PONTIFICIA UNIVERSIDAD CATÓLICA DE VALPARAÍSO
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA INFORMÁTICA

Librería de Huellas Digitales para Unix

Davidlohr Bueso Arnett

INFORME FINAL DEL PROYECTO
PARA OPTAR AL TITULO PROFESIONAL DE
INGENIERO DE EJECUCIÓN EN INFORMÁTICA.

Diciembre 2008

PONTIFICIA UNIVERSIDAD CATÓLICA DE VALPARAÍSO
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA INFORMÁTICA

Librería de Huellas Digitales para Unix

Davidlohr Bueso Arnett

Profesor Guía: **Ivan Mercado Bermudez**

Profesor Co-referente: **Cristian Rusu**

Carrera: **Ingeniería de Ejecución en Informática**

Diciembre 2008

Resumen

En los últimos años ha habido un crecimiento exponencial en el uso de aplicaciones que incorporan tecnologías biométricos. Hoy en día se puede ver su aplicación en la vida cotidiana de todas las personas, saliendo de los laboratorios de investigación y organizaciones policiales. Ejemplos de esto se pueden apreciar en la venta de computadores portátiles e identificación de pacientes en centros médicos, entre otros. Esta tesis propone una colección de herramientas, en forma de librería, para facilitar la integración de tecnologías de huellas digitales a aplicaciones que corren bajo ambientes Unix. Como base al desarrollo de esta libraría, el autor presenta el estudio del proceso y técnicas, incluyendo algoritmos y metodologías, de los sistemas automáticos de identificación y verificación basados en huellas digitales.

palabras claves: **biometría, huellas, minucias, comparación, imágenes, librería, unix**

Abstract

In recent years there has been an expoonential growth in the use of applications that incorporate biometric technology. Their usage is no longer limited to research laboratories and law enforcement organizations, but rather have found their way into everyday life via laptop computers and the identification of patients in medical centers, among others. This thesis proposes a collection of tools, in the form of a library, to facilitate the integration of digital fingerprint technology in applications that are run on Unix. As background to the development of this library, the author presents a study of the proceses and techniques, incluidng algorithms and methodologies, of automatic identification and verification systems based on digital fingerprints.

key words: **biometrics, fingerprints, minutiae, matching, images, library, unix**

A mi madre y tío, por su incondicional amor y apoyo en los momentos más difíciles.

Índice

Índice de figuras

1. Introducción

Las huellas digitales son la forma más antigua y común de identificación biométrica. Si bien esta disciplina ha sido estudiada por varias décadas, sólo en los últimos años se ha masificado, integrándose a tecnologías de uso diario. Muchas empresas están implementando soluciones biométricas para restringir el acceso a áreas seguras o para controlar horarios de trabajo. Los gobiernos de todo el mundo la incorporan a credenciales de identidad; policías nacionales e internacionales la usan para investigaciones criminales, así como para control migratorio en aduanas, entre otras.

