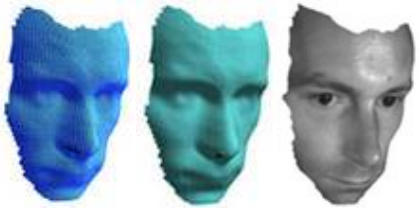

Figura 1.5 Tomada de [MERKATUM, 2007] permite ver una parte del proceso de reconocimiento de rostro.

1.3.2.3. Agrupamiento a través del criterio de comportamiento

Así como los sistemas biométricos que utilizan algún rasgo físico para realizar la comparación, existen otro tipo de métodos los cuales analizan nuestro comportamiento. Entre estos tenemos:

- Patrón de Voz
- Firma manuscrita
- Dinámica de tecleo
- Cadencia del paso
- Análisis gestual

Patrón de Voz

Estos sistemas tienen por objetivo hacer la discriminación de locutores al realizar un análisis a la señal de la voz. Algunas de las disciplinas que permiten dicho análisis son la acústica de cavidades, la anatomía, la física de fluidos o propagación de ondas.

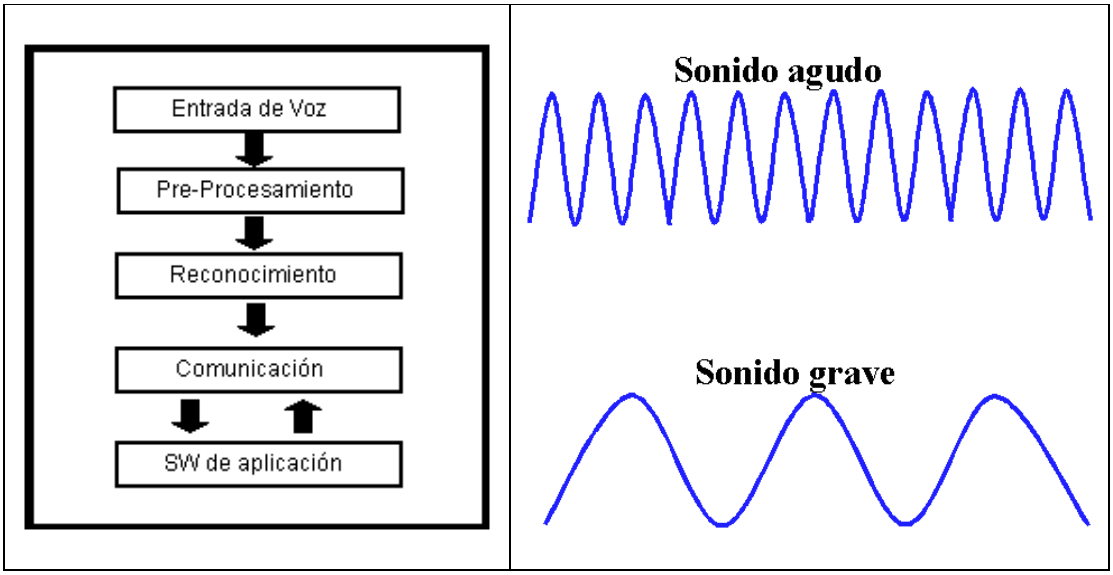

Figura 1.6 Secuencia para el procesamiento de la voz (izquierda), también el tipo de onda que produce un sonido grave y uno agudo (derecha) tomada de [COMPLUTENSE, 2007].

La Figura 1.6 nos indica los tipos de onda producidos por los diferentes sonidos, ya sean grabes o agudos. Además se aprecia la secuencia en la que es procesado el patrón de voz hasta su aplicación.

Firma manuscrita

Como la escritura es un el medio de comunicación común entre los humanos y la firma se ha vuelto una forma de identificación muy natural dentro de la sociedad, se ha vuelto un método muy popular de identificación por comportamiento. Este método de identificación resulta práctico debido a su facilidad de ejecución y su rapidez con la que se puede trazar una firma. Además de que se puede realizar en cualquier lugar y momento y no es necesario ningún aditamento especial [TAPIADOR, 2005] y tan sólo vasta una rápida verificación visual para realizar la comprobación. Actualmente se puede verificar una firma al realizar una captura digital de la misma.

Dinámica de la mecanografía

Ya que el teclado es de las principales formas en que el usuario puede comunicarse con un ordenador es por ello que se ha propuesto que sea usado para identificarlo. Generalmente se emplea dentro de sistemas de comercio electrónico o en sistemas de seguridad informática. Como bien sabemos este no es el único modo de interacción entre el ordenador y el usuario, pero si es por el que se introduce un mayor número de información, casi un 90% de la totalidad de la información se transmite mediante el teclado [TAPIADOR, 2005]. Esta idea nace con el uso de los telégrafos, ya que los operadores eran capaces de identificarse entre si sin importar las estaciones en que se encontraran, esto debido al ritmo en que realizabán las pulsaciones del código Morse.

En la Tabla 1.2 extraída de [WIKIMEDIA, 2007] se muestra un análisis que permite comparar la efectividad de diferentes sistemas biométricos basándose en aspectos como la fiabilidad, estabilidad, su aceptación, así como la facilidad para extraer los datos a procesar.

Tabla 1.2 Modificada de [WIKIMEDIA, 2007] señala un comparativo de diferentes sistemas biométricos.

	Ojo (Iris)	**Ojo (Retina)**	**Huellas dactilares**	**Geometría de la mano**	**Escritura y firma**	**Voz**	**Cara**
Fiabilidad	Muy alta	Muy alta	Alta	Alta	Media	Alta	Alta
Facilidad de uso	Media	Baja	Alta	Alta	Alta	Alta	Alta
Prevención de ataques	Muy alta	Muy alta	Alta	Alta	Media	Media	Media
Aceptación	Media	Media	Media	Alta	Muy alta	Alta	Muy alta
Estabilidad	Alta	Alta	Alta	Media	Baja	Media	Media

1.4. El iris y ventajas de su uso

El humano cuenta con dos ojos que ayudan a la visión estereoscópica que nos permite ver 180°. Estos se forman alrededor del día número 25 en la fase embrionaria y se sigue así hasta que a la octava semana ha terminado la génesis del esbozo ocular [TAPIADOR, 2005], posteriormente seguirá madurando hasta el noveno mes.

El ojo puede dividirse en tres capas, la externa, la media y la interna. De una manera sencilla la capa externa contiene lo que es la córnea y la esclerótica. La siguiente capa que es la que más nos interesa es también denominada úvea, la cual a su vez se puede separar en dos partes, la úvea anterior y la posterior. La úvea anterior está compuesta por el cuerpo ciliar y el iris, la úvea posterior es más que nada un campo vascular que se encuentra entre la retina y la esclerótica. Se muestra en la Figura 1.7 los componentes del ojo.

El iris es un órgano interno del ojo, como ya se mencionó, perteneciente a la capa media o úvea. Básicamente el es un aro de fibras musculares de color que envuelve a la pupila. Se encuentra detrás de la cornea y el humor acuoso. Contiene los músculos esfínter y dilatador del iris que actúan como un diafragma ocular. Los músculos de éste son los que se encarga de darle tamaño a la pupila, lo que permite que entre determinada cantidad de luz a nuestro ojo, se contrae si es la luz es intensa y se dilata si es débil. De este modo se regula la cantidad de luz que entra al ojo según sea necesario.

Y es al iris que le debemos el color a nuestros ojos. Lo que le da el color a nuestro iris son dos capas de células, una delante y otra detrás que contienen materia colorante, denominada pigmentos [SHAPIRO, 1980].

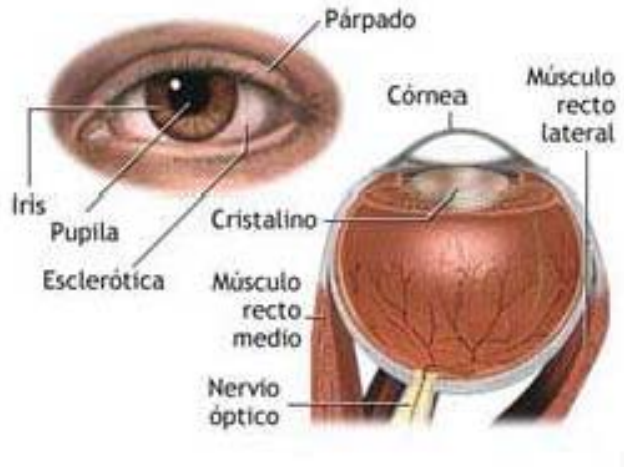

Figura 1.7 Esquema de los componentes del ojo humano.

El iris tiene ventajas ante otras partes del cuerpo ya que este no se modifica durante el transcurso de la vida (ya que es un órgano interno y por ello protegido naturalmente), pues está protegido por la cornea que es transparente y la pupila, lo que lo hace inmune a la influencia del medio ambiente [MAGAÑA, 2005]. Además de que se puede obtener una imagen de este a corta distancia sin necesidad de tener un contacto que resulte agresivo.

De esta manera podemos decir que el iris es un órgano bien posibilitado para ser usado como medio de verificación biométrico por las siguientes razones:

- Gracias a la protección que ofrece la cornea el ojo queda protegido de cambios originados por accidentes.
- El iris esta sujeto a variaciones en su apertura debido a los cambios de iluminación que posteriormente resulta en un sencillo mecanismo de identificación de un sujeto vivo.
- Como ya se dijo, para realizar la captura de una imagen no es necesarios métodos agresivos ya que es un órgano expuesto y por tanto visible desde el exterior.
- Como dato adicional, el intento de suplantar a otro individuo al falsificar el iris conlleva a operaciones que pueden resultar riesgosas pues puede dañar la visión.

Capitulo 2

Morfología matemática y lógica difusa como herramienta de análisis

Este capítulo muestra una descripción de la morfología matemática, sus características y operaciones básicas, mismas que al combinarse con la lógica difusa obtenemos la morfología matemática difusa que es empleada cuando la morfología matemática tradicional encuentra limitantes para aplicar sus operaciones conservando a la morfología clásica como un caso particular de la difusa.

2.1. Morfología matemática

Se empieza a retomar este trabajo a partir de definiciones básicas que nos son de utilidad para tener un mejor concepto de lo que es la morfología y para este caso la morfología borrosa, así como recordar como es que los sistemas de seguridad biométricos se han convertido en los más eficientes.

2.1.1. Características de la morfología matemática

Así como en la biología existe un estudio de la morfología de los seres vivos, existe en matemáticas una herramienta llamada morfología matemática, ésta nos permite hacer un estudio de las imágenes digitales ya sea en blanco y negro, en escala de grises y aunque un poco más complejo imágenes a color.

La palabra morfología significa forma y estructura de un objeto. Para imágenes binarias se definen operaciones morfológicas y con estas se constituye una herramienta de extracción de componentes de imagen útiles en la representación y descripción de la forma de las regiones [FERREIRA, 2004].

El utilizar esta herramienta da oportunidad para realizar reconocimiento de caracteres, análisis en imágenes médicas entre otras aplicaciones. Aun cuando tiene muchos usos, también tiene restricciones para su práctica y es ahí donde es oportuna la morfología matemática difusa o borrosa como una extensión de la morfología matemática clásica.

2.1.1.1. Operaciones básicas de conjuntos

Como la morfología matemática utiliza lo que es la teoría de conjuntos, es necesario recordar algunas operaciones básicas [GONZALEZ, 2003]:

1. La traslación de A por $x \in E^n$ que se nota A_x se define como

$$A_x = \{c \mid c = a + x, \text{para algún } a \in A\}$$

2. La reflexión de B notada $\hat{B}$ se define como

$$\hat{B} = \{x \mid x = -b, \text{para algún } b \in \mathrm{B}\}$$

3. Por último la diferencia de dos conjuntos *A* y *B*, notada *A* - *B*, se define mediante

$$A - B = \{x \mid x \in A, x \notin B\}$$

2.1.1.2. Operaciones básicas de la morfología matemática

Las operaciones básicas de la morfología matemática son: **dilatación** se puede simplificar a decir que es agregar píxeles a un objeto, hacerlo mas grande, y luego **erosión** es hacerlo mas chico [ALBA, 2006]. La erosión saca los "*outlayers* del objeto". En pocas palabras el objetivo principal de estas dos operaciones es hacer más grande o más pequeño a una imagen al agregar o remover píxeles de esta. Los símbolos que representan estas operaciones son: $\ominus$ para la erosión y $\oplus$ para la dilatación.

Como un ejemplo de esto se presenta en las figuras [DWAYNE, 2000]:

La Figura 2.1 es una tabla que nos representa un arreglo de píxeles de una imagen cualquiera.

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	200	200	200	200	200	200	0	0
0	0	200	200	200	200	200	200	0	0
0	0	200	200	200	200	200	200	0	0
0	0	200	200	200	200	200	200	0	0
0	0	200	200	200	200	200	200	0	0
0	0	200	200	200	200	200	200	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Figura 2.1 Extraída de [DWAYNE, 2000] representa una imagen cualquiera.

La Figura 2.2 donde en cada casilla con el número 200 que se encuentre con dos o más ceros a su alrededor (arriba o abajo) este es reemplazado por un cero.

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	200	200	200	200	0	0	0
0	0	200	200	200	200	200	200	0	0
0	0	200	200	200	200	200	200	0	0
0	0	200	200	200	200	200	200	0	0
0	0	200	200	200	200	200	200	0	0
0	0	0	200	200	200	200	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Figura 2.2 Extraída de [DWAYNE, 2000] muestra a la Figura 2.1 Erosionada.

En la figura 2.3 donde en cada casilla que contenga un cero que se encuentre junto a otra que con el número 200 se cambiará por los tres asteriscos.

0	0	0	0	0	0	0	0	0	0
0	***	***	***	***	***	***	***	***	0
0	***	200	200	200	200	200	200	***	0
0	***	200	200	200	200	200	200	***	0
0	***	200	200	200	200	200	200	***	0
0	***	200	200	200	200	200	200	***	0
0	***	200	200	200	200	200	200	***	0
0	***	200	200	200	200	200	200	***	0
0	***	***	***	***	***	***	***	***	0
0	0	0	0	0	0	0	0	0	0

Figura 2.3 Del libro [DWAYNE, 2000] es la Figura 2.1 Dilatada.

Luego la combinación de estas da origen a los operadores **Apertura** y **Clausura**. El primero consiste en aplicar una erosión seguida de una dilatación aplicando la misma forma estructurante, como resultado esta tiende a "abrir pequeños huecos". La clausura es el la aplicación de las operaciones básicas en el sentido inverso, y resulta en "cerrar los huecos".

La apertura de A por un elemento estructural K se nota $A \circ K$, se define como

$$A \circ K = (A \ominus K) \oplus K$$

que en palabras establece que la apertura de A por K es simplemente la erosión de A por K, seguido de la dilatación del resultado por K. [GONZALEZ, 1993]

La clausura de A por un elemento estructural K se nota $A \bullet K$, se define como

$$A \bullet K = (A \oplus K) \ominus K$$

que en palabras establece que la clausura de A por K es simplemente la dilatación de A por K, seguido de la erosión del resultado por K.

Si A no cambia con la apertura con K diremos que A es abierto con respecto a K. Si A no cambia con la clausura con K diremos que A es cerrado con respecto a K [GONZALEZ, 1993].

2.2. Lógica difusa

Para empezar a hablar de lógica difusa primero debemos recordar que la lógica clásica le asigna dos valores al evento para determinar que es verdadero o falso. Es por esta razón que a esta lógica se le conoce como lógica bivalente o binaria [CORZO, 2007].

Pero además existen lógicas que admiten un tercer valor (lógica trivaluada) u otros tantos valores (lógica multivaluada).

La lógica aristotélica nos permite resolver algunos problemas y fenómenos, pero la gran mayoría de ellos son enmarcados dentro de un concepto matemático. Mientras que la

lógica difusa (o también conocida como borrosa) puede usarse para explicar el mundo en el que vivimos ya que sigue el comportamiento humano de razonar, al sacar conclusiones de hechos observados [CORZO, 2007].

Como se ha mencionado la lógica multivaluada permite aceptar diferentes valores para decidir si algo es verdadero o falso. Una de estas es la lógica difusa que se caracteriza por tratar de cuantificar ese valor de verdad dentro de un rango. Por ejemplo si se tiene una proposición *P* se le puede asociar un número *v(P)* en el intervalo [0,1] tal que:

- Si *v(P)* = 0, *P* es falso.
- Si *v(P)* = 1, *P* es verdadero.
- La veracidad de *P* aumenta con *v(P)* [CORZO, 2007].

Esto le da un cierto toque de parecido a la teoría de probabilidad. Pero se le da un enfoque distinto [CORZO, 2007].

Un ejemplo de esto es cuando queremos decir que el agua se encuentra fría o caliente. Ya que se le puede dar cierto grado pues para algunos el agua congelada y el agua hirviendo serían nuestros topes. Pero dentro de estos podemos encontrar que algunos dirán que el agua esta templada si apenas lleva un determinado tiempo de calentarse, a otros les parecerá suficientemente para decir que está caliente. Para algunos el agua a una temperatura ambiente les parecerá agradable, mientras que a otros les puede parecer fría.

Para este ejemplo podremos decir que el valor mínimo es 0 para el agua congelada, que el 0.25 es para el agua en la mañana, un 0.50 para el agua tibia, un 0.75 para el agua caliente y el 1 para el agua hirviendo que sería nuestro valor máximo.

Podemos darle cierto grado de verdad a la mayoría de los eventos, como si una persona es alta, dar un pronóstico del clima, entre otros.

El nacimiento de la lógica difusa (fuzzy en ingles) se remonta a Aristóteles el cual propone la existencia de los grados de verdad o falsedad. Mientras tanto Platón trabaja sobre los grados de pertenencia.

Posteriormente encontramos que Kant y David Hume adoptan este tipo de ideología pues dicen que el razonamiento se va generando por sucesos de los que somos testigos durante toda nuestra vida. Esto es a lo que se le denomina lógica del sentido común.

Entre otros tenemos a Charles Sanders Peirce quien es fundador de la corriente del pragmatismo, la cual es la primera en introducir las vaguedades de la lógica. Esto permite un acercamiento al pensamiento humano. También tenemos al personaje británico Bertrand Russell con la paradoja del conjunto, Ludwig Wittgenstein quien estudiase las acepciones de una palabra, Jan Lukasiewicz con la lógica trivaluada.

Y quien finalmente le dio forma a todo esto es Lotfi A. Zadeh, quien publica un artículo titulado "Fuzzy Sets" (*Conjuntos Difusos*) 1975. Esta lógica permite darle una representación matemática a conceptos o conjuntos imprecisos [CORZO, 2007], algunos de estos son días fríos, meses calurosos, personas altas, entre otras. En este

artículo introduce el concepto de la lógica infinito valorada, donde da a conocer el concepto de conjunto difuso y así mismo el de lógica difusa [MAGAÑA, 2005].

2.2.2. Morfología matemática difusa

Como se mencionó anteriormente, la Morfología Matemática Difusa nace cuando la morfología convencional se encuentra con limitantes para poder trabajar dentro de las distintas aplicaciones.

Estas complicaciones se dan cuando no tenemos la certeza de los datos o métodos que vamos a procesar. Es por ello que llegamos a técnicas basadas en la lógica difusa. Ya dentro de lo que es el procesamiento de imágenes podemos encontrarnos con la incertidumbre al realizar la manipulación de píxeles, de reconocimiento geométrico y de lo que llamamos extracción de conocimiento de imágenes [SERRA, 1982]. Y en el Diagrama 2.1 podemos apreciar los diferentes niveles en los que se puede presentar la incerteza al realizar el procesamiento de imágenes.

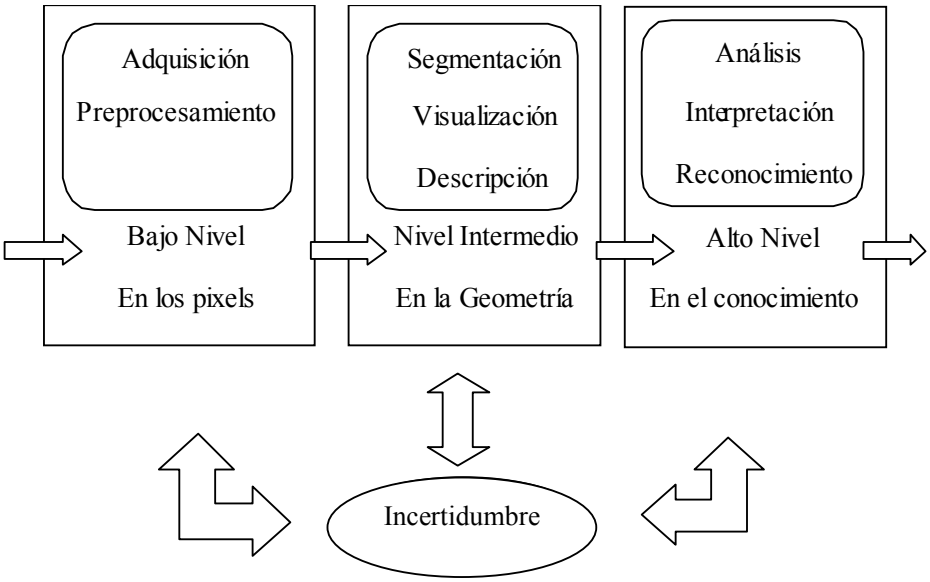

Diagrama 2.1 Diferentes niveles a los que se aplica la morfología matemática difusa.