Este documento presenta un estudio de las distintas áreas de la biometría, como la individualidad de las personas, las clasificaciones de sistemas biométricos, y las diversas representaciones de huellas digitales existentes, entre otros aspectos. Se profundiza en el estado del arte de diversos proyectos que experimentan y plantean soluciones para trabajar con huellas digitales, lo que conlleva a un estudio de diversos algoritmos de análisis y tratamiento de huellas digitales, para finalmente diseñar e implementar de una librería para la integración de aplicaciones biométricas.
Si bien la tecnología de huellas digitales no es nueva, a lo largo de los años ha sido muy cerrada y algo casi *mágico* para los usuarios, en donde una persona puede acceder a un sistema sólo con colocar un dedo sobre un sensor. Las herramientas y algoritmos para desarrollar software comerciales que usen, de una forma u otra, huellas digitales, se limita a los que proveen los fabricantes de scanners y los SDK (*Software Development Kit*) que distribuyen. Por razones comerciales, estas empresas distribuyen sus kits de desarrollo de forma cerrada, y el hardware se vende junto con el software para usarlo. Esto es un factor limitante, dado que obliga al usuario a tener que usar los algoritmos de huellas digitales que el vendedor les impone. Esta falta de flexibilidad hace que el proceso biométrico actúe como una caja negra (se entregan los kits en forma de archivos objeto, como DLL, COM para ActiveX[14] o formato *class* para Java), y el usuario muchas veces no tiene un claro entendimiento de lo que se está haciendo, lo cual puede perjudicar el rendimiento y seguridad. Al no contar con el código fuente, el usuario tampoco puede modificar el producto a sus necesidades. Tampoco se puede agregar scanners ajenos al que soporta el SDK (o puede soportar familias de scanners, pero del mismo fabricante), por lo que un usuario no puede desarrollar un software que, en una fase, haga uso de un scanner X y, en otra fase, use un scanner Y. Para esto debe comprar las licencias del SDK para el dispositivo X y otra licencia para el Y. Otro factor negativo es la falta de compatibilidad para distintos sistemas operativos, distintos a Microsoft Windows. Para sistemas Unix, no existen muchas alternativas, siendo estos muy fuertes en el mercado del software, y actualmente obteniendo muchos usuarios finales con Linux u OpenSolaris, en el caso del *Desktop*. Lamentablemente, la gran mayoría de los proyectos Open Source y Free Software[9], relacionados con proveer soluciones de huellas digitales para Unix, fracasan por diversos motivos, sin poder llenar el vacío que actualmente existe. Por otro lado, existe una serie de proyectos privados de software de huellas digitales que jamás se dan a conocer afuera de los laboratorios de investigación o de las empresas que los desarrollan, como en el caso de IBM.
Este proyecto propone una colección de herramientas, en forma de una librería, para ayudar al desarrollo de aplicaciones que usen huellas digitales. Este software brinda una alternativa abierta, sencilla y computacionalmente eficiente a los SDK que actualmente existen en el mercado. La solución propuesta está especialmente pensada en la independencia de los dispositivos de captura de huellas, y la fácil integración para una amplia gama de aplicaciones, brindando una flexibilidad y compatibilidad actualmente inexistente para uso

público.

Se estructura este documento de la siguiente forma:

- En el capítulo 2 se presentan los objetivos generales y específicos del proyecto.
- El capítulo 3 muestra el estado del arte, cubriendo los sistemas biométricos, para introducir al lector en la materia y conceptos que se abordarán durante el resto del documento. Tambien se menciona el proceso de la identificación y verificación de sistemas usando huellas digitales, pasando por algunos proyectos ya existentes.
- El capítulo 4 cubre la metodología a utilizar en la realización del proyecto.
- En el capítulo 5 se presentan métricas para estudiar el rendimiento de sistemas biométricos. Esta sección ilustra conceptos importantes de entender, para luego medir y fundamentar la elección de algoritmos para su implementación en el proyecto.
- En el capítulo 6 se señalan las herramientas y algoritmos que se utilizaron para el desarrollo del proyecto. Se muestran algunos estudios de rendimiento de comparación de huellas, y se justifica su utilización en la librería.
- El capítulo 7 abarca el proceso de extracción de minucias a usar, una de las partes más importantes y críticas del proyecto. Además, se describen las distintas fases y componentes del algoritmo.
- En el capítulo 8 se describe el proceso de matching (o comparación de huellas) basado en minucias a usar en el proyecto y se muestra el funcionamiento interno del algoritmo y una comparación con otras técnicas basadas en correlación.
- El capítulo 9 describe el análisis y diseño de la librería, mostrando las características principales, tales como diseño de la API y almacenamiento de datos, junto con ilustraciones de diagramas de casos de uso.
- En el capítulo 10 se describe la implementación de la librería.
- En el capítulo 11 se muestra el desarrollo del prototipo, describiendo el propósito, sus características principales, y las interfaces de usuario.
- En el capítulo 12 se muestran las diversas pruebas realizadas a la librería, como de funcionalidad, rendimiento computacional y etrés.
- Finalmente, el capítulo 13 cubre las conclusiones del trabajo realizado.