- **Incertidumbre en la manipulación de los píxeles**. Existe incertidumbre en el procesamiento de bajo nivel, relacionado con la manipulación de la intensidad de píxeles. Operaciones utilizadas en este nivel son: Mejora de Contraste, Suavizamiento, Binarización, etc.

- **Incertidumbre en la interpretación geométrica**. Existe incertidumbre en el procesamiento de nivel intermedio, que tiene que ver con las relaciones geométricas de los diferentes componentes de las imágenes. Operaciones utilizadas en este nivel son: Detección de Bordes, Seguimiento de Contornos, etc.

- **Incertidumbre en la extracción de conocimiento**. Existe incertidumbre en el procesamiento de alto nivel de las imágenes (análisis). En este nivel se realizan las tareas de Análisis de Escenas y de Reconocimiento de Objetos, tareas en las cuales frecuentemente existe incertidumbre.

Una de las primeras etapas dentro de cada sistema difuso es la de *fuzzificación*. Para el caso de procesamiento de imágenes existen tres niveles para llevarla a cabo, una de ellas es la de fuzzificación por histograma donde se realiza una normalización entre cero y uno de los valores del histograma los cuales son una función de membresia. También tenemos la denominada fizzificación local o de vecindades donde cada píxel tiene una cierta función de membrecía de acuerdo al valor de dicho píxel. Por último tenemos la fuzzificación de características, en la cual se usan reglas del tipo if-then.

2.2.2.1. Operaciones básicas de la morfología difusa

Las ecuaciones de dilatación y de erosión difusas utilizadas en el trabajo son las propuestas por Bloch y Maître [MAGAÑA, 2005]:

Dilatación difusa

$$(g \oplus \mu)(x) = \sup_{y \in X} \min[g(x-y), \mu(y)]$$

Erosión difusa

$$(g \Theta \mu)(x) = \inf_{y \in X} \max[g_{\max} - g(x-y), \mu(y)]$$

Donde: $g(x)$ es la imagen difusa, $\mu(x)$ es el elemento estructural difuso, X dominio (Rn) ó elemento del discurso y x, y son vectores de posición del píxel [MAGAÑA, 2005].

De igual forma las operaciones de apertura y cerradura en su forma difusa son extendidas de la combinación de las operaciones básicas de la morfología difusa aplicando el principio de extensión (cita donde se menciona este principio).

Capítulo 3

Procesamiento morfológico y segmentación de la imagen del iris

Dentro de éste capítulo se explica la metodología que es empleada para la extracción del iris de una imagen digitalizada del ojo, con el fin de poder someterla a diferentes procesos hasta la obtención como producto de ello de un código de barras que sirve como identificador único. De igual forma se hace mención de la herramienta empleada para poder manipular y extraer las características de la imagen del ojo, sus ventajas sobre otras herramientas y finalmente la forma de instalación en el ambiente de trabajo.

3.1. Procesamiento morfológico

El método para la obtención del código de barras resulta de procesos secuenciales que no pueden ser ejecutados sin que se haya terminado el proceso que le antecede. Esto es debido a que es necesario procesar la imagen de esta manera para trabajar sólo con los elementos que representen información relevante. En el Diagrama 3.1 se aprecia el flujo que se encarga de procesar la imagen del iris hasta la obtención del código de barras. La base para la obtención del iris es determinar el centro del iris, estimar el radio interior, recortar la imagen y estimar el radio interior [ALMEIDA, 2004].

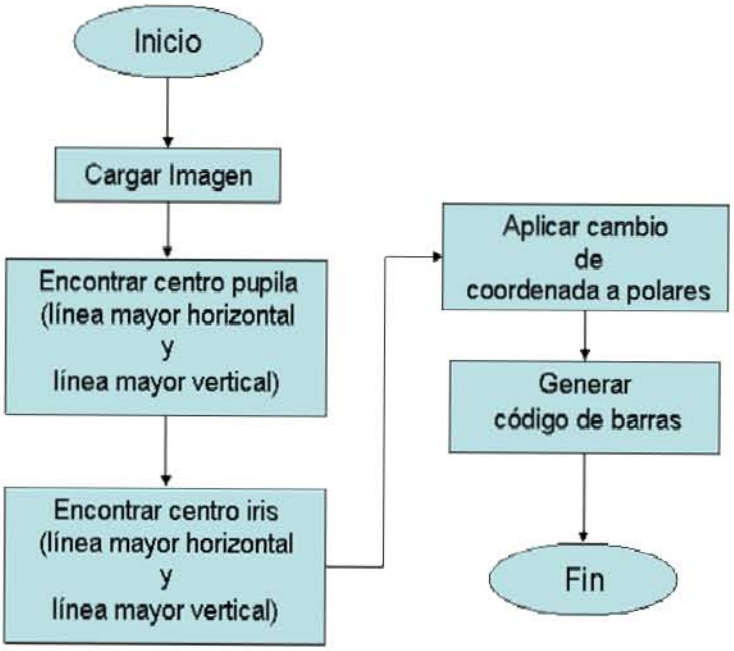

Diagrama 3.1 Muestra el flujo para el proceso de identificación del iris.

3.1.1. Carga de la imagen

La imagen obtenida del prospecto es cargada mediante una función `IMG_Load("iris.jpg")` de la librería que se utiliza para este proyecto (librería SDL, ver sección 3.2 del presente capítulo), la cual se ha seleccionado por sus bondades. El formato de imagen es jpg, el archivo se debe encontrar en el directorio "*Img*" de la

aplicación y debe ser nombrado con el identificador “iris.jpg” para ser reconocido por la aplicación.

3.1.2. Límite interno

Para encontrar el límite interno (correspondiente al área de la pupila) de la imagen se calcula antes que nada la ubicación del centro. Esto lo logramos al ubicar tanto la línea con mayor número de píxeles en vertical como la línea con mayor número de elementos en horizontal.

El área con la que se trabaja debe de pasar por un par de procesos previos, uno de ellos llamado umbralización, que en otras palabras limpia la imagen dejando las áreas más oscuras, esto abarca el área de la pupila (en su mayoría) y algunos puntos dentro de toda la imagen (estas en menor sentido). El siguiente código muestra el proceso de umbralización. El código fuente de toda la aplicación se encuentra disponible en el Anexo 1.

```
.
.
.
for(i = 0; i < surface->h; i++){
	for(j = 0; j < surface->w; j++){
		// Asumimos que es una imagen en escala de grices, podemos usar
	cualquier color, en este caso rojo
		otroPixel = getpixel(surface, j, i);
		tempR = otroPixel&surface->format->Rmask;
		tempR = tempR>>surface->format->Rshift;
		tempR = tempR<<surface->format->Rloss;
		red = (Uint8)tempR;

		if(red > t){
			otroPixel = SDL_MapRGB(surface->format, 255, 255, 255);
			putpixel(surface, j, i, otroPixel);
		} else {
			otroPixel = SDL_MapRGB(surface->format, 0, 0, 0);
			putpixel(surface, j, i, otroPixel);
		}
	}
}
.
.
.
```

Donde `suface` es el área total de nuestra imagen que es recorrida en su totalidad y evalúa cada uno de sus píxeles transformando en blancos a todos aquellos que tienen un valor mayor a 60 y en negro a todos aquellos que no lo sean.

El valor de los píxeles se toma en base a un solo color, ya que al tratarse de una imagen en escala de grises, el combinar los tres colores básicos en un mismo nivel proporciona una tonalidad de gris. Para este caso se hace uso de la identificación de la tonalidad del color rojo. Este valor lo localizamos dentro del código expuesto en la variable `red`.

El segundo proceso se encarga de dejar en una imagen resultante simplemente un área circular correspondiente al área de la pupila, todo lo demás son desechado o como se ha mencionado al principio del trabajo se remueve la basura y se quitan todas las manchas en el área (ver sección 4.1 del Capítulo 4) aplicando las operaciones de dilatación y erosión.

```
.
.
.
/*
* vecWht y vecBlk son usados para saber cuantos píxeles vecinos tienen color
blanco y negro,
* en base a eso decidir que color tomará nuestro píxel
*/
if(vecWht != 12 && vecBlk != 12){
	if(vecWht > vecBlk && red != 255){
		otroPixel = SDL_MapRGB(surface->format, 255, 255, 255);
//Erosión
		putpixel(surface, j, i, otroPixel);
	} else if(vecBlk > vecWht && red != 0){
		// Dilatación con mascara de 3 x 3
		otroPixel = SDL_MapRGB(surface->format, 0, 0, 0);
		putpixel(surface, j - 1, i - 1, otroPixel);
		putpixel(surface, j, i - 1, otroPixel);
		putpixel(surface, j + 1, i - 1, otroPixel);
		putpixel(surface, j - 1, i, otroPixel);
		putpixel(surface, j, i, otroPixel);
		putpixel(surface, j + 1, i, otroPixel);
		putpixel(surface, j - 1, i + 1, otroPixel);
		putpixel(surface, j, i + 1, otroPixel);
		putpixel(surface, j + 1, i + 1, otroPixel);
	}
}
.
.
.
```

El fragmento de código citado anteriormente muestra la aplicación de las operaciones de para obtener el área en la forma deseada. En él se usan dos variables calculadas con anterioridad (`vecBlk, vecWht`) que nos indican el número de píxeles de color negro y de color blanco que se encuentran alrededor del píxel evaluado en la posición `i,j`. Si la variable `vecBlk` resulta ser mayor que `vecWht` se realiza una dilatación con una mascara de 3 x 3, cuyo resultado es pintar de color negro los píxeles en las posiciones que se muestra en el código. En caso contrario únicamente el píxel evaluado es coloreado en blanco.

La variable `surface` es la que contiene toda el área de la figura original, además se encarga de mantener los cambios sin necesidad de afectar a la imagen origen, las modificaciones se guardan en memoria. La variable `otroPixel` es la encargada de indicar el color que se modifica al píxel dado.

Para realizar dicho cambio se aplica la función de la librería SDL (ver sección 3.2) "*putpixel*", la cual recibe una superficie (`surface`), la posición del píxel a modificar dentro de la superficie (`i,j`) y el formato deseado para el píxel (`otroPixel`).

Con la nueva área calculada se busca la línea horizontal con más píxeles de color negro, de igual manera se obtiene la línea vertical con el mayor número de píxeles negros. Para ello es usado un contador que incrementa al cumplirse la condición de que el píxel revisado y su consecutivo son de color negro. El resultado es el centro de la pupila.

3.1.3. Limite externo

De la misma manera que al extraer el límite interno se requiere de calcular el centro del iris, el cual resulta ser el mismo que el centro de la pupila. La diferencia se encuentra en el tamaño del radio que tiene el iris el cual siempre es mayor que el de la pupila.

Obteniendo el radio del iris se remueve todo aquello que no se encuentre dentro de la nueva área. Haciendo uso al mismo tiempo de los datos extraídos al encontrar el área de la pupila, se extrae esta del área que le corresponde al iris. De esta forma se obtiene una imagen resultante en forma de dona.

Se hace uso del mismo método de coincidencias que en el límite interno, la diferencia más importante es que el área nueva para esta evaluación es aproximadamente el doble de la que tiene el área de la pupila. Además se hace el uso de las coincidencias para loca la amplitud del área que queremos trabajar.

```
.
.
.
for (i = 0; i < 3; i++){
   for(j = 0; j < 3; j++){
               if(i == 1 && j == 1){
                       ES[i][j] = 8;
               } else {
                       ES[i][j] = 1;
               }
               tempV = vecino[i][j]&surface->format->Rmask;
               tempV = tempV>>surface->format->Rshift;
               tempV = tempV<<surface->format->Rloss;
               vColor = (Uint8)tempV;
               if(vColor == 0){
                       vColor = 1;
               }
               Imxy[i][j] = vColor;
               IMES[i][j] = Imxy[i][j] * ES[i][j];
   }
}

sumVec = IMES[0][0] + IMES[0][1] + IMES[0][2] + IMES[1][0] +
         IMES[1][2] + IMES[2][0] + IMES[2][1] + IMES[2][2];
.
.
.
```

Cada resultado de `sumVec` es comparado con la variable `IMES[1][1]` cuyo valor siempre es de ocho, si el resultado de la comparación es verdadera, se dice que tenemos una coincidencia. Todas las coincidencias son guardadas en un contador hasta que no existan más. De esta manera obtendremos un radio.

```
.
.
.
if(sumVec == IMES[1][1]){
   coincidencias++;
} else {
   break;
}
.
.
.
```

Una vez obtenido el rádio se procede a obtener el diámetro, hecho esto calculamos el límite interno y lo removemos, posteriormente coloreamos de blanco todos los píxeles que se encuentran fuera del diámetro que calculamos con anterioridad.

```
.
.
.
radioC = coincidencias * 2;
tmpRadioC = radioC;
radioC2 = radioC * radioC;
limiteInterno(surface);
BorraLimInterno(surface);
for(x = 0; x < surface->w; x++){
  for (y = 0; y < surface->h; y++){
          x2 = (x - xcentro) * (x - xcentro);
          y2 = (y - ycentro) * (y - ycentro);
          x2y2 = x2 + y2;
          if(x2y2 >= radioC2){
                    otroPixel = SDL_MapRGB(surface->format, 255, 255, 255);
                    putpixel(surface, x, y, otroPixel);
          }
  }
}
.
.
.
```

Al final obtenemos simplemente la imagen del iris que como se ha mensionado anteriormente es una imagen en forma de dona. Se hace uso del mismo método de coincidencias que en el límite interno, la diferencia más importante es que el área nueva para esta evaluación es aproximadamente el doble de la que tiene el área de la pupila.

3.1.4. Cambio de coordenadas

Como resulta más práctico trabajar con imágenes en forma rectangular en vez de figuras circulares, es necesario transformar la imagen resultante a un rectángulo, para ello se realizan transformaciones de coordenadas polares a cartesianas. Esta forma permite posteriormente aplicar ondeletas para realizar las compresiones necesarias.

```
.
.
.
for(rTotal = radioC; rTotal >= radio; rTotal--){
  xpc++;
  ypc = 0;

  for(te = 629; te >= 270;){
          teta = (M_PI * te) / 180;
          xt = xcentro + (rTotal * cos(teta));
          xtt = round(xt);
          yt = ycentro + (rTotal * sin(teta));
          ytt = round(yt);
          ypc++;
          imPolar[xpc - 1][ypc - 1] = getpixel(surface, xtt, ytt);
          te = te - 0.8;
  }
}
.
.
.
```

Guardamos en una matriz (`imPolar[ ][ ]`) el valor de los píxeles, el valor de las posiciones se obtiene en contadores dentro de los ciclos y el valor del píxel corresponde a los cambios de coordenadas que se van realizando.

Una vez obtenida la matriz con las nuevas coordenadas de la imagen, se pintan los píxeles en la posición correspondiente.

3.1.5. Compresión por ondeletas

Con la imagen obtenida al aplicarle el cambio de coordenadas, se aplica una compresión por ondeletas Haar [STOLLNITZ, 2007] un total de cuatro veces, en la cual la imagen es dividida en cuatro subimágenes. Dichas imágenes son una compresión de la imagen anterior dando como resultado cuatro imágenes de la mitad del tamaño de la original. La Figura 3.1 tomada de [MAGAÑA, 2005] muestra la forma en que se realiza la compresión mediante las ondeletas de Haar.

Para inciar la primera compresión se recorta la imagen hasta que tenga un tamaño de 448 x 48 píxeles. Una vez que la imagen tenga el tamaño adecuado se realiza la compresión de manera que la imagen tenga la mitad del tamaño actual. Durante el proceso de compresión se extraen datos representativos que son útiles para la generación del código de barras. El siguiente fragmento de código muestra la extracción de dichos datos para la primera compresión.

```
if(x1 < 448 && y1 < 48){
       d1i = (red[0] + red[1] + red[2] + red[3])/4;
       tmpd = (red[0] - red[1] - red[2] + red[3])/4;
}
```

Donde `d1i` da la compresión de los píxeles de la nueva imagen A y `tmpd` es un contenedor que se suma posteriormente a una variable cuyo objetivo es obtener un promedio de todos los valores contenidos en la matriz D.

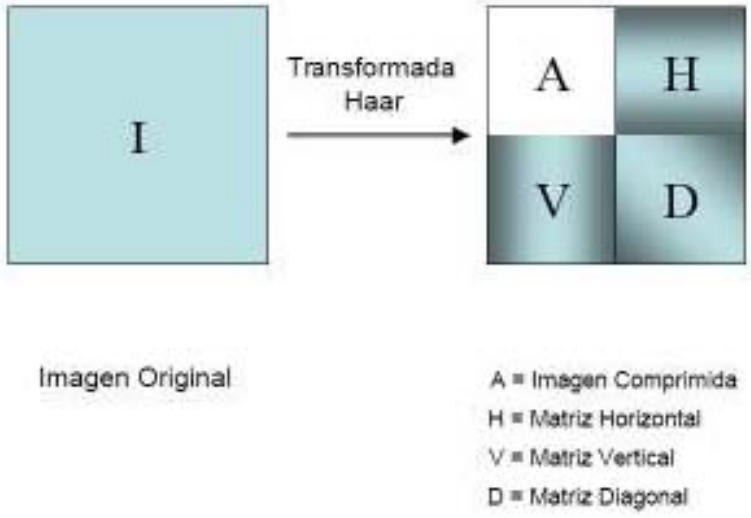

Figura 3.1 Compresión por ondeletas de Haar [MAGAÑA, 2005]

Las ecuaciones retomadas de [MAGAÑA, 2005] propuestas por Sánchez-Pérez-Nakano permiten obtener cada una de los componentes de la compresión.

$$A = (a_1 + a_2 + a_3 + a_4)/4$$
$$H = (a_1 + a_2 - a_3 - a_4)/4$$
$$V = (a_1 - a_2 + a_3 - a_4)/4$$
$$D = (a_1 + a_2 - a_3 + a_4)/4$$

De igual manera que para la primera compresión se realiza el proceso tres veces más, así se obtiene una imagen final de 28 x 3 y se guardan los resultados de las matrices diagonales anteriores D1, D2, D3 que corresponden a las compresiones anteriores.

3.1.6. Generación de código de barras

El área de la última imagen obtenida es de 84 píxeles los cuales cuentan con valores tanto positivos como negativos al hacer uso de las ecuaciones para cada píxel. Dichos valores toman una represetación binaria (los valores positivos corresponden a 1 y los negativos al 0). Estas representaciones forman una nueva imagen con puntos de su respectivo color y como se propone en [MAGAÑA, 2005], para ser visualizados son multiplicados varias veces (en esta trabajo se hace 100 veces).

Finalmente se agrega el valor obtenido de las compresiones anteriores agregando tres valores más a nuestro código binario sumando un total de 87 bits. En el capítulo 4 se describe la funcionalidad de este proceso en un ejemplo real.

El siguiente código muestra la impresión del código binario y el código de barras sin contar los tres dígitos adicionales de las compresiones anteriores.

```
.
.
.
for(xa5 = 0; xa5 < 28; xa5++){
	for(ya5 = 0; ya5 < 3; ya5++){
		if(d4tmp[xa5][ya5] >= 0){
			otroPixel[3] = SDL_MapRGB(surface->format, 255, 255, 255);
			printf("1");
		} else {
			otroPixel[3] = SDL_MapRGB(surface->format, 0, 0, 0);
			printf("0");
		}

		yc5++;
		for(i = 0; i <= 100; i++){
			putpixel(surface, yc5, i, otroPixel[3]);
		}
	}
}
.
.
.
```

La variable d4tmp es una matriz con los valores correspondientes a los obtenidos de la imagen en la última compresión. Como se puede apresiar los valores que son positivos son coloreados de blanco en la nueva imagen y se les asigna el número 1 dentro del código de barras. Por otra parte los valores negativos son coloreados en negro y asignado el número 0 dentro del código de barras.

3.2. Herramienta SDL

Para realizar la manipulación de la imagen se ha seleccionado como librería base a la llamada SDL (Simple DirectMedia Layer) versión 1.2.8, sus complementos de audio en la versión 1.2.6 y el de imágenes versión 1.2.4. Como es una librería que principalmente fue creada para el desarrollo de videojuegos, esta se encuentra dividida en ocho subsistemas que corresponde a todos los elementos que generalmente engloba un videojuego como lo es el Audio, la lectura desde el CD-ROM, manejo de eventos,

manipulación de archivos de entrada y salida, uso de Joystick, procesos iniciados mediante hilos (*Threads*), cronómetros y contadores, además de la parte principal que es el Video. Por esta variedad de subsistemas se ha creado una función principal que sirve para hacer uso de las demás funciones llamada `SDL_Init()`. Las siguientes invocaciones de la función principal son ejemplos de formas que se pueden utilizar de acuerdo a la intención o uso de la libreria SDL.

```
SDL_Init ( SDL_INIT_VIDEO );
SDL_Init ( SDL_INIT_VIDEO | SDL_INIT_TIMER );
```

Además de ello cuenta con una función que indica que todos los demás procesos terminan a la que se le conoce `SDL_Quit()`; esta acción es también conocida *shutdown.*

Los desarrolladores de esta librería indican que su herramienta es robusta, es por ello que se ha vuelto necesario el manejo de los errores que puedan aparecer. Así, dentro de ella se ha agregado un manejador para errores (invocado mediante la función `SDL_GetError()`), que nos permite detectar los errores en el momento que sucedan, mandar un aviso, que resulta muy útil al momento de probar la aplicación o realizar lo que en programación se conoce como *debug.*

Entre algunos de los puntos que la caracterizan se encuentra su configuración que resulta más sencilla dentro del entorno de trabajo IDE para el lenguaje de programación C++ en su versión de WxDevcpp 10.1[1]. Además de que existe una mejor documentación que facilita el trabajar con la librería así como el conjunto de funciones con las que cuenta. Cabe mencionar que la librería cuenta con muchos aplicativos que nos permite hacer uso no sólo del Joystick sino que también permite tanto el manejo del teclado como del ratón (mouse) y otros dispositivos E/S.

Otra de las ventajas de la librería es que cuenta con una interconexión con opengl, lo que facilita la creación de ventanas y manejo de imágenes que es la parte esencial de este trabajo. La librería SDL en sus últimas versiones incluye soporte para trabajar con diferentes formatos de imágenes, jpg, gif, bmp, entre otras (ver la sección 1.2 del Capítulo 1).

La librería está enfocada para personas que desean desarrollar particularmente un videojuego, es por ello que contiene tan gran variedad de herramientas. Está diseñada para trabajar dentro de ambientes Linux, MacOS y Win32 usando interfaces multimedia básicas de gran desempeño, audio y video principalmente [CALL, 2007].

3.2.1. Herramientas similares

Existen otro tipo de herramientas que permiten la manipulación de imágenes, pero que por la complejidad de su implementación y la falta de documentación no son utilizadas dentro de la aplicación. Se hace mención de algunas de ellas a las que se ha encontrado información suficiente.

GBM (Generalised Bitmap Module): Ésta librería entre sus componentes permite realizar la carga y guardado de imágenes en los formatos más populares. Esta

[1] Esta versión puede ser descargada de: http://wxdsgn.sourceforge.net/

característica permite que se escriban códigos donde se realice la carga de una imagen sin necesidad de conocer exactamente el tipo de imagen que se desea usar.

Cuenta con funciones básicas para trabajar con imágenes como lo son:

- Reflexión
- Transposición en x,y
- Rotación
- *Scaling* (Ampliación de Imágenes)
- Difuminado
- Entre otras

Da soporte para el manejo de imágenes del tipo jpeg, pero usa la librería externa para jpeg, que puede ser obtenida en una de las versiones de GBM que la implementa. Además se debe de dar crédito al grupo IJG (Independent JPEG Group) [2] para que sea permitido su uso.

Entre los formatos más comunes (incluyendo todas sus extensiones) que permite manipular la librería se encuentran: Bitmap, GIF, PCX, TIFF, JPEG, entre otras más.

GraphApp : Es una librería para gráficos en C. Permite trabajar con diferentes sistemas operativos como lo son Windows-XP, UNIX, Linux y Macintosh. De igual manera permite que el programador haga uso de botones, cajas de texto entre otros objetos dentro de la aplicación.

El autor de dicha librería es L. Patrick quien es profesor en la Universidad de Sydney en Australia para el departamento de computación. El proyecto de la librería GraphApp nace como parte su trabajo de titulación.

La librería es distribuida bajo el licenciamiento GNU (GNU Library General Public License), pero aun cuando el software se distribuye gratuitamente no asegura la completa funcionalidad de este.

Jpeglib: Esta librería es desarrollada bajo la norma de distribución y licenciamiento GNU y creada por el grupo independiente IJG. Esta librería tiene como propósito el procesamiento de imágenes en formato JPEG. Cuenta con dos cabeceras principales llamadas *jconfig.h* y *jmorecfg.h*. Da soporte para realizar el procesamiento sobre Pocket PC. Además existe una versión llamada IJG Win32 que es ocupada como un complemento o "*Solution*" para .Net (Visual C++ .Net Solution), que incluye lossless JPEG para evitar la una mayor perdida al momento de realizar la compresión [IJG].

3.2.2. Instalación de la librería SDL dentro de WxDevcpp

Para hacer uso de esta librería no se requiere más configuración que descargar tres archivos comprimidos (*.zip* para la versión de Windows) y cargar su contenido en las respectivas carpetas. El primer archivo *SDL-devel-1.2.8-mingw32.zip* cuenta con varios archivos y carpetas como *"bin"*, *"lib"*, *"include"*, etc. De estos nos interesa la carpeta

[2] Es un grupo independiente el cual escribe y distribuye gratuitamente librerías generales para imágenes JPEG.

llamada *include* que contiene otra carpeta llamada *"**SDL**"*, que debe ser copiada dentro de nuestro entorno de trabajo de WxDevcpp en la carpeta *include* ya existente.

Como ejemplo de una ruta para localizar estos archivos tenemos a: *ROOT_DESCOMPRIMIDO\SDL-1.2.8\include**SDL***, seleccionando el subdirectorio completo para ser copiado y quedando en una ruta similar a la que a continuación se muestra: *C:\Archivos de programa\Dev-Cpp\include**SDL.***

El siguiente archivo que se copia es *SDL.dll* (identificando a los archivos dll como las librerías de enlace dinámico *dinamic link library*) que se encuentra dentro de la carpeta *"bin"* en el lugar donde descomprimimos nuestro archivo, este archivo se coloca en la carpeta de system dentro de WINDOWS (*C:\WINDOWS\system*) con la intención de ser localizado al momento de la ejecución de los programas que lo utilizan.

Es pertinente comentar que en caso de la exportación o ejecución en otro procesador diferente al de desarrollo es indispensable incluir los archivos "*.dll*" junto con el archivo de extensión "*.exe*".

De los archivos *SDL_mixer-devel-1.2.6-VC6* y *SDL_image-devel-1.2.4-VC6*, sólo se crean dos carpetas para cada uno al descomprimirlos, *"lib"* e *"include"*, de las cuales realizaremos el mismo procedimiento que en el anterior. Dentro de la carpeta *"include"* se encuentra *"SDL"* y ahora sólo se copia el archivo *.h* que se encuentra (*SDL_mixer.h* y *SDL_image.h* respectivamente). De la carpeta lib, copiamos dentro de system todos los archivos *.dll* que *jpeg.dll*, *libpng13.dll*, *SDL_image.dll, SDL_mixer.dll* y *zlib1.dll*.

Por último dentro del entorno Dev-Cpp en la etiqueta de "Herramientas" y posterior mente en las "Opciones de Compilación" se agregan las siguientes líneas: *-lmingw32 -lSDLmain -lSDL -lSDL_image -lSDL_mixer* en la sección que se identifica como "Añadir estos comandos a la línea de comandos del linker". Así termina la configuración de la librería quedando integrada para su utilización en los programas de desarrollo de C++.

Tabla 3.1 Dos de los subsistemas de la librería SDL y su uso.

Librería	Subsistema	Funcionalidad
SDL	SDL_mixer	Utilizada para el manejo de archivos con extención CMD, MDI, WAV, MOD, OGG y MP3; que son reproducidos por la aplicación.
	SDL_image	Usada para cargar archivos BMP, PNM, XPM, XCF, PCX, GIF, JPG, TIF, PNG, LBM y realizar la manipulación.

La Tabla 3.1 nos muestra dos de lso subsistemas que tiene la librería SDL y que son de utilidad para el desarrollo del procesamiento de la imagen.

Capítulo 4

Análisis y generación de identificador único por biometría del iris

Éste capítulo presenta el funcionamiento de la aplicación generada empleando la metodología descrita en la sección 3.1 del Capítulo 3, así es posible la generación sucesiva de código que permite la obtención del número binario y posteriormente el código de barras con lo que es posible identificar a un individuo. El flujo del proceso inicia cargando la imagen, a continuación se extrae tanto el límite interno como el externo con lo que se obtiene una imagen resultante en forma de dona del iris. Al resultado se le aplica un cambio de coordenadas para así finalmente aplicar el método de ondeletas para conseguir una imagen de código binario correspondiente a la figura inicial.

Una vez iniciada la aplicación "ProyectoTesis" damos clic en el botón "Abrir" la cual abre una nueva ventana que nos muestra una imagen de ojo.

4.1. Obtención del límite interno

Para realizar la extracción del iris se comienza con la extracción del límite interno, es decir, se busca obtener la parte del iris que toca a la pupila y al mismo tiempo es removida esta parte de la imagen original. Mediante este proceso se genera una imagen como la que se presenta en la Figura 4.1.

Figura 4.1 Imagen a la que se le ha removido el área de la pupila.

Siguiendo el método descrito en la sección 3.1, se realiza la umbralización con un grado de 60 (damos clic en el botón "Umbralización 60"), que no es más que obtener la parte más oscura de la imagen, que para el caso es el área de la pupila. La Figura 4.2 es una imagen que muestra la zona de la pupila obtenida al aplicar este proceso.

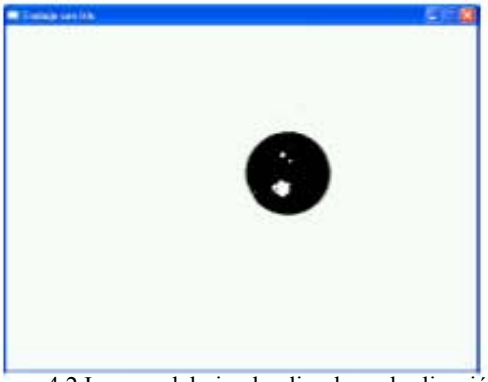

Figura 4.2 Imagen del ojo al aplicarle umbralización.

Como se aprecia en la Figura 4.2 la imagen de la pupila obtenida, tiene manchas originadas por el reflejo de la luz en el ojo y lo que se conoce como basura que en otras palabras tan solo son píxeles que no son relevantes para el proceso. Dado esto, se somete la nueva imagen a los procesos de dilatación y erosión para obtener un área de pupila completamente circular y en base a ello poder obtener el radio y al mismo tiempo el límite interior.

Lo que hace la función de dilatación y de erosión (en el botón "Dilatación y Erosión") es que borra los píxeles que no sean necesarios y agregará en zonas blancas para rellenar el área deseada, dicho de otra forma volverá blancos los píxeles que se encuentran a su alrededor o lo que es llamado vecindad. En el otro caso volverá a al píxel y a los cuatro que estén a su alrededor (izquierda, derecha, arriba y abajo) de color negro y así completar el área. La Figura 4.3 presenta el proceso hasta obtener la imagen deseada.

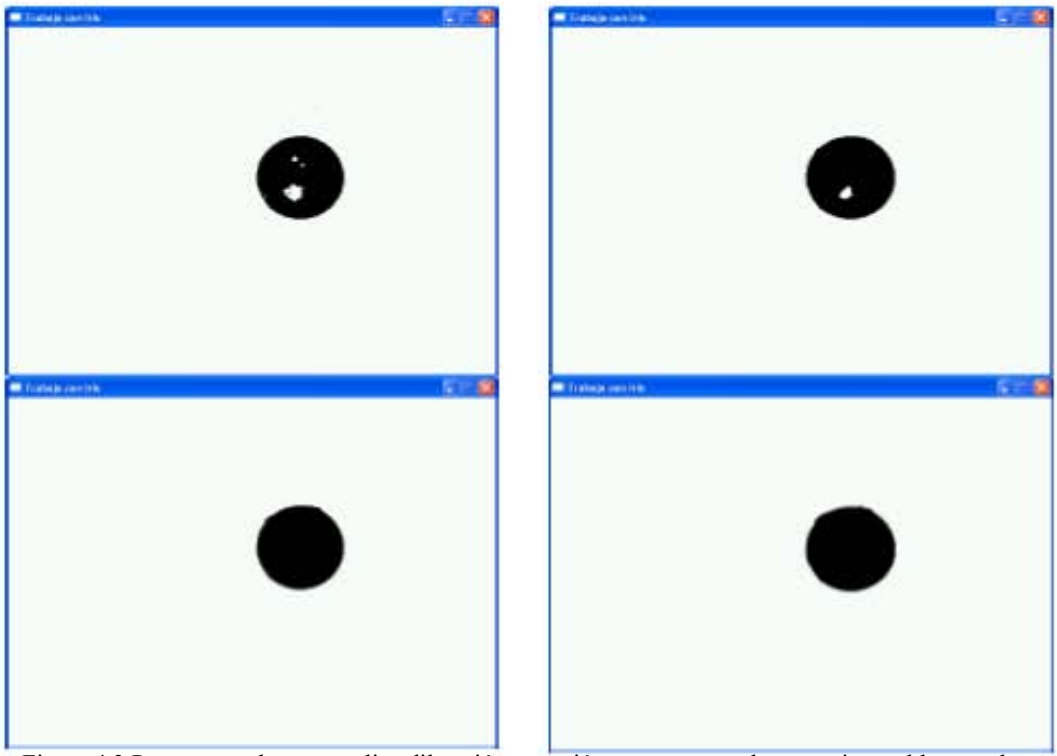

Figura 4.3 Proceso en el que se aplica dilatación y erosión para remover los espacios en blanco y los puntos fuera del área de la pupila.

Por cada vez que se repite el proceso la nueva imagen que se va obteniendo muestra el área de la pupila cada vez más redonda y va cubriendo los espacios en blanco que tiene en el interior. Dentro de la ejecución del programa se ha hecho que se repita el proceso un total de 15 veces para asegurar que el área tenga las características deseadas.

Aun cuando dentro del proceso completo (botón "Todo el proceso") no se aprecia la imagen generada que muestra la extracción del límite interno, el siguiente paso es obtenerla. Para ello debemos extraer el radio de la circunferencia obtenida en el paso anterior, aquí es donde se aplica la revisión de la imagen para encontrar la línea horizontal que contenga más píxeles de color negro y de igual forma la línea vertical. Jutas estas líneas y calculado el píxel de en medio de ambas obtendremos el centro de la pupila.

Posteriormente es necesario revisar que toda el área encontrada sea cambiada de tonos grisáceos y negro claro al negro más alto. Así realizando un proceso similar al de umbralización pero ahora con las partes más oscuras removeremos toda la región de la pupila. La Figura 4.4 muestra como el área de la pupila ha sido retocada en color totalmente negro para después ser removida.

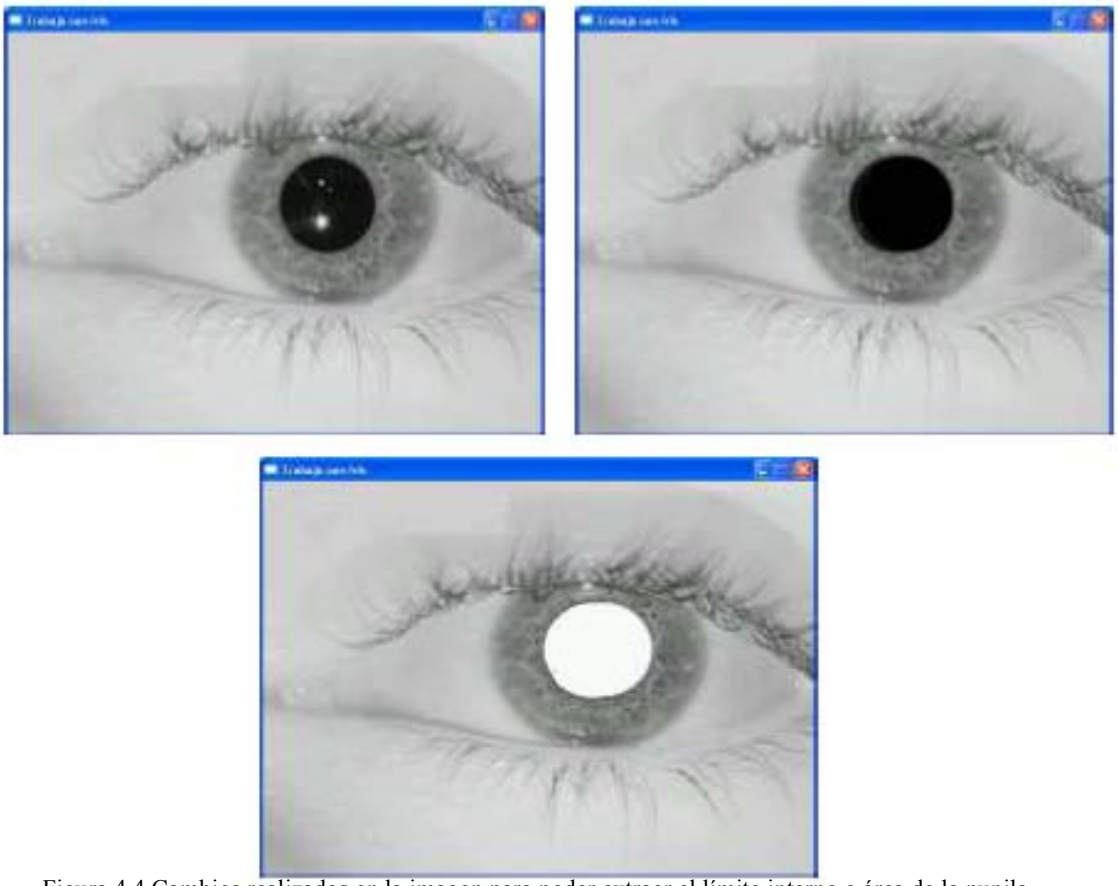

Figura 4.4 Cambios realizados en la imagen para poder extraer el límite interno o área de la pupila.

Hasta este momento se ha extraído la zona de la pupila con lo que se conoce el límite interno del iris ("Límite Interno" y "Borrar Límite Interno"). Esta es la primera fase del proceso y a continuación se debe realizar un proceso semejante con el fin de extraer el límite externo, lo que dará como resultado la obtención de una imagen solo del iris.

4.2. Obtención del límite externo

De la misma forma que al extraer el límite interno, se desea obtener el límite donde se tocan el iris y la esclerótica. Al obtener este límite, volveremos blancos todos los píxeles que no se encuentren dentro de esta área y al sumarle la extracción del límite interno obtendremos una imagen en forma de dona correspondiente al iris.

Para lograr obtener el límite externo reutilizaremos datos que ya obtuvieron en el paso anterior, como lo son las ubicaciones del centro de la pupila que al final son el mismo para el iris. Estos datos son almacenados en una variable durante todo el proceso hasta que finalice o se reinicie el proceso.

Ahora para poder saber donde se encuentra el límite externo se usan las coincidencias entre píxeles y sumado a las coordenadas del centro de la pupila obtendremos el límite. Las coincidencias son tomadas en una vecindad que incluye los 8 vecinos al píxel que se va verificando durante el proceso, es decir, se evalúan los píxeles circundantes al que durante la revisión conoceremos como píxel actual. La Tabla 4.1 representa una matriz que toma como elementos los píxeles circundantes al actual y así obtener una vecindad.

Tabla 4.1 Representa una vecindad en relación a un píxel.

Pixel x-1, y-1	Píxel x, y-1	Píxel x+1, y-1
Píxel x-1, y	Píxel x, y	Píxel x+1, y
Píxel x-1, y+1	Píxel x, y+1	Píxel x+1, y+1

Para validar que se cumplan las coincidencias se crea una matriz de 3 x 3 donde la posición (2,2) representa al píxel actual y el resto es la vecindad. Dentro de la matriz se le asigna un valor de 8 a nuestro píxel actual, lo cual equivale al número de vecinos que tiene. Al resto les asignaremos el número uno siempre y cuando este coincida con el color de nuestro píxel actual.

Revisando todos los píxeles circundantes a nuestro píxel actual a partir del centro encontrado y asignándole los valores enunciados arriba, se realiza una suma de todos los píxeles circundantes, si la suma de estos resulta ser ocho se dice que tenemos una coincidencia y el método que se encarga de realizar dicha revisión contiene un contador que llevara el número de coincidencias.

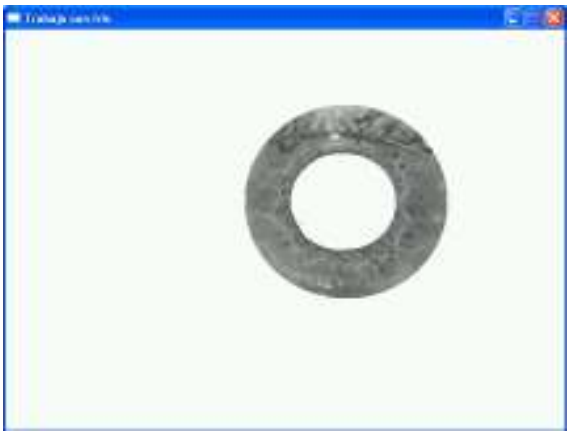

Figura 4.5 Imagen de la extracción del iris.

La imagen 4.5 muestra como se ve finalmente una imagen del iris. Al obtenerla lo que resta es aplicar un cambio de coordenadas para obtener una figura rectangular, es decir se desdobla el iris para continuar con el análisis.

Para realizar la extracción del límite externo al ejecutar la aplicación, no se extraé el límite interno debido a que por simplicidad se encuentra incrustada su invocación y se realiza todo en conjunto al dar clic sobre el botón “Límite Externo”.

4.3. Desdoblamiento del iris mediante conversión de coordenadas

Ya que la imagen resultante es de forma circular por la naturaleza del ojo, el siguiente paso del análisis es realizar un cambio de coordenadas polares a cartesianas sobre la imagen (botón “Coordenadas Cartesianas”) con el fin de obtener una nueva con forma rectangular que nos permitirá el análisis mediante de ondeletas.

Básicamente lo que se hace es revisar los píxeles en sentido de las manecillas del reloj que se encuentran dentro de la circunferencia, realizar la transformación y guardarla en una matriz temporal, para posteriormente pintarla en su nueva forma. La nueva área que se genera tiene dimensiones aproximadas de 450 x 60 píxeles (ancho x alto).