2. Objetivos

2.1. Objetivos Generales

Diseñar e implementar una librería genérica para facilitar la integración de tecnología de huellas digitales a aplicaciones bajo sistemas Unix.

2.2. Objetivos Específicos

Con este proyecto se espera estudiar las distintas áreas de la biometría, con especial énfasis en huellas digitales, analizando y comparando distintos algoritmos y herramientas, junto con sus diversas aplicaciones en el uso cotidiano. Para esto se creó una librería de huellas digitales que permita a desarrolladores usarla fácil e intuitivamente. Los objetivos específicos de este proyecto son:

- Estudiar el área de la biometría, con especial énfasis en huellas digitales, y su actual uso y aplicación.
- Estudiar y comparar diversos algoritmos y herramientas de análisis y tratamiento de huellas digitales.
- Diseñar e implementar una librería para la integración de tecnología de huellas digitales a aplicaciones.
- Diseñar e implementar un prototipo que ejemplifique e ilustre algunas de las funcionalidades más importantes de la librería.
- Diseñar pruebas de rendimiento biométrico, velocidad y uso de recursos computacionales, y aplicarlas a la librería.

3. Estado del Arte

3.1. Problemas de Identidad

A lo largo de la historia se han necesitado mecanismos para verificar si una persona es quien dice ser y tiene acceso a lo que dice tener. Con el crecimiento de la tecnología y uso de sistemas informáticos en la vida diaria ya no se puede confiar en documentos de identificación como único elemento de protección contra personas peligrosas. Entre el 1ro de Marzo y el seis de Abril del 2004 agentes de aerolíneas intentaron en cinco ocasiones prohibir al senador por Massachussets Edward M. Kennedy abordar un avión, por sospechas de terrorismo[34]. Esto es sólo un ejemplo de errores humanos, y lo grave que puede ser confundir y equivocarse con la identidad de un individuo.

Los robos de identidad son delitos muy frecuentes en la sociedad, y pueden causar problemas desastrosos para las víctimas. Se pueden robar contraseñas y números de PIN (como fechas de nacimiento o direcciones particulares) para abrir cuentas bancarias, retirar dinero de cajeros automáticos o incluso obtener préstamos en instituciones financieras. Muchas veces las víctimas facilitan los robos de identidad al usar contraseñas obvias o ser demasiado confiadas e ingenuas. Un estudio internacional indicó que hasta el 25 % de las personas con tarjetas de cajeros automáticos anotan su número de clave en la misma tarjeta. Sólo en el 2002 hubo 3.3 millones de robos de identidad reportadas en los Estados Unidos, y 6.7 millones de víctimas de fraudes de tarjetas de crédito.

La importancia de las contraseñas y el mal uso que la gente le da a ellas hace que los problemas de identidad crezcan aun más en la era informática. No existe una conciencia de contraseñas, lo que facilita el acceso no autorizado a individuos mal intencionados. Las contraseñas muchas veces son anotadas en papel o archivos computacionales, son compartidas por personas *confiables*, como amigos o familiares, se basan en información personal fácilmente obtenible, como nombres o fechas. Es muy común ver contraseñas que son iguales para cada sistema al cual tiene acceso el usuario, por ejemplo, tener una contraseña única para revisar el correo, entrar a la intranet de la empresa, y sacar dinero del banco. Una de las razones de esto es la gran cantidad de contraseñas que un individuo debe administrar en el día a día. Gracias a la expansión de la Internet, un usuario llega a tener un promedio de 11 contraseñas.

Estas representaciones de identidad, como tarjetas, papeles o contraseñas, ya no proveen suficiente seguridad para muchas organizaciones donde es crítico proteger la identidad de los usuarios, como en bancos y gobiernos, entre muchas otras.