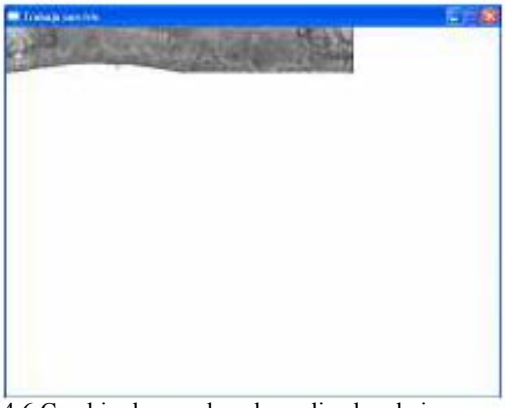

Figura 4.6 Cambio de coordenadas aplicado a la imagen del iris.

En la Figura 4.6 se aprecia el resultado de realizar un cambio de coordenadas polares a cartesianas, lo que implica que ahora es posible trabajar con una imagen plana que resulta mucho más fácil que trabajar con la circunferencia para hacer el análisis.

4.3.1. Recorte de la nueva área del iris

Como paso intermedio antes de poder extraer el código, es necesario redimensionar el área obtenida (botón “Redimensiona”) ya que como se indica en el método [MAGAÑA, 2005] es necesario que la imagen sea divisible entre 16, es por ello que el método propone que la imagen tenga un tamaño de 448 x 48 píxeles que no implica una gran perdida de píxeles, lo que significa que no es grande la perdida de información. La Figura 4.6 presenta a la imagen redimensionada con el ancho y alto propuesto.

El código empleado lo único que realiza es un corte de píxeles, los cuales son manejados como si se tratara de una matriz, esto permite sólo mantener aquellos que se encuentren en los 448 píxeles en el ancho de la imagen y los 48 píxeles de alto.

Figura 4.6 Con las nuevas dimensiones del iris de 448 x 48.

Con la imagen obtenida es posible empezar la extracción del código al aplicar las ondeletas y de esta manera reducir o comprimir la imagen de tal manera que la imagen sea lo suficientemente pequeña para continuar con el análisis.

4.4. Ondeletas y extracción de código

Como se ha mencionado debemos comprimir la imagen un total de cuatro veces, esto no implica que la imagen pierda propiedades que son necesarias para la extracción del código único.

En general para lograr la compresión (cada compresión es iniciada por los botones que van del 1 al 4 con la descripción "Ondeletas…"), se recorren, mediante un ciclo con incremento de dos, todos los píxeles correspondientes al ancho y alto de la última imagen obtenida del iris (en el caso de la primer compresión es de 448 x 48). Al recorrer todos los píxeles se obtiene un promedio del color que tiene el píxel actual y los píxeles circundantes. El color promedio es el color que se muestre en la nueva posición determinada por un contador que dentro del mismo ciclo tiene un incremento de uno.

El resultado final al comprimir la imagen de 448 x 48 es una nueva con dimensiones de 224 x 24, la siguiente tiene un tamaño de 112 x 12, posteriormente una de 56 x 6 y finalmente la imagen tendrá 28 x 3 píxeles. La Figura 4.7 muestra el proceso de compresión hasta obtener una nueva imagen de 28 x 3.

Figura 4.7 Compresión aplicada 4 veces de la imagen presentada en la Figura 4.6.

Durante el proceso se guardan tres variables correspondientes a las primeras 3 compresiones, estas nos ayudará a determinar el código binario pues son el complemento al número de 84 bits que se obtiene. Al agregarle el valor de estas tres variables obtendremos un número con un total de 87 bits. Estas variables se obtienen al restar al color del píxel actual el valor de dos píxeles vecinos (vecino derecho y vecino inferior), posteriormente agregarle el valor de del vecino inferior derecho y finalmente realizar la división entre cuatro. El resultado es vuelto a dividir entre el número de píxeles que se encuentran en la nueva área, por ejemplo al reducir a 224 x 24 la división de la variable es entre 5376 y así para las demás áreas.


Figura 4.8 Resultado de extracción de código de barras.

Con la última compresión realizamos la verificación para generar el código de barras (iniciamos el proceso al dar clic sobre el botón “Extracción de Código”). Este es formado al evaluar todos los píxeles contenidos dentro del área de 28 x 3, que nos dán un total de 84 píxeles internos que sumados a tres variables que se guardan durante las compresiones anteriores nos dan un total de 87 dígitos.

Los digitos se calculan por el color que tiene cada píxel evaluado de igual manera a como se realiza en las demás compresiones. Si la evaluación resulta ser mayor o igual a cero se le asignará al píxel el color blanco y al código se le agrega un número 1. En caso contrario el píxel se pinta negro y se le asigna al código el número 0. Finalmente agragamos los píxeles que corresponden a los bits de complemento que son extraidos de las variables originadas durante cada compresión. Dichas variables contienen un número del mismo tipo al que tienen los generados al realizar el análisis de los píxeles en el área de 28 x 3 y se le asigna el 0 ó 1 de acuerdo al mismo criterio.

Por último se realiza una extención de los nuevos píxeles por un total de 100 veces con lo que al mostrarlos en pantalla se tiene la apariencia del código de barras. La Figura 4.8 presenta un resultado obtenido al realizar todo el proceso.

4.5. Funciones complementarias

Las siguietes funciones no se usan directamente dentro del proceso de análisis, pero son importantes para lograr que el procedimiento continúe su flujo y que la imagen mantenga las características necesarias para realizar el procesamiento. Todas ellas cuentan con un botón con el nombre de la función a la que hace referencia.

Play música y stop

Estas funciones reproducen una melodía o la detienen. La melodía se reproduce automáticamente después de dar clic en el botón “Abrir”. Dicha melodía puede ser detenida o reproducida en cualquier momento.

Se ha incertado este tipo de funciones con el fin de mostrar la potencia de la librería SDL, descrita en la sección 3.2 del capítulo 3. Internamente el código se encarga de cargar la melodía que se encuentra en la ruta dada, así mismo le indica cuando detenerse según sea el caso.

Crea píxel

Esta función esta encargada de dibujar un punto (píxel) de color verde en el lugar en donde se cruzan la línea mayor vertical como la línea mayor horizontal, es decir, se encarga de dar un aproximado de la localización del centro de la pupila y el iris. La Figura 4.9 muestra un ejemplo de la función “Crea Píxel”.

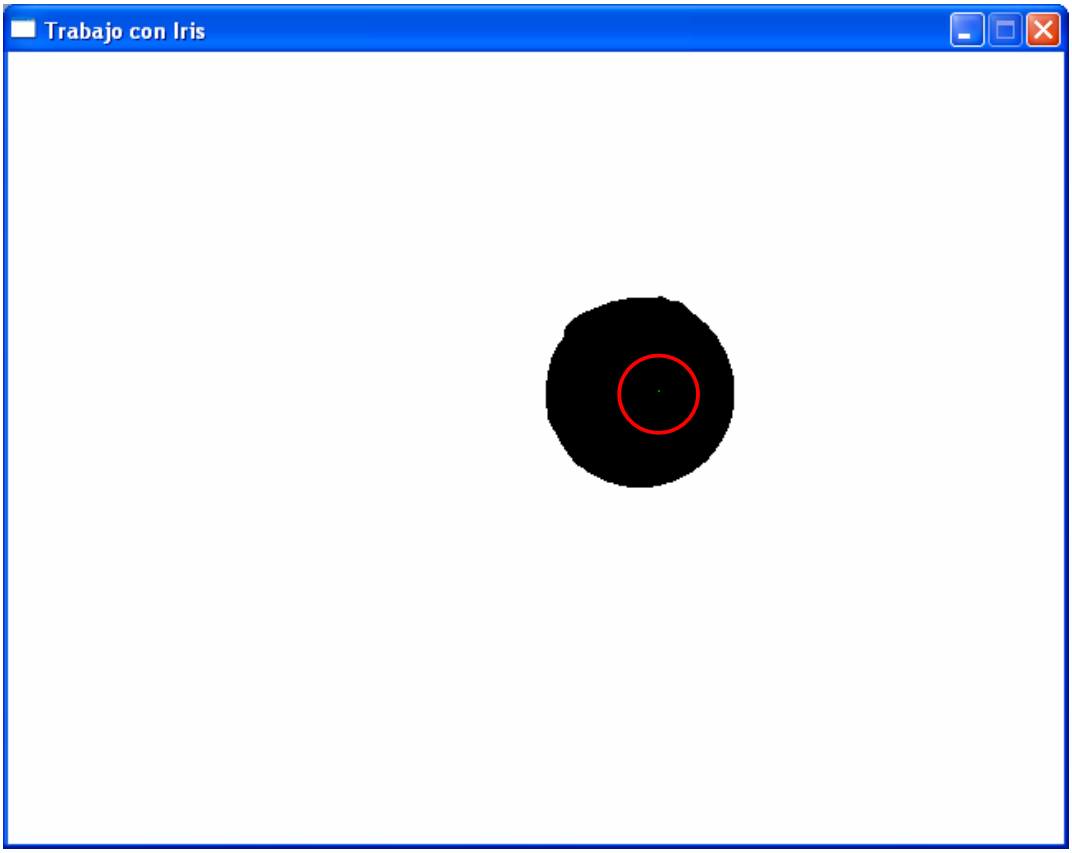

Figura 4.9 Ejemplo de la aproximación del centro calculado.

Refresca

Esta función permite cargar nuevamente la imagen original sin que esta haya perdido sus propiedades ya que nunca se guarda el resultado ni modificación alguna.

Estructura de datos

Esta función permite conocer diferentes datos de la imagen abierta. Entre los datos que se pueden extraer se encuentran el alto y el ancho de la imagen, los colores que conforman todos y cada uno de los píxeles. Por la naturaleza de la librería, estos datos son puestos en un archivo de texto llamado "stdout.txt" donde son guardadas todas las llamadas a la funcion "printf" de C. El siguiente texto es un ejemplo de la llamada a la función "Estrucutra de Datos":

```
PRUEBA FINAL SOBRE PIXELES

Prueba height 480
Prueba width 640

ÉSTE ES EL RENGLÓN 0

Prueba otroPixel [0] 13684944
Prueba red   208
Prueba green 208
Prueba blue  208

Prueba otroPixel [1] 13684944
Prueba red   208
Prueba green 208
Prueba blue  208
```

```
Prueba otroPixel [2] 13684944
Prueba red   208
Prueba green 208
Prueba blue  208

Prueba otroPixel [3] 13684944
Prueba red   208
Prueba green 208
Prueba blue  208
.
.
.
```

Limpiar pantalla

La función limpiar pantalla tiene por objetivo poner en blanco toda la superficie en donde se encuentra la figura. El método se encarga de poner en blanco todos los píxeles que se han cargado para formar la figura. Una vez hecho esto es posible cargar nuevamente la imagen original. En la Figura 4.10 se aprecia la superfície totalmente en blanco al aplicar la función.

Figura 4.10 Superficie en blanco.

Cambio de imagen a color en escala de grises

Esta función permite transformar una imagen a color en otra en escala de grises. Se ha creado esta función con el propósito de poder trabajar con imágenes que no se encuentran en escala de grises, ya que el método así lo requiere.

Lo que se hace para obtener la nueva imagen en color gris es analizar cada píxel que conforma la imagen y comparar el grado en cada uno de los tres colores básicos (RGB – rojo, verde, azul por sus siglas en inglés) y convertir a todos en el tono del color con el valor más alto. Es decir que si el azul tiene un valor de 125 y tanto rojo como verde tienen un valor inferior, todos los demás cambian su valor al del color azul. La figura 4.11 muestra el cambio de color de una figura.

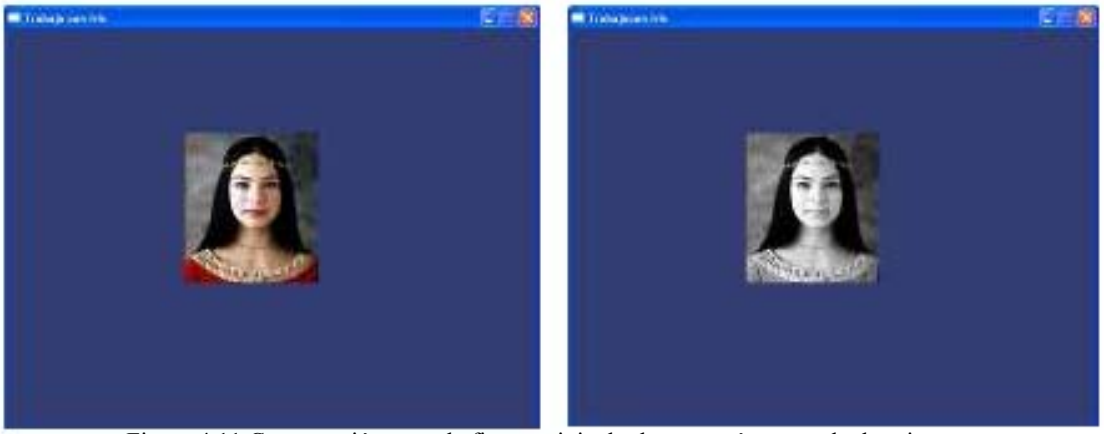

Figura 4.11 Comparación entre la figura original y la que está en escala de grises.

Todas estas funciones son llamadas dentro del proceso para facilitar el procesamiento de imágenes. Ya que en ocaciones es necesario reutilizar la imagen original despues de haber aplicado un proceso, también es necesario llevar la pantalla en blanco para poder píntar solo los píxeles que son necesarios, entre otras operaciones intermedias al proceso.

Conclusiones y trabajos futuros

La conversión metodológica y sus elementos estructurales del procedimiento de análisis de una imagen digital ojo-iris utilizando morfología matemática difusa ha sido presentada, incluyendo sus diefrencias con respecto a la aplicación de morfología matemática estándard.

La integración de la herramienta libreria SDL como agente de procesamiento de imágenes en formato JPG en un ambiente de desarrollo integrado (IDE) en C++ se ha ilustrado en detalle definiendo sus principales funciones y la forma de su utilización.

Se ha presentado la estructura general de manipulación de la imagen del ojo para obtener como salida código binario reducido a 87 bits en contraste con anteriores propuestas de 128 bits.

Se han descrito las partes necesarias para integrar un esquema de reconocimiento biométrico por medio del iris, empleando imágenes en escala de grises con información comprimida a través de ondeletas de Haar.

Se han introducido herramientas de software libre con la intención de ofrecer alternativas al software comercial bajo licencia, lo que amplia el desarrollo sin las complicaciones de patentes o algoritmos registrados.

Los tiempos de respuesta indican una reducción de más del 50% para el análisis completo en comparación con el uso de paquetes comerciales no dirigidos exclusivamente al procesamiento de imágenes como MATLAB o MATEMATICA.

Tabla 5.1 Comparación de procesos en cuanto tiempo

Proceso	Tiempo (promedio)
Proceso completo C++	7.5 segundos
Proceso completo MATLAB	50 segundos

En la Tabla 5.1 se muestra una comparativa entre el tiempo efectuado por las dos aplicaciones. El tiempo de ejecución para la aplicación desarrollada para MATLAB fue tomara de [MAGAÑA. 2005].

En cuanto a tiempo se refiere, la aplicación desarrollada en C++ de este trabajo efectua con mayor velocidad los procesos al no requerir de la participación del usuario para activar los siguientes proceso. De esta manera se agiliza cuando se ejecuta sin interrupción.

En cuanto al desempeño de los algoritmos, no es posible efectuar una comparación precisa de las funciones del trabajo desarrollado en MATLAB ya que se trata de funciones propias del programa y no se cuenta con el código. Cabe mencionar que dichas funciones resultan más precisas (por un par de píxeles) en los calculos respectivos a la localización de los límites tanto interno como externo al verse reflejado por los resultado obtenidos.

Tabla 5.2 Estimación de tiempo máquina por proceso MATLAB [MAGAÑA, 2005]

SUB PROCESO	TIEMPO MÁQUINA
Alisamiento difuso de la imagen del ojo	80%
Detección del límite interno	06%
Detección del límite externo	05%
Extracción del iris	01%
Conversión de coordenadas	02%
Corrección difusa de la imagen	02%
Extracción del código	04%

Tabla 5.3 Estimación de tiempo máquina por proceso C++

SUB PROCESO	TIEMPO MÁQUINA
Alisamiento difuso de la imagen del ojo	15%
Detección del límite interno	05%
Detección del límite externo	09%
Extracción del iris	05%
Conversión de coordenadas	02%
Corrección difusa de la imagen	04%
Extracción del código	60%

En la Tabla 5.2 y 5.3 se muestra respectivamente el tiempo máquina utilizado por cada sub proceso dentro de las diferentes aplicaciones.

La codificación en lenguaje C de las principales funciones implantadas en la propuesta se presentan en forma parcial en el texto general del documento y completas en los anexos.

Trabajos futuros

La investigación contempla incluir como propuestas complementarias:

1) Entrada de imágenes en color y su transformación a escala de grises por su facilidad de manejo.
2) Formación de banco de imágenes de ojo-iris con propósitos de reconocimiento biométrico.
3) Diseño y prototipo de mecanismos mecánicos de captura de imagen y respuesta de aceptación o rechazo a la identificación individual.

Anexo 1

Código fuente

ProyectoTesisApp.cpp

```
//---------------------------------------------------------------------------
//
// Name:        ProyectoTesisApp.cpp
// Author:      Jafet Malvaez
// Created:     14/03/2007 04:40:10 p.m.
// Description:
//
//---------------------------------------------------------------------------

#include "ProyectoTesisApp.h"
#include "ProyectoTesisDlg.h"

IMPLEMENT_APP(ProyectoTesisDlgApp)

bool ProyectoTesisDlgApp::OnInit()
{
	ProyectoTesisDlg* dialog = new ProyectoTesisDlg(NULL);
	SetTopWindow(dialog);
	dialog->Show(true);
	return true;
}

int ProyectoTesisDlgApp::OnExit()
{
	return 0;
}
```

ProyectoTesis.h

```
//
// Name:        ProyectoTesisApp.h
// Author:      Jafet Malvaez
// Created:     14/03/2007 04:40:10 p.m.
// Description:
//
//---------------------------------------------------------------------------

#ifndef __ProyectoTesisApp_h__
#define __ProyectoTesisApp_h__

#ifdef __BORLANDC__
	#pragma hdrstop
#endif

#ifndef WX_PRECOMP
	#include <wx/wx.h>
#else
	#include <wx/wxprec.h>
#endif

class ProyectoTesisDlgApp : public wxApp
{
	public:
		bool OnInit();
		int OnExit();
};

#endif
```