3.2. El Proceso Biométrico

La biometría se refiere al reconocimiento automático de individuos basados en características anatómicas (huellas digitales, retina, geometría facial, etc.) y en comportamientos (forma de caminar, firmas, etc.). Esta disciplina se ha masificado exponencialmente en los últimos años debido a que brinda una alternativa eficaz ante los problemas de identidad señalados previamente. Los identificadores biométricos no se pueden compartir, olvidar, duplicar o perder; son una parte intrínseca de un individuo. El reconocer a una persona por partes de su cuerpo, y luego relacionarlo a una identidad establecida externamente, crea una herramienta poderosa y efectiva[18], que brinda tres funcionalidades:

- **Identificación Positiva**: Se basa en verificar si un individuo es quien dice ser usando un sistema de información con una interfaz cómoda y fácil para el usuario final. Estos sistemas son de uso comercial, como celulares, acceso a redes, PDAs y computadores portátiles, entre otros.
- **Identificación de Gran Escala**: Se somete una entrada biométrica (como una huella digital o las coordenadas geométricas de una mano) a una enorme colección o base de datos de entidades previamente enroladas. Algunos sistemas de identificación de gran escala son el Servicio de Registro Civil, donde diferentes organismos gubernamentales pueden identificar personas de todo Chile para diferentes propósitos. Estos sistemas se caracterizan por tener poca interacción con el usuario y por ser muy rápidos a la hora de verificar las identidades.
- **Filtrado (*Screening*)**: Se produce cuando una entrada biométrica es comparada con una lista de personas, previamente determinadas. Generalmente se refiere a criminales o personas extraviadas, por lo que muchas veces son usadas por organismos policiales. Las aplicaciones de filtrado no tienen una fase de enrolamiento bien definida (donde un individuo es registrado bajo circunstancias controladas y seguras); las listas con las que se comparan no son tan grandes como los de gran escala, pero son mayores que las de identificación positiva.

Todo sistema biométrico sigue un proceso común que consta de una serie de pasos que llevan al buen uso de esta tecnología. Como se ilustra en la Figura 1, existen dos tipos de uso de un sistema biométrico: uno es para la autentificación y el otro es para el enrolamiento. Si bien ambos siguen pasos muy parecidos, como el preprocesamiento y la extracción de características, difieren en que uno depende del otro. El enrolamiento es la fase donde se crea una lista, o base de datos, con los individuos que son de interés para el problema, como empleados de una empresa, mientras que la fase de autentificación requiere de las personas ya enroladas para compararlas con la muestra existente. Es muy recomendable que el enrolamiento y autentificación obtengan sus datos de entradas idénticas, como, por ejemplo, el mismo *scanner*, o sensor, biométrico, pues existen diferencias entre los diferentes dispositivos en el mercado que pueden afectar el rendimiento del sistema.

El siguiente paso a realizarse en el proceso biométrico es el preprocesamiento de la muestra, que consta de procesamiento de señales o imágenes digitales, eliminación del ruido, y remoción del fondo que no interesa en la imagen. Esto se denomina segmentación. Una vez hecho el preprocesamiento, se pasa a extraer las características de la muestra; esto es la información que se usará posteriormente para almacenar y comparar. Esta fase varía mucho de acuerdo al área en la que se trabaja, pues cada parte del cuerpo difiere mucho en la información que se puede extraer. El siguiente paso varía dependiendo de si se desea autentificar o enrolar.

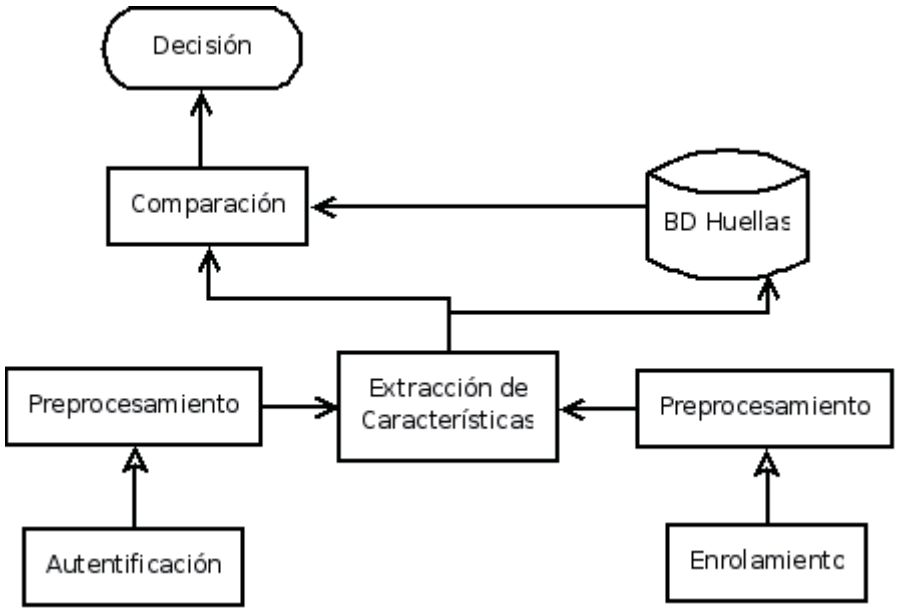

Figura 1: Proceso Biometrico

Para esto último, simplemente se almacenan las características obtenidas en una base de datos, tarjeta inteligente, o cualquier medio conveniente para el almacenamiento de la información. Si se está en un proceso de autentificación de un individuo, se pasa por un *matcher*, o comparador, donde la muestra es sometida a una comparación con las muestras previamente almacenadas. Existen muchas técnicas y algoritmos de comparación, dependiendo del problema, y pueden ser de tipo 1-1 ó 1-N. Con el resultado final de la comparación se puede efectuar una toma de decisiones, tal como permitir al usuario ingresar al sistema, o que pueda entrar a su cuenta bancaria. Una característica importante del *matcher* es que sólo mide similitudes, pues dos muestras del mismo dedo obtenidos en distintos tiempos jamás serán iguales, no así como en el caso de las autentificaciones basadas en contraseñas, donde es correcta o incorrecta, siendo estas 100 % precisas. La biometría necesita un valor de representación que sea suficientemente similar para poder determinar la identificación de un individuo. Esto se llama *threshold*, o umbral. Este umbral regula los índices de errores que determinan el rendimiento general de cualquier sistema biométrico:

- FAR (False Acceptance Rate): Tasa de Aceptaciones Falsas, es decir, el porcentaje de impostores aceptados por el sistema.
- FRR (False Rejection Rate): Tasa de Rechazos Falsos, es decir, el porcentaje de usuarios genuinos rechazados.
- FTE (Failure To Enroll) Fallo de Enrolamiento. Abarca toda la gente que no puede usar el sistema biométrico por razones físicas.

Si se desea tener bajos índices de FAR, se debe incrementar el umbral a un valor adecuado. Para el caso de un bajo FRR, este valor debe disminuir. No existe un valor correcto, pues todo depende del problema y de

las personas que regulan el sistema. Se discutirá más de este tema en el Capítulo 5, Rendimiento de Sistemas Biométricos.

3.3. Clasificaciones Biométricas

La biometría puede dividirse en diversas áreas de investigación, de acuerdo a la característica biológica de una persona, tales como huella digital, iris, y voz, entre otras. No existe un área que sea mejor que otra. Todo depende de los requerimientos y usos que el sistema proveerá, y de las prioridades que deba tener, tales como:

- Universalidad: Todos los usuarios deben tener la característica biométrica. Una persona sin un brazo podría verse incapacitado para usar un sistema basado en geometría de la mano.
- Permanencia: Que la característica biométrica no varíe en el tiempo. Para sistemas basados en huellas digitales, puede ser muy difícil enrolar a un obrero sobre 40 años de edad.
- Cuantitativo: La información obtenida por el sistema biométrico debe ser medible cuantitativamente para la toma de decisiones.
- Rendimiento: Algunos sistemas son más críticos en la seguridad que otros, por lo que necesitan bajos índices de error y un buen desempeño en el procesamiento computacional.
- Aceptación: Los usuarios deben poder aceptar las condiciones de enrolamiento. Mucha gente, por ejemplo, dificultan el escaneo de la retina, al no poder dejar de parpadear cuando pasa el rayo.