ProyectoTesisApp.rc

```
#include <wx/msw/wx.rc>
```

ProyectoTesisDlg.cpp

```
//---------------------------------------------------------------------------
//
// Name:        ProyectoTesisDlg.cpp
// Author:      Jafet Malvaez
// Created:     14/03/2007 04:40:10 p.m.
// Description:
//
//---------------------------------------------------------------------------

#include "ProyectoTesisDlg.h"

//Do not add custom headers
//wxDev-C++ designer will remove them
////Header Include Start
#include "ProyectoTesisDlg_XPM.xpm"
////Header Include End

//---------------------------------------------------------------------------
// ProyectoTesisDlg
//---------------------------------------------------------------------------
//Add Custom Events only in the appropriate block.
//Code added in other places will be removed by wxDev-C++
////Event Table Start
BEGIN_EVENT_TABLE(ProyectoTesisDlg,wxDialog)
	////Manual Code Start
	////Manual Code End

	EVT_CLOSE(ProyectoTesisDlg::ProyectoTesisDlgClose)
	EVT_BUTTON(ID_WXBUTTON23,ProyectoTesisDlg::WxButton23Click)
	EVT_BUTTON(ID_WXBUTTON22,ProyectoTesisDlg::WxButton22Click)
	EVT_BUTTON(ID_WXBUTTON21,ProyectoTesisDlg::WxButton21Click)
	EVT_BUTTON(ID_WXBUTTON20,ProyectoTesisDlg::WxButton20Click)
	EVT_BUTTON(ID_WXBUTTON19,ProyectoTesisDlg::WxButton19Click)
	EVT_BUTTON(ID_WXBUTTON18,ProyectoTesisDlg::WxButton18Click)
	EVT_BUTTON(ID_WXBUTTON17,ProyectoTesisDlg::WxButton17Click)
	EVT_BUTTON(ID_WXBUTTON16,ProyectoTesisDlg::WxButton16Click)
	EVT_BUTTON(ID_WXBUTTON15,ProyectoTesisDlg::WxButton15Click)
	EVT_BUTTON(ID_WXBUTTON14,ProyectoTesisDlg::WxButton14Click)
	EVT_BUTTON(ID_WXBUTTON13,ProyectoTesisDlg::WxButton13Click)
	EVT_BUTTON(ID_WXBUTTON12,ProyectoTesisDlg::WxButton12Click)
	EVT_BUTTON(ID_WXBUTTON11,ProyectoTesisDlg::WxButton11Click)
	EVT_BUTTON(ID_WXBUTTON10,ProyectoTesisDlg::WxButton10Click)
	EVT_BUTTON(ID_WXBUTTON9,ProyectoTesisDlg::WxButton9Click)
	EVT_BUTTON(ID_WXBUTTON8,ProyectoTesisDlg::WxButton8Click)
	EVT_BUTTON(ID_WXBUTTON7,ProyectoTesisDlg::WxButton7Click)
	EVT_BUTTON(ID_WXBUTTON6,ProyectoTesisDlg::WxButton6Click)
	EVT_BUTTON(ID_WXBUTTON5,ProyectoTesisDlg::WxButton5Click)
	EVT_BUTTON(ID_WXBUTTON4,ProyectoTesisDlg::WxButton4Click)
	EVT_BUTTON(ID_WXBUTTON3,ProyectoTesisDlg::WxButton3Click)
	EVT_BUTTON(ID_WXBUTTON2,ProyectoTesisDlg::WxButton2Click)
	EVT_BUTTON(ID_WXBUTTON1,ProyectoTesisDlg::WxButton1Click)
END_EVENT_TABLE()
////Event Table End

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <windows.h>
#include <SDL/SDL.h>
#include <SDL/SDL_image.h>
#include <SDL/SDL_mixer.h>

/* Declaración de funciones */

void limpiar(SDL_Surface *screen);
void limpiarImg(SDL_Surface *surface);
void imprimir(SDL_Surface *screen, SDL_Surface *ima, int x, int y);
void mover_objeto(int *x, int *y, int ancho, int alto, SDL_Surface *screen,
Mix_Music *cancion);
void escituraDatos(SDL_Surface *surface);
void creaPixel (SDL_Surface *surface);
Uint32 getpixel(SDL_Surface *surface, int x, int y);
void putpixel(SDL_Surface *surface, int x, int y, Uint32 pixel);
void umbralizacion(SDL_Surface *surface, int t);
```

```
void morfologia(SDL_Surface *surface);
void morfoClean(SDL_Surface *surface);
void morfoErosion(SDL_Surface *surface, int i, int j);
void morfoDilation(SDL_Surface *surface);
void aGris(SDL_Surface *surface);
int lineaV(SDL_Surface *surface);
int lineaH(SDL_Surface *surface, int centroRadio);
void copiaImage(SDL_Surface *surface);
void limiteInterno(SDL_Surface *surface);
void BorraLimInterno(SDL_Surface *surface);
void limiteExterno(SDL_Surface *surface);
void coordPolares(SDL_Surface *surface);
void resize(SDL_Surface *surface);
void extraeCodigo(SDL_Surface *surface, int imprime);

/* Esta variable contiene los nombres de todas la imágenes con las que se
hacen las pruebas */

char     *files[]={"Img/kreuk.jpg",     "Img/kreuk2.jpg",     "Img/iris.jpg",
"Img/iris2.jpg", "Img/pupil.jpg", "Img/iris.jpg", "Img/O111223042.JPG"};

/*
 * Estas variables son generales para todo el programa ya que contienen los
centros y tamaños de rádio
 *  necesarios  para  realizar  los  cálculos,  los  cuales  no  pueden  ser
recalculados conforme se va modificando la imagen
 *
 */

int tmpRadio, tmpRadioC, tmpXcentro, tmpYcentro;

SDL_Surface *surfaceGeneral;
Mix_Music *cancion;

ProyectoTesisDlg::ProyectoTesisDlg(wxWindow  *parent,  wxWindowID  id,  const
wxString &title, const wxPoint &position, const wxSize& size, long style)
: wxDialog(parent, id, title, position, size, style)
{
    CreateGUIControls();
}

ProyectoTesisDlg::~ProyectoTesisDlg() {}

void ProyectoTesisDlg::CreateGUIControls(void)
{
    //Do not add custom code here
       //wxDev-C++ designer will remove them.
       //Add the custom code before or after the blocks
       ////GUI Items Creation Start

       this->SetSize(8,8,192,736);
       this->SetTitle(wxT("ProyectoTesis"));
       this->Center();
       wxIcon ProyectoTesisDlg_ICON (ProyectoTesisDlg_XPM);
       this->SetIcon(ProyectoTesisDlg_XPM);


       WxButton23  =  new  wxButton(this,  ID_WXBUTTON23,  wxT("Redimenziona"),
wxPoint(50,485), wxSize(75,25), 0, wxDefaultValidator, wxT("WxButton23"));

       WxButton22  =  new  wxButton(this,  ID_WXBUTTON22,  wxT("Extracción  de
Código"),    wxPoint(33,635),    wxSize(111,25),    0,    wxDefaultValidator,
wxT("WxButton22"));

       WxButton21  =  new  wxButton(this,  ID_WXBUTTON21,  wxT("Ondeletas  4"),
wxPoint(50,605), wxSize(75,25), 0, wxDefaultValidator, wxT("WxButton21"));

       WxButton20  =  new  wxButton(this,  ID_WXBUTTON20,  wxT("Ondeletas  3"),
wxPoint(50,575), wxSize(75,25), 0, wxDefaultValidator, wxT("WxButton20"));

       WxButton19  =  new  wxButton(this,  ID_WXBUTTON19,  wxT("Ondeletas  2"),
wxPoint(50,545), wxSize(75,25), 0, wxDefaultValidator, wxT("WxButton19"));

       WxButton18  =  new  wxButton(this,  ID_WXBUTTON18,  wxT("Ondeletas  1"),
wxPoint(50,515), wxSize(75,25), 0, wxDefaultValidator, wxT("WxButton18"));
```

```
	WxButton17 = new wxButton(this, ID_WXBUTTON17, wxT("Coordenadas Polares"), wxPoint(36,455), wxSize(109,25), 0, wxDefaultValidator, wxT("WxButton17"));

	WxButton16 = new wxButton(this, ID_WXBUTTON16, wxT("Límite Externo"), wxPoint(50,425), wxSize(75,25), 0, wxDefaultValidator, wxT("WxButton16"));

	WxButton15 = new wxButton(this, ID_WXBUTTON15, wxT("Borrar Límite Interno"), wxPoint(36,395), wxSize(105,25), 0, wxDefaultValidator, wxT("WxButton15"));

	WxButton14 = new wxButton(this, ID_WXBUTTON14, wxT("Límite Interno"), wxPoint(50,365), wxSize(75,25), 0, wxDefaultValidator, wxT("WxButton14"));

	WxButton13 = new wxButton(this, ID_WXBUTTON13, wxT("A Gris"), wxPoint(50,335), wxSize(75,25), 0, wxDefaultValidator, wxT("WxButton13"));

	WxButton12 = new wxButton(this, ID_WXBUTTON12, wxT("Limpiar Pantalla"), wxPoint(48,305), wxSize(82,25), 0, wxDefaultValidator, wxT("WxButton12"));

	WxButton11 = new wxButton(this, ID_WXBUTTON11, wxT("Estructura de Datos"), wxPoint(37,275), wxSize(104,25), 0, wxDefaultValidator, wxT("WxButton11"));

	WxButton10 = new wxButton(this, ID_WXBUTTON10, wxT("Crea Píxel"), wxPoint(50,245), wxSize(75,25), 0, wxDefaultValidator, wxT("WxButton10"));

	WxButton9 = new wxButton(this, ID_WXBUTTON9, wxT("Dilatación y Erosión"), wxPoint(34,215), wxSize(108,25), 0, wxDefaultValidator, wxT("WxButton9"));

	WxButton8 = new wxButton(this, ID_WXBUTTON8, wxT("Umbralización 130"), wxPoint(40,185), wxSize(97,25), 0, wxDefaultValidator, wxT("WxButton8"));

	WxButton7 = new wxButton(this, ID_WXBUTTON7, wxT("Umbralización 60"), wxPoint(43,155), wxSize(90,25), 0, wxDefaultValidator, wxT("WxButton7"));

	WxButton6 = new wxButton(this, ID_WXBUTTON6, wxT("Todo el proceso"), wxPoint(47,125), wxSize(85,25), 0, wxDefaultValidator, wxT("WxButton6"));

	WxButton5 = new wxButton(this, ID_WXBUTTON5, wxT("Stop"), wxPoint(50,95), wxSize(75,25), 0, wxDefaultValidator, wxT("WxButton5"));

	WxButton4 = new wxButton(this, ID_WXBUTTON4, wxT("Play Música"), wxPoint(50,65), wxSize(75,25), 0, wxDefaultValidator, wxT("WxButton4"));

	WxButton3 = new wxButton(this, ID_WXBUTTON3, wxT("Salir"), wxPoint(50,664), wxSize(75,25), 0, wxDefaultValidator, wxT("WxButton3"));

	WxButton2 = new wxButton(this, ID_WXBUTTON2, wxT("Refresca"), wxPoint(50,35), wxSize(75,25), 0, wxDefaultValidator, wxT("WxButton2"));

	WxButton1 = new wxButton(this, ID_WXBUTTON1, wxT("Abrir"), wxPoint(50,7), wxSize(75,25), 0, wxDefaultValidator, wxT("WxButton1"));
	////GUI Items Creation End
}

void ProyectoTesisDlg::ProyectoTesisDlgClose(wxCloseEvent& event)
{
    Destroy();
}


/*
 * WxButton1Click
 */
void ProyectoTesisDlg::WxButton1Click(wxCommandEvent& event)
{
     SDL_Surface *screen;
     SDL_Surface *ima;

     SDL_Event eventSDL;
     int salir = 0;
     int x = /*18*/0;
     int y = /*15*/0;

     if (SDL_Init(SDL_INIT_VIDEO | SDL_INIT_AUDIO) == -1){
```

```
          printf("Error: %s\n", SDL_GetError());
          //return 1;
     }

     screen = SDL_SetVideoMode(640, 480, 16, SDL_HWSURFACE);

     if (screen == NULL) {
          printf("Error: %s\n", SDL_GetError());
          //return 1;
     }

     if (Mix_OpenAudio(44100, MIX_DEFAULT_FORMAT, 2, 1024) == -1){
          printf("Error en Mix_OpenAudio: %s\n", Mix_GetError());
          //return 1;
     }

     SDL_WM_SetCaption("Trabajo con Iris", NULL);

     // Cargamos nuestra imagen
     ima = IMG_Load(files[2]);
     copiaImage(ima);

     if (ima == NULL){
          printf("Error en IMG_Load= %s\n", SDL_GetError());
          //return 1;
     }

     // Cargamos la melodía, de no existri no causa problemas, ya que no la
toma en cuanta en esos casos
     cancion = Mix_LoadMUS("Midi/ff8-Timber_Owls.mid");

     if (!cancion)
          printf("Error en Mix_LoadMUS: %s\n", Mix_GetError());

     if (cancion){
          if (Mix_PlayMusic(cancion, -1) == -1)
               printf("Error en Mix_PlayMusic: %s\n", Mix_GetError());
     }

     while (! salir){
          SDL_PollEvent(&eventSDL);

          if (eventSDL.type == SDL_QUIT)
               salir = 1;

          // Nos permite mover nuestra imagen por la pantalla
          surfaceGeneral = ima;
          mover_objeto(&x, &y, ima->w, ima->h, ima, cancion);

          // Con esta función mantenemos el fondo sin importar que se mueva la
imagen
          limpiar(screen);

          // Nos permite mostrar la imagen sobre el fondo
          imprimir(screen, ima, x, y);
          SDL_Flip(screen);
          SDL_Delay(10);
     }

     if (cancion){
          Mix_HaltMusic();
          Mix_FreeMusic(cancion);
     }

     Mix_CloseAudio();
     SDL_Quit();
     exit (0);
     //return 0;
}

/*
 * WxButton2Click
 */
void ProyectoTesisDlg::WxButton2Click(wxCommandEvent& event)
{
      copiaImage(surfaceGeneral);
}
```

```
/*
 * WxButton3Click
 */
void ProyectoTesisDlg::WxButton3Click(wxCommandEvent& event)
{
	exit (0);
}

/*
 * WxButton4Click
 */
void ProyectoTesisDlg::WxButton4Click(wxCommandEvent& event)
{
	Mix_PlayMusic(cancion, -1);
}

/*
 * WxButton5Click
 */
void ProyectoTesisDlg::WxButton5Click(wxCommandEvent& event)
{
	Mix_HaltMusic();
}

/*
 * WxButton6Click
 */
void ProyectoTesisDlg::WxButton6Click(wxCommandEvent& event)
{
	morfologia(surfaceGeneral);
}

/*
 * WxButton7Click
 */
void ProyectoTesisDlg::WxButton7Click(wxCommandEvent& event)
{
	umbralizacion(surfaceGeneral, 60);
}

/*
 * WxButton8Click
 */
void ProyectoTesisDlg::WxButton8Click(wxCommandEvent& event)
{
	umbralizacion(surfaceGeneral, 130);
}

/*
 * WxButton9Click
 */
void ProyectoTesisDlg::WxButton9Click(wxCommandEvent& event)
{
	morfoClean(surfaceGeneral);
}

/*
 * WxButton10Click
 */
void ProyectoTesisDlg::WxButton10Click(wxCommandEvent& event)
{
	creaPixel(surfaceGeneral);
}

/*
 * WxButton11Click
 */
void ProyectoTesisDlg::WxButton11Click(wxCommandEvent& event)
{
	escituraDatos(surfaceGeneral);
}

/*
 * WxButton12Click
 */
void ProyectoTesisDlg::WxButton12Click(wxCommandEvent& event)
```

```
{
	limpiarImg(surfaceGeneral);
}

/*
 * WxButton13Click
 */
void ProyectoTesisDlg::WxButton13Click(wxCommandEvent& event)
{
	aGris(surfaceGeneral);
}

/*
 * WxButton14Click
 */
void ProyectoTesisDlg::WxButton14Click(wxCommandEvent& event)
{
	limiteInterno(surfaceGeneral);
}

/*
 * WxButton15Click
 */
void ProyectoTesisDlg::WxButton15Click(wxCommandEvent& event)
{
	BorraLimInterno(surfaceGeneral);
}

/*
 * WxButton16Click
 */
void ProyectoTesisDlg::WxButton16Click(wxCommandEvent& event)
{
	limiteExterno(surfaceGeneral);
}

/*
 * WxButton17Click
 */
void ProyectoTesisDlg::WxButton17Click(wxCommandEvent& event)
{
	coordPolares(surfaceGeneral);
}

/*
 * WxButton18Click
 */
void ProyectoTesisDlg::WxButton18Click(wxCommandEvent& event)
{
	extraeCodigo(surfaceGeneral, 1);
}

/*
 * WxButton19Click
 */
void ProyectoTesisDlg::WxButton19Click(wxCommandEvent& event)
{
	extraeCodigo(surfaceGeneral, 2);
}

/*
 * WxButton20Click
 */
void ProyectoTesisDlg::WxButton20Click(wxCommandEvent& event)
{
	extraeCodigo(surfaceGeneral, 3);
}

/*
 * WxButton21Click
 */
void ProyectoTesisDlg::WxButton21Click(wxCommandEvent& event)
{
	extraeCodigo(surfaceGeneral, 4);
}
```

```
/*
 * WxButton22Click
 */
void ProyectoTesisDlg::WxButton22Click(wxCommandEvent& event)
{
	extraeCodigo(surfaceGeneral, 5);
}

/*
 * WxButton23Click
 */
void ProyectoTesisDlg::WxButton23Click(wxCommandEvent& event)
{
	resize(surfaceGeneral);
}

/*
 * Función para que el fondo siempre se mantenga
 *
 * @ a Formato de un pixel que será usado para darle color al fondo (screen)
 * @ screen Es el fondo de nuestra pantalla
 *
 */

void limpiar(SDL_Surface *screen){
     Uint32 a = SDL_MapRGB(screen->format, 55, 61, 117);
     SDL_FillRect(screen, NULL, a);
}

/*
 * Función para que el fondo siempre se mantenga
 *
 * @ otroPixel Formato de un pixel que será usado para darle color al fondo
(imagen)
 * @ surface Es el fondo de nuestra imagen
 *
 */

void limpiarImg(SDL_Surface *surface){
     Uint32 otroPixel = SDL_MapRGB(surface->format, 255, 255, 255);
     SDL_FillRect(surface, NULL, otroPixel);
     surfaceGeneral = surface;
}

/*
 * Con esta función imprimimos nuestra imagen sobre el fondo
 *
 * @ screen Es el fondo de nuestra pantalla
 * @ ima Es la imagen con la que trabajamos
 * @ x, y Son la posición en la que se desea que aparesca la imagen dentro de
screen
 *
 */

void imprimir(SDL_Surface *screen, SDL_Surface *ima, int x, int y){
     SDL_Rect rect = {x, y, 0, 0};
     SDL_BlitSurface(ima, 0, screen, &rect);
}

/*
 * Esta función es la que permite mover una imagen, además de darnos un evento
para notar el cambio de píxel
 *
 * @ key Es un apintador que nos hará referencia a la tecla presionada
 * @ x, y Son apuntadores a la posición de la imagen dentro de screen
 * @ ancho, alto Son las medidas de nuestra imagen
 * @ surface Recibe a nuestra imagen actual para modificarla
 * @ cancion Contiene la melodia que intridujimos
 *
 */

void mover_objeto(int *x, int *y, int ancho, int alto, SDL_Surface *surface,
Mix_Music *cancion){
     Uint8 *key = SDL_GetKeyState(NULL);
```

```
//  Estas  condiciones  nos  permiten  desplazar  la  imagen  mediante  las
direccionales del teclado
     if(key[SDLK_LEFT] && * x > 0){
          (*x) -= 5;
     }

     if(key[SDLK_RIGHT] && (*x + ancho) < 640){
          (*x) += 5;
     }

     if(key[SDLK_UP] && *y > 0){
          (*y) -= 5;
     }

     if(key[SDLK_DOWN] && (*y + alto) < 480){
          (*y) += 5;
     }

     // Con esta tecla terminamos el programa
     if(key[SDLK_ESCAPE]){
          if (cancion){
               Mix_HaltMusic();
               Mix_FreeMusic(cancion);
          }

          Mix_CloseAudio();
          SDL_Quit();
          exit (0);
     }

     // Este par de teclas nos permiten tocar o parar el midi
     if(key[SDLK_HOME]){
          Mix_PlayMusic(cancion, -1);
     }

     if(key[SDLK_END]){
          Mix_HaltMusic();
     }

     // De aquí en adelante las teclas activan funciones para trabajar la
imagen
     if(key[SDLK_RETURN]){
          umbralizacion(surface, 60);
     }

     if(key[SDLK_BACKSPACE]){
          umbralizacion(surface, 130);
     }

     if(key[SDLK_KP_ENTER]){
          morfoClean(surface);
     }

     if(key[SDLK_F2]){
          morfologia(surface);
     }

     if(key[SDLK_SPACE]){
          creaPixel(surface);
     }

     if(key[SDLK_F1]){
          escituraDatos(surface);
     }

     if(key[SDLK_F4]){
          limpiarImg(surface);
     }

     if(key[SDLK_F5]){
          copiaImage(surface);
     }

     if(key[SDLK_DELETE]){
          aGris(surface);
     }
```

```
     if(key[SDLK_PAGEDOWN]){
          limiteInterno(surface);
     }

     if(key[SDLK_PAGEUP]){
          BorraLimInterno(surface);
     }

     if(key[SDLK_KP_PLUS]){
          limiteExterno(surface);
     }

     if(key[SDLK_KP_MULTIPLY]){
          coordPolares(surface);
     }

     if(key[SDLK_KP_DIVIDE]){
          resize(surface);
     }

     if(key[SDLK_F8]){
          extraeCodigo(surface, 1);
     }

     if(key[SDLK_F9]){
          extraeCodigo(surface, 2);
     }

     if(key[SDLK_F10]){
          extraeCodigo(surface, 3);
     }

     if(key[SDLK_F11]){
          extraeCodigo(surface, 4);
     }

     if(key[SDLK_F12]){
          extraeCodigo(surface, 5);
     }
}

/*
 * Esta función tiene como objetivo incertar un pixel color verde si se quiere
averiguar donde es el centro de la
 * pupila
 *
 * @ surface Reprecentará nuestra imagen actual
 * @ otroPixel Define un píxel nuevo que será incertado
 * @ centroH, centroV Nos indican el centro de la pupila
 *
 */

void creaPixel(SDL_Surface *surface){
     int centroH, centroV;
     Uint32 otroPixel;
     SDL_LockSurface(surface);      // Con esto podemos atrapar un píxel para
manipularlo

     // Definimos un nuevo píxel (color) para incertarlo en la posición
     otroPixel = SDL_MapRGB(surface->format, 0, 255, 0);

     //Mediante estas dos funciones obtenemos el centro de la pupila
     centroV = lineaV(surface);
     centroH = lineaH(surface, 0);

     // Incertamos el píxel en la posición dada
     putpixel(surface, centroH, centroV, otroPixel);
     SDL_UnlockSurface(surface);      // Lo que nos permite liberar el píxel
que atrapamos
     surfaceGeneral = surface;
}
```

```
/*
 * Esta función nos permite mostrar el contenido de nuestra imagen como lo es
tamaño, colores en los píxeles
 *
 * @ surface Es nuestra imagen actual
 * @ i, j Para las coordenadas de los píxeles "x" corresponde a "j" y "y"
correspondiente a "i"
 * @ otroPixel Define un píxel nuevo que será incertado
 * @ tempR, tempB, tempG Contienen el mismo píxel que se descompone en los
diferentes colores
 * @ red, green, blue contienen el grado del color (RGB)
 *
 */

void escituraDatos(SDL_Surface *surface){
     int i, j;
     Uint32 tempR, tempB, tempG, otroPixel;
     Uint8 red, blue, green;

     // Existen muchas características que podemos imprimir de la surface,
pero estas dos son las más importantes
     // en este momento
     printf("PRUEBA FINAL SOBRE PIXELES\n\n");
     printf("Prueba height %d\n", surface->h);       // Nos permite saber el
alto de la imagen
     printf("Prueba width %d\n\n", surface->w);     // Nos permite conocer el
ancho de la imagen

     SDL_LockSurface(surface);      // Con esto podemos atrapar un píxel para
manipularlo

     // Con este for realizaremos las pruebas que necesitamos para provar las
nuevas funciones
     for(i = 0; i < surface->h; i++){
          for(j = 0; j < surface->w; j++){
               if(j == 0){
                    printf("ÉSTE ES EL RENGLÓN %d\n\n", i);
               }

               // Atrapamos el píxel en la posición indicada (al parecer
empieza en 0,0 y temina en i-1 y j-1)
               otroPixel = getpixel(surface, j, i);
               printf("Prueba otroPixel [%d] %d\n", j, otroPixel);

               // Descomponemos el pìxel para saber los colores por los que
está compuesto (no es necesario un orden)
               // El primer color que extraemos es el rojo
               tempR = otroPixel&surface->format->Rmask;
               tempR = tempR>>surface->format->Rshift;
               tempR = tempR<<surface->format->Rloss;
               red = (Uint8)tempR;
               printf("Prueba red   %d\n", red);

               // El siguiente que tenemos es el verde
               tempG = otroPixel&surface->format->Gmask;
               tempG = tempG>>surface->format->Gshift;
               tempG = tempG<<surface->format->Gloss;
               green =(Uint8)tempG;
               printf("Prueba green %d\n", green);

               // El último que tenemos es azul
               tempB = otroPixel&surface->format->Bmask;
               tempB = tempB>>surface->format->Bshift;
               tempB = tempB<<surface->format->Bloss;
               blue = (Uint8)tempB;
               printf("Prueba blue  %d\n\n", blue);

               // Definimos un nuevo píxel (color) para incertarlo en la
posición
               otroPixel = SDL_MapRGB(surface->format, 0, 255, 0);
          }
     }

     SDL_UnlockSurface(surface);      // Lo que nos permite liberar el píxel
que atrapamos
     surfaceGeneral = surface;
}
```

```
/*
 *  Esta función es la que nos trae el píxel que queremos, para ello solo
necesitamos indicarle la posición en
 * que se encuentra
 *
 * @ surface Es nuestra imagen actual
 * @ x, y Es la posición del píxel dentro de nuestra imagen
 * @ bpp Son los bytes que tiene dado píxel segun la imagen
 *
 */

Uint32 getpixel(SDL_Surface *surface, int x, int y){
     int bpp = surface->format->BytesPerPixel;

     /* Aquí p es la dirección del píxel que queremos traer */
     Uint8 *p = (Uint8 *)surface->pixels + y * surface->pitch + x * bpp;

     switch(bpp) {
          case 1:
               return *p;

          case 2:
               return *(Uint16 *)p;

          case 3:
               if(SDL_BYTEORDER == SDL_BIG_ENDIAN)
                    return p[0] << 16 | p[1] << 8 | p[2];
               else
                    return p[0] | p[1] << 8 | p[2] << 16;

          case 4:
               return *(Uint32 *)p;

          default:
               return  0;         /*  no  debería  pasar,  pero  nos  quita  los
"warnings" */
     }
}

/*
 * Con esta función incertamos el nuevo píxel en la posición deseada
 *
 * @ surface Es el espacio donde se incerta el píxel
 * @ x, y La posición dentro de la imagen
 * @ pixel Es el nuevo píxel que se incertará
 * @ bpp es la cantidad de bytes por píxel que tiene la imagen
 *
 */

void putpixel(SDL_Surface *surface, int x, int y, Uint32 pixel){
     int bpp = surface->format->BytesPerPixel;

     /* Aquí p es la dirección del píxel que queremos incertar */
     Uint8 *p = (Uint8 *)surface->pixels + y * surface->pitch + x * bpp;

     switch(bpp) {
          case 1:
               *p = pixel;
          break;

          case 2:
               *(Uint16 *)p = pixel;
          break;

          case 3:
               if(SDL_BYTEORDER == SDL_BIG_ENDIAN) {
                    p[0] = (pixel >> 16) & 0xff;
                    p[1] = (pixel >> 8) & 0xff;
                    p[2] = pixel & 0xff;
               } else {
                    p[0] = pixel & 0xff;
                    p[1] = (pixel >> 8) & 0xff;
                    p[2] = (pixel >> 16) & 0xff;
               }
          break;

          case 4:
```

```
                *(Uint32 *)p = pixel;
            break;
        }
}

/*
 * Esta función nos permite realizar la umbralización del área de la pupila
 *
 * @ surface Es nuestra imagenactual
 * @ i, j Para las coordenadas de los píxeles "x" corresponde a "j" y "y"
corresponde a "i"
 * @ t Es usado como un margen para visualizar solo las partes más oscuras o
claras de la imagen
 * @ otroPixel Nos permite definir un nuevo píxel para trabajar
 * @ tempR Nos ayuda a extraer el color rojo de un píxel
 * @ red Nos da la cantidad de rojo que tiene el píxel
 *
 */

void umbralizacion(SDL_Surface *surface, int t){
     int i, j;
     Uint32 tempR, otroPixel;
     Uint8 red;

     SDL_LockSurface(surface);

     for(i = 0; i < surface->h; i++){
          for(j = 0; j < surface->w; j++){
               // Asumimos que es una imagen en escala de grices, podemos usar
cualquier color, en este caso rojo
               otroPixel = getpixel(surface, j, i);
               tempR = otroPixel&surface->format->Rmask;
               tempR = tempR>>surface->format->Rshift;
               tempR = tempR<<surface->format->Rloss;
               red = (Uint8)tempR;

               if(red > t){
                    otroPixel = SDL_MapRGB(surface->format, 255, 255, 255);
                    putpixel(surface, j, i, otroPixel);
               } else {
                    otroPixel = SDL_MapRGB(surface->format, 0, 0, 0);
                    putpixel(surface, j, i, otroPixel);
               }
          }
     }

     SDL_UnlockSurface(surface);
     surfaceGeneral = surface;
}

/*
 * Esta función contendrá todas las operaciones morfológicas que se requieran
 *
 * @ surface Es nuestra imagenactual
 * @ i Indices para realizar el ciclo
 * @ surface Es nuestra imagen
 *
 */

void morfologia(SDL_Surface *surface){
     int i;
     //aGris(surface);
     umbralizacion(surface, 60);
     for(i = 0; i < 15; i++){
          morfoClean(surface);
     }
     limiteExterno(surface);
     coordPolares(surface);
     resize(surface);
     extraeCodigo(surface, 5);
}
```

```
/*
 * Esta función recrea la función de dilatación
 *
 * @ surface Es nuestra imagen actual
 * @ k Permite recorrer el arreglo de vecinos
 * @ i, j Para las coordenadas de los píxeles "x" corresponde a "j" y "y"
corresponde a "i"
 * @ vecBlk Nos indica el número de vecinos de color negro
 * @ tempR, tempV Nos permiten capturar el color rojo (para ambos) de un píxel
 * @ otroPixel Es un píxel nuevo con el que trabajaremos
 * @ red, vColor Son la intencidad de rojo del píxel
 * @ vecino Es un arreglo de 8 componentes que nos ayuda averificar los
píxeles al rededor del actual
 *
 */

void morfoDilation(SDL_Surface *surface){
     int i, j, k, vecBlk;
     Uint32 tempR, tempV, otroPixel;
     Uint32 vecino[8];
     Uint8 red, vColor;

     SDL_LockSurface(surface);

     for(i = 0; i < surface->h; i++){
          for(j = 0; j < surface->w; j++){

               // Asumimos que es una imagen en escala de grices, podemos usar
cualquier color, en este caso rojo
               otroPixel = getpixel(surface, j, i);
               tempR = otroPixel&surface->format->Rmask;
               tempR = tempR>>surface->format->Rshift;
               tempR = tempR<<surface->format->Rloss;
               red = (Uint8)tempR;

               vecino[0] = getpixel(surface, j - 1, i - 1);
               vecino[1] = getpixel(surface, j, i - 1);
               vecino[2] = getpixel(surface, j + 1, i - 1);
               vecino[3] = getpixel(surface, j - 1, i);
               vecino[4] = getpixel(surface, j + 1, i);
               vecino[5] = getpixel(surface, j - 1, i + 1);
               vecino[6] = getpixel(surface, j, i + 1);
               vecino[7] = getpixel(surface, j + 1, i + 1);

               vecBlk = 0;
               if(i != 0 && i != 479 && j!= 0 && j != 640){
                   for(k = 0; k < 8; k++){
                        tempV = vecino[k]&surface->format->Rmask;
                        tempV = tempV>>surface->format->Rshift;
                        tempV = tempV<<surface->format->Rloss;
                        vColor = (Uint8)tempV;
                        if(vColor == 0){
                             vecBlk++;
                        }
                   }
                   if(vecBlk >= 4){
                        otroPixel = SDL_MapRGB(surface->format, 0, 0, 0);
                        putpixel(surface, j, i, otroPixel);
                   }
               }
          }
     }

     SDL_UnlockSurface(surface);
}
```

```
/*
 * Esta función trata de representar que es lo que sucede cuando se aplica la
erosión
 *
 * @ surface Es nuestra imagen actual
 * @ k Permite recorrer el arreglo de vecinos
 * @ i, j Para las coordenadas de los píxeles "x" corresponde a "j" y "y"
corresponde a "i"
 * @ vecWht Nos indica el número de vecinos de color blanco
 * @ tempR, tempV Nos permiten capturar el color rojo (para ambos) de un píxel
 * @ otroPixel Es un píxel nuevo con el que trabajaremos
 * @ red, vColor Son la intencidad de rojo del píxel
 * @ vecino Es un arreglo de 8 componentes que nos ayuda averificar los
píxeles al rededor del actual
 *
 */

void morfoErosion(SDL_Surface *surface, int i, int j){

     // Para las coordenadas de los píxeles x corresponde a j y y a i
     int k, vecWht;
     Uint32 tempR, tempV, otroPixel;
     Uint32 vecino[8];
     Uint8 red, vColor;

     SDL_LockSurface(surface);

     //for(i = 0; i < surface->h; i++){
          //for(j = 0; j < surface->w; j++){

               // Asumimos que es una imagen en escala de grices, podemos usar
cualquier color, en este caso rojo
               otroPixel = getpixel(surface, j, i);
               tempR = otroPixel&surface->format->Rmask;
               tempR = tempR>>surface->format->Rshift;
               tempR = tempR<<surface->format->Rloss;
               red = (Uint8)tempR;

               vecino[0] = getpixel(surface, j - 1, i - 1);
               vecino[1] = getpixel(surface, j, i - 1);
               vecino[2] = getpixel(surface, j + 1, i - 1);
               vecino[3] = getpixel(surface, j - 1, i);
               vecino[4] = getpixel(surface, j + 1, i);
               vecino[5] = getpixel(surface, j - 1, i + 1);
               vecino[6] = getpixel(surface, j, i + 1);
               vecino[7] = getpixel(surface, j + 1, i + 1);

               vecWht = 0;
               if(i != 0 && i != 479 && j!= 0 && j != 640){
                   for(k = 0; k < 8; k++){
                        tempV = vecino[k]&surface->format->Rmask;
                        tempV = tempV>>surface->format->Rshift;
                        tempV = tempV<<surface->format->Rloss;
                        vColor = (Uint8)tempV;
                        if(vColor == 255){
                             vecWht++;
                        }
                   }
                   if(vecWht >= 3){
                        otroPixel  =  SDL_MapRGB(surface->format,  255,  255,
255);
                        putpixel(surface, j, i, otroPixel);
                   }
               }
          //}
     //}

     SDL_UnlockSurface(surface);
}
```

```
/*
 * Con esta función removemos todas las impuresas de nuestra figura, es lo más
parecido a las funciones de cerradura
 * y apertura
 *
 * @ surface Es nuestra imagen actual
 * @ k Permite recorrer el arreglo de vecinos
 * @ i, j Para las coordenadas de los píxeles "x" corresponde a "j" y "y"
corresponde a "i"
 * @ vecBlk Nos indica el número de vecinos de color negro
 * @ vecWht Nos indica el número de vecinos de color blanco
 * @ tempR, tempV Nos permiten capturar el color rojo (para ambos) de un píxel
 * @ otroPixel Es un píxel nuevo con el que trabajaremos
 * @ red, vColor Son la intencidad de rojo del píxel
 * @ vecino Es un arreglo de 24 componentes que nos ayuda averificar los
píxeles al rededor del actual
 *
 */

void morfoClean(SDL_Surface *surface){

     // Para las coordenadas de los píxeles x corresponde a j y y a i
     int i, j, k, vecBlk, vecWht;
     Uint32 tempR, tempV, otroPixel;
     Uint32 vecino[24];
     Uint8 red, vColor;

     SDL_LockSurface(surface);

     for(i = 0; i < surface->h; i++){
          for(j = 0; j < surface->w; j++){

               // Asumimos que es una imagen en escala de grices, podemos usar
cualquier color, en este caso rojo
               otroPixel = getpixel(surface, j, i);
               tempR = otroPixel&surface->format->Rmask;
               tempR = tempR>>surface->format->Rshift;
               tempR = tempR<<surface->format->Rloss;
               red = (Uint8)tempR;

               vecino[0] = getpixel(surface, j - 1, i - 1);
               vecino[1] = getpixel(surface, j, i - 1);
               vecino[2] = getpixel(surface, j + 1, i - 1);
               vecino[3] = getpixel(surface, j - 1, i);
               vecino[4] = getpixel(surface, j + 1, i);
               vecino[5] = getpixel(surface, j - 1, i + 1);
               vecino[6] = getpixel(surface, j, i + 1);
               vecino[7] = getpixel(surface, j + 1, i + 1);
               vecino[8] = getpixel(surface, j - 2, i - 2);
               vecino[9] = getpixel(surface, j - 1, i - 2);
               vecino[10] = getpixel(surface, j, i - 2);
               vecino[11] = getpixel(surface, j + 1, i - 2);
               vecino[12] = getpixel(surface, j + 2, i - 2);
               vecino[13] = getpixel(surface, j - 2, i - 1);
               vecino[14] = getpixel(surface, j + 2, i - 1);
               vecino[15] = getpixel(surface, j - 2, i);
               vecino[16] = getpixel(surface, j + 2, i);
               vecino[17] = getpixel(surface, j - 2, i + 1);
               vecino[18] = getpixel(surface, j + 2, i + 1);
               vecino[19] = getpixel(surface, j - 2, i + 2);
               vecino[20] = getpixel(surface, j - 1, i + 2);