Otra forma de clasificar un sistema biométrico se basa en sus características:

1. Cooperativo / No-Cooperativo
2. Abierto / Encubierto
3. Habituado / No-Habituado
4. Atendido / No-Atendido
5. Estandard / No-Estandard
6. Abierto / Cerrado

Un sistema biométrico cooperativo o no-cooperativo se refiere al comportamiento que tiene el usuario cuando está interactuando con el sistema. En un ambiente cooperativo, el usuario coopera con el uso del sistema, por ejemplo, para ingresar a un ambiente de trabajo. Para los casos no-cooperativos, el usuario intenta vulnerar el sistema para ingresar sin que el sistema detecte su verdadera identidad, como en los casos de sistemas de aeropuertos que buscan sospechosos terroristas.
Si un usuario está al tanto de que está expuesto a un reconocimiento biométrico, se denomina abierto. Estos son los más usados por aplicaciones comerciales. Las agencias policiales y forenses generalmente usan

los sistemas encubiertos, donde el usuario desconoce que se está intentando su identificación de manera biométrica.
Los ambientes habituales contra los no-habituales se orientan hacia la cantidad de uso que el usuario tiene con el sistema. Un usuario habitual conoce muy bien el sistema y como interactuar con él, como en los casos para ingresar a un sistema. Sin embargo, casos como el Servicio de Registro Civil, en donde el usuario se somete al sistema una vez en un largo tiempo, no es habitual.
En los sistemas atendidos existe mucha interacción humana, al contrario que en los no-atendidos. Muchas veces la parte de enrolamiento es atendido, pero el *matching* o verificación no lo es, pues puede ser hecho en otra parte y con posterioridad, dependiendo del dominio del sistema.
Cuando se controla el ambiente donde se opera el sistema (como temperatura y luz) se llama estándar. Si no se sabe donde se usará, se denomina no-estándar. Esta clasificación ayuda mucho para el diseño, pues permite saber el tipo de sensor o *scanner* a utilizar.

Durante el diseño de un software biométrico hay que decidir si compartir las características obtenidas de los usuarios a través de múltiples sistemas (tales como ingresar a un computador, entrar al edificio, entrar a lugares seguros, etc.). Si sólo existe una base de datos para acceder a todos estos sistemas, se denomina abierto, de lo contrario el sistema es cerrado.

3.4. Análisis y Representación de Huellas Digitales

Las huellas digitales son una de las formas de autentficación más efectivas que existen. Desde su primera implementación en Scotland Yard en el año 1900, han dado un mecanismo de seguridad efectivo y confiable, y es una de las áreas de la biometría mas desarrolladas que hay actualmente. La identificación basada en huellas digitales se sostiene en dos premisas: (i) **persistencia**: las características básicas de las huellas no varían con el tiempo, ni pueden ser alteradas con cicatrices u otros elementos externos. (ii) **individualidad**: la huella es única para cada individuo, ni siquiera dos hermanos gemelos tiene la mismas huellas digitales.Esto no es aplicable a todas las áreas de la biología, como el ADN, que puede ser repetible entre dos o más individuos.

Una huella digital es una representación de la epidermis de la huella de una persona que se produce cuando se presiona la parte de abajo de un dedo sobre una superficie lisa. La estructura más evidente que se puede apreciar a simple vista es un patrón de crestas y valles, que son las líneas oscuras y las partes claras, respectivamente. Lesiones como quemaduras y cortes no afectan las huellas de una persona, pues cuando se forma nueva piel, se repite el patrón ya existente, es parte integral de la persona.
Muchas veces las crestas y valles corren paralelamente, y pueden bifurcarse o simplemente terminar. Cuando una huella es analizada a un nivel global, se pueden ver regiones donde las crestas y valles asumen formas distintivas, llamadas singularidades. Estas pueden ser clasificados como se ve en la Figura 2.