               vecino[21] = getpixel(surface, j, i + 2);
               vecino[22] = getpixel(surface, j + 1, i + 2);
               vecino[23] = getpixel(surface, j + 2, i + 2);

               vecBlk = 0;
               vecWht = 0;
               if(i != 0 && i != 479 && j!= 0 && j != 640){
                   for(k = 0; k < 24; k++){
                        tempV = vecino[k]&surface->format->Rmask;
                        tempV = tempV>>surface->format->Rshift;
                        tempV = tempV<<surface->format->Rloss;
                        vColor = (Uint8)tempV;
                        if(vColor == 255){
                             vecWht++;
                        } else if(vColor == 0){
                             vecBlk++;
```

```
                    }
                }

                /*
                 * vecWht y vecBlk son usados para saber cuantos píxeles
vecinos tienen color blanco y negro,
                 * en base a eso decidir que color tomara nuestro píxel
                 */
                if(vecWht != 12 && vecBlk != 12){
                    if(vecWht > vecBlk && red != 255){
                        otroPixel = SDL_MapRGB(surface->format, 255, 255,
255);     //Erosión
                        putpixel(surface, j, i, otroPixel);
                    } else if(vecBlk > vecWht && red != 0){
                        // Dilatación con mascara de 3 x 3
                        otroPixel = SDL_MapRGB(surface->format, 0, 0, 0);
                        putpixel(surface, j - 1, i - 1, otroPixel);
                        putpixel(surface, j, i - 1, otroPixel);
                        putpixel(surface, j + 1, i - 1, otroPixel);
                        putpixel(surface, j - 1, i, otroPixel);
                        putpixel(surface, j, i, otroPixel);
                        putpixel(surface, j + 1, i, otroPixel);
                        putpixel(surface, j - 1, i + 1, otroPixel);
                        putpixel(surface, j, i + 1, otroPixel);
                        putpixel(surface, j + 1, i + 1, otroPixel);
                    }
                }
            }
        }
    }

    SDL_UnlockSurface(surface);
}

/*
 *  Esta función nos permite saber la línea vertical con mayor número de
píxeles negros dentro de la pupila
 *
 * @ surface Es nuestra imagen
 * @ x, y Es la posición de un píxel dentro de nuestra imagen
 * @ otroPixel Define un píxel nuevo que será incertado
 *  @ tempR Contienen el mismo píxel que se descompone en los diferentes
colores
 * @ continih, contfinh Son variables temporales que nos irán guardando la
línea con mayor número de píxeles
 * @ Ih, tmpIh contienen el grado del color (RGB)
 *  @  x0,  y0,  yf,  x1h,  y1h,  y2h Son variables temporales durante nuestro
análisis
 * @ ycentro Nos da el centro con respecto a y
 * @ xinc, xigu Nos permiten realizar comparaciones
 *
 */

    int lineaV(SDL_Surface *surface){
    int x, xinc, y, ycentro = 0;
    int continih = 0, x1h = 0, y1h = 0, y2h = 0;
    int xigu = 0, x0 = 0, y0 = 0, yf = 0, contfinh = 0;
    Uint32 tempR, otroPixel;
    Uint8 Ih, tmpIh;

    SDL_LockSurface(surface);

    for(x = 0; x < surface->w; x++){
        xinc = 0;
        contfinh = 0;
        for (y = 0; y < surface->h; y++){
            otroPixel = getpixel(surface, x, y);
            tempR = otroPixel&surface->format->Rmask;
            tempR = tempR>>surface->format->Rshift;
            tempR = tempR<<surface->format->Rloss;
            Ih = (Uint8)tempR;

            otroPixel = getpixel(surface, x, y - 1);
            tempR = otroPixel&surface->format->Rmask;
            tempR = tempR>>surface->format->Rshift;
            tempR = tempR<<surface->format->Rloss;
            tmpIh = (Uint8)tempR;
```

```
                if (Ih == 0){
                     xinc++;
                     contfinh = xinc;
                }
                if (xinc == 1){
                     x0 = x;
                     y0 = y;
                } else if (x > 0 && y > 0 && Ih == 255 && tmpIh == 0){
                     yf = y - 1;
                }
           }
           if (contfinh > continih && contfinh > 0){
                xigu = 0;
           }
           if (contfinh >= continih && contfinh > 0){
               if (contfinh == continih ){
                    xigu++;
               }
               continih = contfinh;
               x1h = x0;
               y1h = y0;
               y2h = yf;
           }
     }

     ycentro = round((y1h + y2h) / 2);
     SDL_UnlockSurface(surface);
     return ycentro;
}

/*
 *  Esta función nos permite saber la línea horizontal con mayor número de
píxeles negros dentro de la pupila
 *
 * @ surface Es nuestra imagen
 * @ x, y Es la posición de un píxel dentro de nuestra imagen
 * @ centroRadio Es una bandera que nos indica si nuestro análisis requiere el
centro o el rádio
 * @ otroPixel Define un píxel nuevo que será incertado
 *  @ tempR Contienen el mismo píxel que se descompone en los diferentes
colores
 * @ continiv, contfinv Son variables temporales que nos irán guardando la
línea con mayor número de píxeles
 * @ Ih, tmpIh contienen el grado del color (RGB)
 *  @ x0, y0, xf, x1v, y1v, x2v Son variables temporales durante nuestro
análisis
 * @ xcentro Nos da el centro con respecto a x
 * @ yinc, yigu Nos permite realizar comparaciones
 *
 */

int lineaH(SDL_Surface *surface, int centroRadio){
     int x, yinc, y, xcentro = 0;
     int continiv = 0, x1v = 0, y1v = 0, x2v = 0;
     int yigu = 0, x0 = 0, y0 = 0, xf = 0, contfinv = 0;
     Uint32 tempR, otroPixel;
     Uint8 Ih, tmpIh;

     SDL_LockSurface(surface);

     // Ralizamos la búsqueda de la línea horizontal con mayor número de
píxeles
     for(x = 0; x < surface->w; x++){
          yinc = 0;
          contfinv = 0;
          for (y = 0; y < surface->h; y++){
               otroPixel = getpixel(surface, x, y);
               tempR = otroPixel&surface->format->Rmask;
               tempR = tempR>>surface->format->Rshift;
               tempR = tempR<<surface->format->Rloss;
               Ih = (Uint8)tempR;

               otroPixel = getpixel(surface, x, y - 1);
               tempR = otroPixel&surface->format->Rmask;
               tempR = tempR>>surface->format->Rshift;
               tempR = tempR<<surface->format->Rloss;
```

```
                tmpIh = (Uint8)tempR;

                if (Ih == 0){
                     yinc++;
                     contfinv = yinc;
                }
                if (yinc == 1){
                     x0 = x;
                     y0 = y;
                } else if (x > 0 && y > 0 && Ih == 255 && tmpIh == 0){
                     xf = x - 1;
                }
           }
           if (contfinv > continiv && contfinv > 0){
                yigu = 0;
           }
           if (contfinv >= continiv && contfinv > 0){
               if (contfinv == continiv ){
                    yigu++;
               }
               continiv = contfinv;
               y1v = y0;
               x1v = x0;
               x2v = xf;
           }
     }

     // Calculamos el centro
     xcentro = round((x1v + x2v) / 2);

     SDL_UnlockSurface(surface);

     // Regresamos el resultado del análisis según la bandera
     if(centroRadio == 0){
          return xcentro;
     } else if(centroRadio == 1){
          return continiv;
     }
}

/*
 * Esta función nos permite transformar una imagen a color en una a escala de
grises
 *
 * @ surface Es nuestra imagen actual
 * @ i, j Para las coordenadas de los píxeles "x" corresponde a "j" y "y"
corresponde a "i"
 * @ tmpColor Es el grado de color que adquirira nuestro píxel
 * @ otroPixel Define un píxel nuevo que será incertado
 * @ tempR, tempB, tempG Contienen el mismo píxel que se descompone en los
diferentes colores
 * @ red, green, blue contienen el grado del color (RGB)
 *
 */

void aGris(SDL_Surface *surface){
     int i, j, tmpColor;
     Uint32 tempR, tempB, tempG, otroPixel;
     Uint8 red, blue, green;

     SDL_LockSurface(surface);

     // Revisamos todos los píxeles para realizar la transformación
     for(i = 0; i < surface->h; i++){
          for(j = 0; j < surface->w; j++){

               // Realizamos la extraccion de los colores
               otroPixel = getpixel(surface, j, i);
               tempR = otroPixel&surface->format->Rmask;
               tempR = tempR>>surface->format->Rshift;
               tempR = tempR<<surface->format->Rloss;
               red = (Uint8)tempR;
               tempG = otroPixel&surface->format->Gmask;
               tempG = tempG>>surface->format->Gshift;
               tempG = tempG<<surface->format->Gloss;
               green =(Uint8)tempG;
               tempB = otroPixel&surface->format->Bmask;
```

```
                tempB = tempB>>surface->format->Bshift;
                tempB = tempB<<surface->format->Bloss;
                blue = (Uint8)tempB;

                // Buscamos cual de ellos es el más oscuro para que los demás
tengan el mismo tono
                if(red > green && red > blue){
                     tmpColor = red;
                     otroPixel    =    SDL_MapRGB(surface->format,    tmpColor,
tmpColor, tmpColor);
                     putpixel(surface, j, i, otroPixel);
                }

                if(green > red && green > blue){
                     tmpColor = green;
                     otroPixel    =    SDL_MapRGB(surface->format,    tmpColor,
tmpColor, tmpColor);
                     putpixel(surface, j, i, otroPixel);
                }

                if(blue > green && red < blue){
                     tmpColor = blue;
                     otroPixel    =    SDL_MapRGB(surface->format,    tmpColor,
tmpColor, tmpColor);
                     putpixel(surface, j, i, otroPixel);
                }
           }
     }

     SDL_UnlockSurface(surface);
     surfaceGeneral = surface;

}

/*
 * Nos permite esta función recargar nuestra imagen
 *
 * @ surface Es nuestra imagen actual
 * @ i, j Para las coordenadas de los píxeles "x" corresponde a "j" y "y"
corresponde a "i"
 * @ otroPixel Define un píxel nuevo que será incertado
 * @ image Es la imagen que se recargará
 *
 */

void copiaImage(SDL_Surface *surface){
     int i, j;
     SDL_Surface *image;
     Uint32 otroPixel;

     SDL_LockSurface(surface);

     // Nos indica cual es la imágen que vamos a copiar
     image = IMG_Load(files[5]);

     // Limpiamos toda el área para tener todo en blanco
     limpiarImg(surface);

     // Realizamos la copia píxel por píxel
     for(i = 0; i < image->h; i++){
          for(j = 0; j < image->w; j++){
               otroPixel = getpixel(image, j, i);
               putpixel(surface, j, i, otroPixel);
          }
     }

     SDL_UnlockSurface(surface);
     surfaceGeneral = surface;
}
```

```
/*
 * Con esta función podemos obtener el límite interno del iris
 *
 * @ surface Es nuestra imagen
 * @ x, y Es la posición de un píxel dentro de nuestra imagen
 * @ x2, y2 Calcula una distancia para el píxel con respecto al centro de la
pupila
 * @ xcentro, ycentro Nos dan el centro de la pupila
 * @ x2y2 Nos da un nuevo centro con respecto al píxel actual
 * @ radio Es el radio de la pupila
 * @ radio2 Es el radio al cuadrado
 * @ otroPixel Es un nuevo píxel con el que trabajaremos
 *
 */

void limiteInterno(SDL_Surface *surface){
     int x, y, x2, y2, xcentro, ycentro, x2y2, radio, radio2;
     Uint32 otroPixel;

     SDL_LockSurface(surface);

     // Obtenemos el radio a partir de los centros que encontramos anterior
mente
     radio = (lineaH(surface, 1)) / 2;
     tmpRadio = radio;
     radio2 = radio * radio;
     xcentro = lineaH(surface, 0);
     ycentro = lineaV(surface);
     copiaImage(surface);
     aGris(surface);

     // Buscamos entre todos los píxeles y ponemos en negro los estan dentro
del rádio
     for(x = 0; x < surface->w; x++){
          for (y = 0; y < surface->h; y++){
               x2 = (x - xcentro) * (x - xcentro);
               y2 = (y - ycentro) * (y - ycentro);
               x2y2 = x2 + y2;
               if(x2y2 <= radio2){
                    otroPixel = SDL_MapRGB(surface->format, 0, 0, 0);
                    putpixel(surface, x, y, otroPixel);
               }
          }
     }

     SDL_UnlockSurface(surface);
     surfaceGeneral = surface;
}

/*
 * Esta función nos permite extraer la pupila y marcar el límite interno
 *
 * @ surface Es nuestra imagen actual
 * @ i, j Para las coordenadas de los píxeles "x" corresponde a "j" y "y"
corresponde a "i"
 * @ otroPixel Es un nuevo píxel con el que trabajaremos
 * @ tempR Contiene el mismo píxel que se descompone en los diferentes colores
(rojo en este caso)
 * @ red contienen el grado del color rojo (RGB)
 *
 */

void BorraLimInterno(SDL_Surface *surface){
     int i, j;
     Uint32 tempR, otroPixel;
     Uint8 red;

     SDL_LockSurface(surface);

     // Realizamos un barrido en toda la imagen en busca del límite
     for(i = 0; i < surface->h; i++){
          for(j = 0; j < surface->w; j++){
               otroPixel = getpixel(surface, j, i);
               tempR = otroPixel&surface->format->Rmask;
               tempR = tempR>>surface->format->Rshift;
               tempR = tempR<<surface->format->Rloss;
               red = (Uint8)tempR;
```

```
                // Verificamos que sean los grados más oscuros y los llevamos a
blanco
                if(red <= 60){
                     otroPixel = SDL_MapRGB(surface->format, 255, 255, 255);
                     putpixel(surface, j, i, otroPixel);
                }
           }
     }

     SDL_UnlockSurface(surface);
     surfaceGeneral = surface;
}

/*
 * Esta función nos permite obtener el límite externo
 *
 * @ i, j Son los índices que nos permitiran manejar nuestros arreglos
 * @ x, y Es la posición de un píxel dentro de nuestra imagen
 * @ x2, y2 Calcula una distancia para el píxel con respecto al centro que
hemos calculado
 * @ x2y2 Nos da un nuevo centro con respecto al píxel actual
 * @ radioC Es el radio que generamos a partir de las coincidencias
 * @ radioC2 Es el radio al cuadrado de las coincidencias
 * @ otroPixel Es un nuevo píxel con el que trabajaremos
 *
 */

void limiteExterno(SDL_Surface *surface){
     int i, j, x, y, x2, y2, xcentro, ycentro, x2y2;
     Uint32 otroPixel, vecino[3][3], tempV;
     Uint8 vColor;
     int ES[3][3], Imxy[3][3], IMES[3][3];
     int sumVec, coincidencias = 1, radioC, radioC2;

     xcentro = lineaH(surface, 0);
     ycentro = lineaV(surface);
     tmpXcentro = xcentro;
     tmpYcentro = ycentro;
     x = xcentro;

     SDL_LockSurface(surface);

     for(y = ycentro; y < surface->h; y++){
          vecino[0][0] = getpixel(surface, x - 1, y - 1);
          vecino[0][1] = getpixel(surface, x, y - 1);
          vecino[0][2] = getpixel(surface, x + 1, y - 1);
          vecino[1][0] = getpixel(surface, x - 1, y);
          vecino[1][1] = getpixel(surface, x, y);
          vecino[1][2] = getpixel(surface, x + 1, y);
          vecino[2][0] = getpixel(surface, x - 1, y + 1);
          vecino[2][1] = getpixel(surface, x, y + 1);
          vecino[2][2] = getpixel(surface, x + 1, y + 1);

          for (i = 0; i < 3; i++){
               for(j = 0; j < 3; j++){
                    if(i == 1 && j == 1){
                         ES[i][j] = 8;
                    } else {
                         ES[i][j] = 1;
                    }
                    tempV = vecino[i][j]&surface->format->Rmask;
                    tempV = tempV>>surface->format->Rshift;
                    tempV = tempV<<surface->format->Rloss;
                    vColor = (Uint8)tempV;
                    if(vColor == 0){
                         vColor = 1;
                    }
                    Imxy[i][j] = vColor;
                    IMES[i][j] = Imxy[i][j] * ES[i][j];
               }
          }

          sumVec  =  IMES[0][0]  +  IMES[0][1]  +  IMES[0][2]  +  IMES[1][0]  +
IMES[1][2] + IMES[2][0] + IMES[2][1] + IMES[2][2];
          if(sumVec == IMES[1][1]){
               coincidencias++;
```

```
          } else {
               break;
          }
     }

     SDL_UnlockSurface(surface);
     radioC = coincidencias * 2;
     tmpRadioC = radioC;
     radioC2 = radioC * radioC;
     limiteInterno(surface);
     BorraLimInterno(surface);

     SDL_LockSurface(surface);

     for(x = 0; x < surface->w; x++){
          for (y = 0; y < surface->h; y++){
               x2 = (x - xcentro) * (x - xcentro);
               y2 = (y - ycentro) * (y - ycentro);
               x2y2 = x2 + y2;
               if(x2y2 >= radioC2){
                    otroPixel = SDL_MapRGB(surface->format, 255, 255, 255);
                    putpixel(surface, x, y, otroPixel);
               }
          }
     }

     SDL_UnlockSurface(surface);
     surfaceGeneral = surface;
}

/*
 * Esta función realiza la transformación de coordenadas polares a cartesianas
 *
 * @ surface Es nuestra imagen actual
 * @ x, y Es la posición de un píxel dentro de nuestra imagen
 * @ x2, y2 Calcula una distancia para el píxel con respecto al centro que
hemos calculado
 * @ radio Es el radio de la pupila
 * @ radioC Es el radio que generamos a partir de las coincidencias
 * @ rTotal Es un índice que nos ayuda a buscar todo lo que se encuentra en el
rádio del íris
 * @ xcentro, ycentro Nos dan el centro de la pupila
 *  @  xpc,  ypc  Son  los  índices  para  alojar  a  las  transformaciones  de
coordenadas en una matríz
 * @ xtt, ytt Son el redondeo de las nuevas coordenadas
 * @ xt, yt Son nuestras nuevas coordenadas
 * @ teta Es el ángulo en radianes
 * @ te Es un índice
 *  @  imPolar  Es  una  matríz  de  60  x  450  que  contendrá  todas  nuestras
coordenadas que modificarán la imagen
 * @ otroPixel Es un nuevo píxel con el que trabajaremos
 * @ tempR, tempB, tempG Contienen el mismo píxel que se descompone en los
diferentes colores
 * @ red, green, blue contienen el grado del color (RGB)
 *
 */

void coordPolares(SDL_Surface *surface){
     int x, y, radio, radioC, xcentro, ycentro, rTotal;
     int xpc = 0, ypc, xtt, ytt;
     double te, teta = 0, xt, yt;
     Uint32  imPolar[60][450], tempR, tempB, tempG, otroPixel;
     Uint8 red, blue, green;

     radio = tmpRadio;
     radioC = tmpRadioC;
     xcentro = tmpXcentro;
     ycentro = tmpYcentro;

     //Exploramos toda el área en el rádio para realizar los cálculos de las
nuevas coordenadas
     for(rTotal = radioC; rTotal >= radio; rTotal--){
          xpc++;
          ypc = 0;

          for(te = 629; te >= 270;){
               teta = (M_PI * te) / 180;
```

```
                xt = xcentro + (rTotal * cos(teta));
                xtt = round(xt);
                yt = ycentro + (rTotal * sin(teta));
                ytt = round(yt);
                ypc++;
                imPolar[xpc - 1][ypc - 1] = getpixel(surface, xtt, ytt);
                te = te - 0.8;
           }
     }

     // Realizamos una limpieza del área para borrar todo aquello que no sirva
     limpiarImg(surface);

     // Pintamos ahora los píxeles con sus nuevas coordenadas contenidos en
nuestra matríz
     for(y = 0; y < 60; y++){
          for (x = 0; x < 450; x++){
                tempR = imPolar[y][x]&surface->format->Rmask;
                tempR = tempR>>surface->format->Rshift;
                tempR = tempR<<surface->format->Rloss;
                red = (Uint8)tempR;

                tempG = imPolar[y][x]&surface->format->Gmask;
                tempG = tempG>>surface->format->Gshift;
                tempG = tempG<<surface->format->Gloss;
                green =(Uint8)tempG;

                tempB = imPolar[y][x]&surface->format->Bmask;
                tempB = tempB>>surface->format->Bshift;
                tempB = tempB<<surface->format->Bloss;
                blue = (Uint8)tempB;

                otroPixel = SDL_MapRGB(surface->format, red, green, blue);
                putpixel(surface, x /*+ 80*/, y /*+ 160*/, otroPixel);
           }
     }
     surfaceGeneral = surface;
}

/*
 * Esta función nos permite realizar un recorte a la imagen hasta dejarla del
tamaño de 48 x 448
 *
 * @ surface Es nuestra imagen acutal
 * @ x, y Es la posición de un píxel dentro de nuestra imagen
 * @ tempR, tempB, tempG Contienen el mismo píxel que se descompone en los
diferentes colores
 * @ red, green, blue contienen el grado del color (RGB)
 *
 */

void resize(SDL_Surface *surface){
     int x, y;
     Uint32  tempR, tempB, tempG, otroPixel;
     Uint8 red, blue, green;

     for(x = 0; x < surface->w; x++){
           for(y = 0; y < surface->h; y++){
                if(x <= 448 /*+ 80*/ && y <= 48 /*+ 160*/){
                     otroPixel = getpixel(surface, x, y);

                     tempR = otroPixel&surface->format->Rmask;
                     tempR = tempR>>surface->format->Rshift;
                     tempR = tempR<<surface->format->Rloss;
                     red = (Uint8)tempR;

                     tempG = otroPixel&surface->format->Gmask;
                     tempG = tempG>>surface->format->Gshift;
                     tempG = tempG<<surface->format->Gloss;
                     green =(Uint8)tempG;

                     tempB = otroPixel&surface->format->Bmask;
                     tempB = tempB>>surface->format->Bshift;
                     tempB = tempB<<surface->format->Bloss;
                     blue = (Uint8)tempB;
```

```
                    otroPixel  =  SDL_MapRGB(surface->format,  red,  green,
blue);
                    putpixel(surface, x, y, otroPixel);
               }else{
                    // Blanqueamos todo lo que no esté dentro del área
                    otroPixel = SDL_MapRGB(surface->format, 255, 255, 255);
                    putpixel(surface, x, y, otroPixel);
               }
          }
     }
     surfaceGeneral = surface;
}


void extraeCodigo(SDL_Surface *surface, int imprime){
     Uint32  tempR, nuevoPixel[448][48], otroPixel[4];
     Uint8 red[4];
     int i, x1, y1, yc5 = 0;
     int xa1 = 0, xa2 = 0, xa3 = 0, xa4 = 0, xa5 = 0;
     int ya1 = 0, ya2 = 0, ya3 = 0, ya4 = 0, ya5 = 0;
     double d1i = 0, d2i = 0, d3i = 0, d4i = 0, tmpd = 0;
     double d1ic = 0, d2ic = 0, d3ic = 0, d4ic = 0, d4tmp[56][6];
     double d1cod = 0, d2cod = 0, d3cod = 0, d4cod = 0;

     // Reducimos a la mitad (1/2)
     for(x1 = 0; x1 < 448;){
          xa1++;
          ya1 =0;
          for(y1 = 0; y1 < 48;){
               otroPixel[0]= getpixel(surface, x1, y1);
               tempR = otroPixel[0]&surface->format->Rmask;
               tempR = tempR>>surface->format->Rshift;
               tempR = tempR<<surface->format->Rloss;
               red[0]= (Uint8)tempR;

               otroPixel[1]= getpixel(surface, x1, y1 + 1);
               tempR = otroPixel[1]&surface->format->Rmask;
               tempR = tempR>>surface->format->Rshift;
               tempR = tempR<<surface->format->Rloss;
               red[1]= (Uint8)tempR;

               otroPixel[2]= getpixel(surface, x1 + 1, y1);
               tempR = otroPixel[2]&surface->format->Rmask;
               tempR = tempR>>surface->format->Rshift;
               tempR = tempR<<surface->format->Rloss;
               red[2]= (Uint8)tempR;

               otroPixel[3]= getpixel(surface, x1 + 1, y1 + 1);
               tempR = otroPixel[3]&surface->format->Rmask;
               tempR = tempR>>surface->format->Rshift;
               tempR = tempR<<surface->format->Rloss;
               red[3]= (Uint8)tempR;

               if(x1 < 448 && y1 < 48){
                    d1i = (red[0] + red[1] + red[2] + red[3])/4;
                    tmpd = (red[0] - red[1] - red[2] + red[3])/4;
               }

               ya1++;
               nuevoPixel[xa1][ya1] = SDL_MapRGB(surface->format, round(d1i),
round(d1i), round(d1i));
               d1ic += tmpd;
               y1 += 2;
          }

          x1 +=2;
     }

     // De desearlo imprimimos el resultado
     if(imprime == 1 || imprime == 5){
         // Realizamos una limpieza del área para borrar todo aquello que no
sirva
          limpiarImg(surface);

          for(x1 = 0; x1 < 224; x1++){
               for(y1 = 0; y1 < 24; y1++){
                    putpixel(surface, x1, y1, nuevoPixel[x1][y1]);
```

```
                }
            }
        }

        d1cod = d1ic/5376;
        //printf("El codec es: %f\n\n", d1cod);

        // Reducimos a la mitad de la mitad (1/4)
        for(x1 = 0; x1 < 224;){
              xa2++;
              ya2 =0;
              for(y1 = 0; y1 < 24;){
                   otroPixel[0]= getpixel(surface, x1, y1);
                   tempR = otroPixel[0]&surface->format->Rmask;
                   tempR = tempR>>surface->format->Rshift;
                   tempR = tempR<<surface->format->Rloss;
                   red[0]= (Uint8)tempR;

                   otroPixel[1]= getpixel(surface, x1, y1 + 1);
                   tempR = otroPixel[1]&surface->format->Rmask;
                   tempR = tempR>>surface->format->Rshift;
                   tempR = tempR<<surface->format->Rloss;
                   red[1]= (Uint8)tempR;

                   otroPixel[2]= getpixel(surface, x1 + 1, y1);
                   tempR = otroPixel[2]&surface->format->Rmask;
                   tempR = tempR>>surface->format->Rshift;
                   tempR = tempR<<surface->format->Rloss;
                   red[2]= (Uint8)tempR;

                   otroPixel[3]= getpixel(surface, x1 + 1, y1 + 1);
                   tempR = otroPixel[3]&surface->format->Rmask;
                   tempR = tempR>>surface->format->Rshift;
                   tempR = tempR<<surface->format->Rloss;
                   red[3]= (Uint8)tempR;

                   if(x1 < 224 && y1 < 24){
                        d2i = (red[0] + red[1] + red[2] + red[3])/4;
                        tmpd = (red[0] - red[1] - red[2] + red[3])/4;
                   }

                   ya2++;
                   nuevoPixel[xa2][ya2] = SDL_MapRGB(surface->format, round(d2i),
round(d2i), round(d2i));
                   d2ic += tmpd;
                   y1 += 2;
              }

              x1 +=2;
        }

        // De desearlo imprimimos el resultado
        if(imprime == 2 || imprime == 5){
            // Realizamos una limpieza del área para borrar todo aquello que no
sirva
            limpiarImg(surface);

            for(x1 = 0; x1 < 112; x1++){
                 for(y1 = 0; y1 < 12; y1++){
                      putpixel(surface, x1, y1, nuevoPixel[x1][y1]);
                 }
            }
        }

        d2cod = d2ic/1344;
        //printf("El codec2 es: %f\n\n", d2cod);

        // Reducimos a la mitad de la mitad de la mitad (1/8)
        for(x1 = 0; x1 < 112;){
              xa3++;
              ya3 =0;
              for(y1 = 0; y1 < 12;){
                   otroPixel[0]= getpixel(surface, x1, y1);
                   tempR = otroPixel[0]&surface->format->Rmask;
                   tempR = tempR>>surface->format->Rshift;
                   tempR = tempR<<surface->format->Rloss;
                   red[0]= (Uint8)tempR;
```

```
                otroPixel[1]= getpixel(surface, x1, y1 + 1);
                tempR = otroPixel[1]&surface->format->Rmask;
                tempR = tempR>>surface->format->Rshift;
                tempR = tempR<<surface->format->Rloss;
                red[1]= (Uint8)tempR;

                otroPixel[2]= getpixel(surface, x1 + 1, y1);
                tempR = otroPixel[2]&surface->format->Rmask;
                tempR = tempR>>surface->format->Rshift;
                tempR = tempR<<surface->format->Rloss;
                red[2]= (Uint8)tempR;

                otroPixel[3]= getpixel(surface, x1 + 1, y1 + 1);
                tempR = otroPixel[3]&surface->format->Rmask;
                tempR = tempR>>surface->format->Rshift;
                tempR = tempR<<surface->format->Rloss;
                red[3]= (Uint8)tempR;

                if(x1 < 112 && y1 < 12){
                     d3i = (red[0] + red[1] + red[2] + red[3])/4;
                     tmpd = (red[0] - red[1] - red[2] + red[3])/4;
                }

                ya3++;
                nuevoPixel[xa3][ya3] = SDL_MapRGB(surface->format, round(d3i),
round(d3i), round(d3i));
                d3ic += tmpd;
                y1 += 2;
           }

           x1 +=2;
     }

     // De desearlo imprimimos el resultado
     if(imprime == 3 || imprime == 5){
         // Realizamos una limpieza del área para borrar todo aquello que no
sirva
         limpiarImg(surface);

         for(x1 = 0; x1 < 56; x1++){
              for(y1 = 0; y1 < 6; y1++){
                  putpixel(surface, x1, y1, nuevoPixel[x1][y1]);
              }
         }
     }

     d3cod = d3ic/336;
     //printf("El codec3 es: %f\n\n", d3cod);

     // Reducimos a la mitad de la mitad de la mitad de la mitad (1/16)
     for(x1 = 0; x1 < 56;){
           xa3++;
           ya3 =0;
           for(y1 = 0; y1 < 6;){
                otroPixel[0]= getpixel(surface, x1, y1);
                tempR = otroPixel[0]&surface->format->Rmask;
                tempR = tempR>>surface->format->Rshift;
                tempR = tempR<<surface->format->Rloss;
                red[0]= (Uint8)tempR;

                otroPixel[1]= getpixel(surface, x1, y1 + 1);
                tempR = otroPixel[1]&surface->format->Rmask;
                tempR = tempR>>surface->format->Rshift;
                tempR = tempR<<surface->format->Rloss;
                red[1]= (Uint8)tempR;

                otroPixel[2]= getpixel(surface, x1 + 1, y1);
                tempR = otroPixel[2]&surface->format->Rmask;
                tempR = tempR>>surface->format->Rshift;
                tempR = tempR<<surface->format->Rloss;
                red[2]= (Uint8)tempR;

                otroPixel[3]= getpixel(surface, x1 + 1, y1 + 1);
                tempR = otroPixel[3]&surface->format->Rmask;
                tempR = tempR>>surface->format->Rshift;
                tempR = tempR<<surface->format->Rloss;
```

```
                red[3]= (Uint8)tempR;

                if(x1 < 56 && y1 < 6){
                     d4i = (red[0] + red[1] + red[2] + red[3])/4;
                     tmpd = (red[0] - red[1] - red[2] + red[3])/4;
                     d4tmp[x1][y1] = tmpd;
                     //printf("La serie de codec es: %f\n\n", d4tmp[x1][y1]);
                }

                ya4++;
                nuevoPixel[xa4][ya4] = SDL_MapRGB(surface->format, round(d4i),
round(d4i), round(d4i));
                d4ic += tmpd;
                y1 += 2;
           }

           x1 +=2;
     }

     // De desearlo imprimimos el resultado
     if(imprime == 4 || imprime == 5){

          // Realizamos una limpieza del área para borrar todo aquello que no
sirva
          limpiarImg(surface);

          for(x1 = 0; x1 < 28; x1++){
               for(y1 = 0; y1 < 3; y1++){
                    putpixel(surface, x1, y1, nuevoPixel[x1][y1]);
               }
          }
     }

     d4cod = d4ic/84;

     //Empezamos a generar el código de barras
     if(imprime == 5){
          // Realizamos una limpieza del área para borrar todo aquello que no
sirva
          limpiarImg(surface);

          for(xa5 = 0; xa5 < 28; xa5++){
               for(ya5 = 0; ya5 < 3; ya5++){
                    //printf("La serie de codec es: %f\n\n", d4tmp[xa5][ya5]);
                    if(d4tmp[xa5][ya5] >= 0){
                         otroPixel[3] =  SDL_MapRGB(surface->format,  255,  255,
255);
                         printf("1");
                    } else {
                         otroPixel[3] = SDL_MapRGB(surface->format, 0, 0, 0);
                         printf("0");
                    }

                    yc5++;
                    for(i = 0; i <= 100; i++){
                          putpixel(surface, yc5, i, otroPixel[3]);
                    }
               }
          }

          // Tercer bit complementario
          if(d3cod >= 0){
               otroPixel[2] = SDL_MapRGB(surface->format, 255, 255, 255);
               printf("1");
          } else {
               otroPixel[2] = SDL_MapRGB(surface->format, 0, 0, 0);
               printf("0");
          }

          for(i = 0; i <= 100; i++){
                putpixel(surface, 85, i, otroPixel[2]);
          }

          // Segundo bit complementario
          if(d2cod >= 0){
               otroPixel[1] = SDL_MapRGB(surface->format, 255, 255, 255);
               printf("1");
```

```
            } else {
                 otroPixel[1] = SDL_MapRGB(surface->format, 0, 0, 0);
                 printf("0");
            }

            for(i = 0; i <= 100; i++){
                  putpixel(surface, 86, i, otroPixel[1]);
            }

            // Primer bit complementario
            if(d1cod >= 0){
                 otroPixel[0] = SDL_MapRGB(surface->format, 255, 255, 255);
                 printf("1\n");
            } else {
                 otroPixel[0] = SDL_MapRGB(surface->format, 0, 0, 0);
                 printf("0\n");
            }

            for(i = 0; i <= 100; i++){
                  putpixel(surface, 87, i, otroPixel[0]);
            }
       }
       surfaceGeneral = surface;
}
```