Existen ocasiones, sobretodo cuando se dispone de una cantidad importante de muestras, cuando es necesario indexar bases de datos de huellas de acuerdo a las singularidades. Esto agiliza el proceso de matching, pues sólo se compara una huella con otras de su mismo tipo.

A un nivel más local y detallado se pueden ver anomalías en las crestas y valles, tales como terminaciones,

Figura 2: Clasificación de Huellas Digitales

islas, bifurcaciones, trifurcaciones y deltas, entre otras. A estas se les denomina minucias, o *minutiae*[1]. Una persona puede llegar a tener hasta 120 distintos tipos de minucias en un mismo dedo[16], y eso tiene directa relación con la calidad de la imagen: a mejor calidad, se pueden detectar más minucias, y, por ende, más preciso será el proceso de matching. Si bien pueden considerarse muchos tipos de minucias (los más comunes se muestran en la figura a continuación), en la práctica no se entra en los detalles específicos de cada minucia. Esto simplifica en cierto grado el complejo proceso de diferenciar automáticamente las características de una huella.

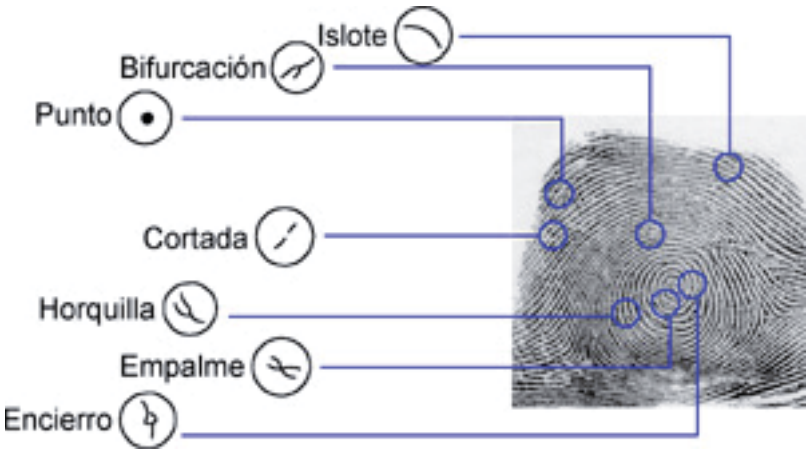

Figura 3: Tipos de Minucias Comunes

[1]pequeños detalles, en latín

3.5. Obtención de Huellas Digitales

Un sistema biométrico basado en huellas digitales siempre funcionará mejor con muestras de huellas obtenidas de fuentes confiables y seguras; Sin embargo no siempre ocurre así. Existen tres formas principales para obtener una huella digital, y cada sistema debe ser diseñado para usar alguna de estas (o una combinación de varias), por lo que es de suma importancia conocer la fuente de obtención de las muestras en la fase de análisis, pues afectará todo el ciclo de vida del software.

3.5.1. Generadores Sintéticos

Este método consiste exclusivamente en generar huellas digitales artificiales usando software especial que se basa en patrones y algoritmos matemáticos. Los generadores sintéticos de huellas nacen a raíz de problemas durante la investigación de algoritmos de extracción de minucias y matching, pues, muchas veces, la base de datos que se disponía era muy pequeña, con imágenes de baja calidad y poco reusables (ya que cuando se *usa* una huella, ya no se puede analizar nuevamente. Las huellas sintéticas ayudan a desarrollar, probar y optimizar métodos y procesos biométricos en centros de investigación, pues emulan huellas reales con una alta precisión, y a un bajo costo.
Los generadores se basan en la generación de una huella maestra que representa un patrón de las características únicas de una huella sintética, sin tomar en cuenta ruido, condición de la piel, rotación, o cualquier elemento que pueda *contaminar* una huella. Posteriormente se generan huellas *hijas* (las cantidades pueden ser infinitas), aplicando, en distintas medidas, estos elementos que distorsionan la imagen, creando una muestra que emula una huella real. Uno de los programas más conocidos para generar huellas es el SFINGE[37] (**S**ynthetic **Fin**gerprint **Ge**nerator), desarrollado por el Departamento de Computación de la Universidad de Bologna, en Italia.