ProyectoTesisDlg.h

```
//-----------------------------------------------------------------------------
//
// Name:        ProyectoTesisDlg.h
// Author:      Jafet Malvaez
// Created:     14/03/2007 04:40:10 p.m.
// Description:
//
//-----------------------------------------------------------------------------

#ifndef __PROYECTOTESISDLG_h__
#define __PROYECTOTESISDLG_h__

#ifdef __BORLANDC__
	#pragma hdrstop
#endif

#ifndef WX_PRECOMP
	#include <wx/wx.h>
	#include <wx/dialog.h>
#else
	#include <wx/wxprec.h>
#endif

//Do not add custom headers
//wxDev-C++ designer will remove them
////Header Include Start
#include <wx/button.h>
////Header Include End

//Compatibility for 2.4 code
#ifndef wxCLOSE_BOX
	#define wxCLOSE_BOX 0x1000
#endif
#ifndef wxFIXED_MINSIZE
	#define wxFIXED_MINSIZE 0
#endif

////Dialog Style Start
#undef ProyectoTesisDlg_STYLE
#define ProyectoTesisDlg_STYLE wxCAPTION | wxSYSTEM_MENU | wxDIALOG_NO_PARENT
| wxMINIMIZE_BOX | wxCLOSE_BOX
////Dialog Style End

class ProyectoTesisDlg : public wxDialog
{
	private:
		DECLARE_EVENT_TABLE();

	public:
		ProyectoTesisDlg(wxWindow *parent, wxWindowID id = 1, const
wxString &title = wxT("ProyectoTesis"), const wxPoint& pos =
wxDefaultPosition, const wxSize& size = wxDefaultSize, long style =
ProyectoTesisDlg_STYLE);
		virtual ~ProyectoTesisDlg();

	public:
		//Do not add custom control declarations
		//wxDev-C++ will remove them. Add custom code after the block.
		////GUI Control Declaration Start
		wxButton *WxButton23;
		wxButton *WxButton22;
		wxButton *WxButton21;
		wxButton *WxButton20;
		wxButton *WxButton19;
		wxButton *WxButton18;
		wxButton *WxButton17;
		wxButton *WxButton16;
		wxButton *WxButton15;
		wxButton *WxButton14;
		wxButton *WxButton13;
		wxButton *WxButton12;
		wxButton *WxButton11;
		wxButton *WxButton10;
		wxButton *WxButton9;
		wxButton *WxButton8;
```

```
		wxButton *WxButton7;
		wxButton *WxButton6;
		wxButton *WxButton5;
		wxButton *WxButton4;
		wxButton *WxButton3;
		wxButton *WxButton2;
		wxButton *WxButton1;
		////GUI Control Declaration End

	public:
		//Note: if you receive any error with these enum IDs, then you
need to
		//change your old form code that are based on the #define control
IDs.
		//#defines may replace a numeric value for the enum names.
		//Try copy and pasting the below block in your old form header
files.
		enum
		{
			////GUI Enum Control ID Start
			ID_WXBUTTON23 = 1024,
			ID_WXBUTTON22 = 1023,
			ID_WXBUTTON21 = 1022,
			ID_WXBUTTON20 = 1021,
			ID_WXBUTTON19 = 1020,
			ID_WXBUTTON18 = 1019,
			ID_WXBUTTON17 = 1018,
			ID_WXBUTTON16 = 1017,
			ID_WXBUTTON15 = 1016,
			ID_WXBUTTON14 = 1015,
			ID_WXBUTTON13 = 1014,
			ID_WXBUTTON12 = 1013,
			ID_WXBUTTON11 = 1012,
			ID_WXBUTTON10 = 1011,
			ID_WXBUTTON9 = 1010,
			ID_WXBUTTON8 = 1009,
			ID_WXBUTTON7 = 1008,
			ID_WXBUTTON6 = 1007,
			ID_WXBUTTON5 = 1006,
			ID_WXBUTTON4 = 1005,
			ID_WXBUTTON3 = 1004,
			ID_WXBUTTON2 = 1003,
			ID_WXBUTTON1 = 1001,
			////GUI Enum Control ID End
			ID_DUMMY_VALUE_ //don't remove this value unless you have
other enum values
		}; //End of Enum

	public:
		void ProyectoTesisDlgClose(wxCloseEvent& event);
		void CreateGUIControls(void);
		void WxButton1Click(wxCommandEvent& event);
		void WxButton2Click(wxCommandEvent& event);
		void WxButton3Click(wxCommandEvent& event);
		void WxButton4Click(wxCommandEvent& event);
		void WxButton5Click(wxCommandEvent& event);
		void WxButton6Click(wxCommandEvent& event);
		void WxButton7Click(wxCommandEvent& event);
		void WxButton8Click(wxCommandEvent& event);
		void WxButton9Click(wxCommandEvent& event);
		void WxButton10Click(wxCommandEvent& event);
		void WxButton11Click(wxCommandEvent& event);
		void WxButton12Click(wxCommandEvent& event);
		void WxButton13Click(wxCommandEvent& event);
		void WxButton14Click(wxCommandEvent& event);
		void WxButton15Click(wxCommandEvent& event);
		void WxButton16Click(wxCommandEvent& event);
		void WxButton17Click(wxCommandEvent& event);
		void WxButton18Click(wxCommandEvent& event);
		void WxButton19Click(wxCommandEvent& event);
		void WxButton20Click(wxCommandEvent& event);
		void WxButton21Click(wxCommandEvent& event);
		void WxButton22Click(wxCommandEvent& event);
	 void WxButton23Click(wxCommandEvent& event);
};

#endif
```

ProyectoTesisDlg.wxform

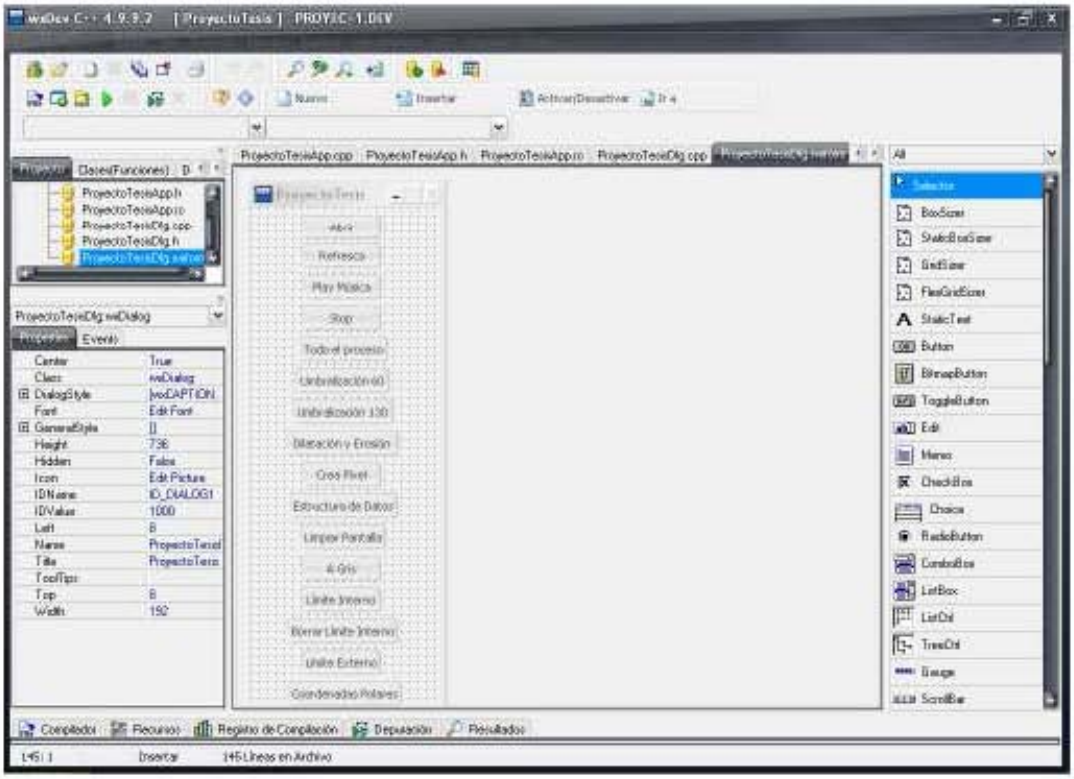

Figura A.1 Ventana de wxDevC++ para el diseño de la pantalla principal de la aplicación.

Referencias bibliográficas

[ALBA, 2006]
ALBA, Castro José Luis et al, Procesado de imagen y visión artificial Vigo, [online]. España 2006 [citado Abril 10 del 2007] Disponible en Internet:
www.gts.tsc.uvigo.es/pi/Morfologia_matematica.pdf

[ALMEIDA, 2004]
ALMEIDA, Rosario Reconocimiento de Iris Uruguay Grupo de tratamiento de Imágenes (Universidad de la República) [online]. 2004 [citado Abril 11 del 2007] Disponible en Internet:
http://iie.fing.edu.uy/investigacion/grupos/gti/timag/trabajos/2004/recon_iris/index.htm

[CALL, 2007]
CALL, Scott, et al, SDL Lib (Simple DirectMedia Layer) MoninMonin Wiki [online]. Mayo 1 del 2007 [citado Mayo 2 del 2007] Disponible en Internet:
http://www.libsdl.org/cgi/docwiki.cgi

[CASTRO, 2007]
CASTRO, Hernández Pedro Enrique Sistema de Identificación de Personas Mediante el Análisis Biométrico del Iris National Instruments Corporation [online]. 2007 [citado Abril 11 del 2007] Disponible en Internet:
http://digital.ni.com/worldwide/latam.nsf/b63ef100ab4b5df486256425006883b7/c9d72e537885c89686256be400703830?OpenDocument

[COMPLUTENSE, 2007]
Universidad Complutense Patron de onda de voz [online]. España, 2007 [citado Mayo 04 del 2007] Disponible en Internet:
http://html.rincondelvago.com/files/9/4/4/000599442.png

[CORZO, 2007]
CORZO, Yuliana, Lógica Difusa [online]. Porlamar, Venezuela, Abril 4 del 2007 [citado Abril 10 del 2007] Disponible en Internet:
http://personales.ya.com/casanchi/mat/difusa01.htm

[DAUGMAN]
DAUGMAN, John. Biometric Decision Landscapes. Cambridge, Inglaterra. 2003

[DIMITRIE, 2005]
Dimitrie Moyasevich B., Ivan Biometría en la Ingeniería Industrial. Lima, [online]. Perú, Febrero 16 del 2005 [citado Agosto 21 del 2007] Disponible en Internet: http://usuarios.lycos.es/imoyasevich/a_ing/temas/biometria_en_la_ingenieria_industrial.htm

[DWAYNE, 2000]
DWAYNE, Phillips Image Processing in C R & D Publications Segunda Edición Abril 26 del 2000

[FERREIRA, 2004]
FERREIRA, Federico Morfología Matemática Facultad de Ingeniería [online]. Uruguay, 2004 [citado Abril 10 del 2007] Disponible en Internet: http://iie.fing.edu.uy/investigacion/grupos/gti/timag/trabajos/2004/deteccion_circulos/tmorfo.html

[GONZALEZ, 2005]
GONZALEZ, Eduardo Introducción a los Biométricos Bolietín Tress [online]. México, Julio 2005 [citado Abril 10 del 2007] Disponible en Internet: http://www.tress.com.mx/boletin/julio2005/biometricos.htm

[GONZALEZ, 2003]
GONZÁLEZ, Díaz Rosio Morfología y Análisis de Imágenes Digitales Universidad de Sevilla, [online]. España, 2003 [citado Abril 08 del 2007] Disponible en Internet: http://www.us.es/gtocoma/morfologia/indice.html

[GONZALEZ, 1993]
GONZÁLEZ, R.C.; Wood, R.e. Digital Image Processing Addison-Wesley Publishing Company, Inc. 1993.

[HOMINI, 2004]
Homini S.A. Plataforma Biométrica [online]. Medellín, Colombia, 2004 [citado Abril 11 del 2007] Disponible en Internet: http://www.homini.com/new_page_1.htm

[IJG]
Independent JPEG Group [citado Agosto 21 del 2007] Disponible en Internet: http://www.ijg.org/

[KIRSCHNING, 2006]
KIRSCHNING, Tlatoa Ingrid Reconocimiento de Voz Universidad de las Americas, [online]. Puebla (UDLAP) Mexico, Junio 6 del 2006 [citado Abril 21 del 2007] Disponible en Internet: http://ict.udlap.mx/people/ingrid/Clases/IS412/index.html

[MAGAÑA, 2005]
MAGAÑA, González Vicente. Propuesta alterna del reconocimiento del iris usando procesamiento morfológico borroso México 2005

[MEDRANO, 2006]
MEDRANO, Garfia Belén Procesamiento de Imágenes Departamento de Matemáticas Aplicadas I (Universidad de Sevilla) [online]. Sevilla, España, Noviembre 22 del 2006 [citado Abril 10 del 2007] Disponible en Internet: http://alojamientos.us.es/gtocoma/pid/pid7/cap5.pdf

[MERKATUM, 2007]
Merkatum Biometric Security & ID Biometría [online]. 2007 [citado Abril 11 del 2007] Disponible en Internet: http://www.merkatum.com

[MORENO, 2005]
MORENO, Luciano Formatos de Imágenes [online]. Mayo 19 del 2005 [citado Mayo 04 del 2007] Disponible en Internet: http://www.desarrolloweb.com/articulos/1974.php

[SCIELO, 2006]
SAAVEDRA-GASTELUM, V., FERNANDEZ-HARMONY, T., HARMONY-BAILLET, T. *et al*. Ondeletas en ingeniería: Principios y aplicaciones. *Ing. invest. y tecnol.* [online]. 2006, vol. 7, no. 3 [citado Abril 10 del 2007], pp. 185-190. Disponible en Internet: http://scielo.unam.mx/scielo.php?script=sci_arttext&pid=S1405-77432006000300005&lng=es&nrm=iso

[SERRA, 1982]
SERRA, J. Image Analysis and Mathematical Morphology, ACADEMIC PRESS LIMITED, Volumen 1, Londres, 1982.

[SHAPIRO, 1980]
SHAPIRO, Willian E.; et al. Enciclopedia Lectum, CREDSA Ediciones y publicaciones, España, 1980.

[SIEUN, 2007]
SIENU (Servicio de Innovación Educativa. Universidad de Navarra), Detalles útiles sobre una imagen [online]. Pamplona, España, 2007 [citado Abril 10 del 2007] Disponible en Internet:
http://www.unav.es/innovacioneducativa/manuales/imagenes.doc

[STOLLNITZ, 2007]
STLLNITZ, D. T., J. E. DeRose, & Salesin, H. D. GRAIL, the Graphics and Imaging Laboratory Wavelets for computer graphics [online]. Washington, Estados Unidos de America, Mayo 30 del 2007 [citado Agosto 21 del 2007] Disponible en Internet: http://grail.cs.washington.edu/pub/stoll/wavelet1.pdf

[TAPIADOR, 2005]
TAPIADOR, Mateos Mariano; et al, Tecnologías biométricas aplicadas a la seguridad, Alfaomega RA-MA, México, 2005.

[TECNOCIENCIA, 2005]
Tecnociencia Biometría (Miradas Únicas) Monográficos de Divulgación [online]. España, Noviembre del 2005 [citado Abril 11 del 2007] Disponible en Internet: http://www.tecnociencia.es/monograficos/biometria/biometria3.html

[WIKIMEDIA, 2007]
WIKIPEDIA, La enciclopedia libre Biometría [online]. España, Abril 04 del 2007 [citado Abril 11 del 2007] Disponible en Internet:
http://es.wikipedia.org/wiki/Biometr%C3%ADa

[WIKIMEDIA_P, 2007]
WIKIPEDIA, La enciclopedia libre Píxel Wikipedia La enciclopedia libre [online]. España, Abril 11 del 2007 [citado Abril 21 del 2007] Disponible en Internet: http://es.wikipedia.org/wiki/P%C3%ADxel