3.5.2. Mecanismos Offline

Si bien esta forma es la más antigua, aún es utilizada, sobretodo por organismos policiales o forenses. Una huella se puede obtener con mecanismos offline de dos formas: usando métodos de tinta negra o con formas forenses. El primero consiste en cubrir la superficie del dedo con tinta negra y luego colocarlo sobre un papel, para luego ser incorporado manualmente al sistema. Este fue el método usado por muchos años por el Servicio de Registro Civil de Chile para obtener una cédula de identidad.
En muchas ocasiones los organismos policiales deben obtener huellas bajo condiciones especiales, como en casos de crímenes o investigaciones. Una persona deja su huella al tocar un objeto, pues se le adhiere una capa de humedad y/o grasa con la huella, estas son llamadas **huellas latentes**. Las técnicas más comunes para limpiar estas huellas consisten en la aplicación de polvos o químicos (como nitrato y yodo). Una vez que se tiene una clara impresión de la huella, es posible almacenarla en un sistema para su posterior uso.

3.5.3. Scanners de Huellas

Sin duda esta es la técnica más común para obtener una huella digital, sobretodo en el uso comercial. En el último tiempo, con la expansión de la biometría en la industria, han aparecido muchos fabricantes de

dispositivos o *scanners* de huellas. El uso de un scanner para capturar huellas digitales también es conocido como *live-scan sensing*. La Figura 4 ilustra el funcionamiento básico de todo scanner.

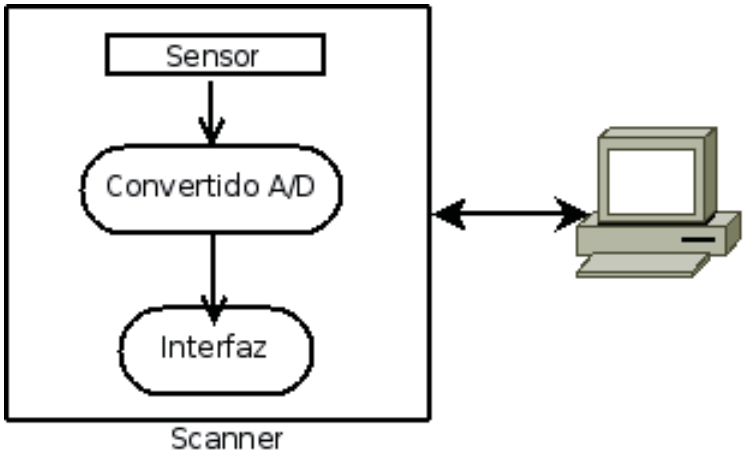

Figura 4: Scanner de Huellas Digitales

Un sensor lee la superficie del dedo y forma la imagen en un formato analógico, para posteriormente ser digitalizado a través de un convertidor A/D (Analógico a Digital). Existe una interfaz que sirve como puente de comunicación (tales como enviar imágenes o comandos USB) entre los dispositivos externos (como un computador personal) y el scanner. Si bien todos los componentes son importantes, el elemento vital de todo dispositivo es el sensor, que puede ser óptico, de ultrasonido, o de silicio (también conocido como *solid-state*). Cada tipo de sensor crea diferentes tipos de imágenes donde varían factores como el dpi, tamaño, peso y costo, entre otros.

Otros factores a considerar para determinar qué dispositivo usar en un sistema son:

- FPS: Los *frames per second* son un indicador de la cantidad de imágenes que un scanner puede capturar y enviar al host en un segundo. A mayor FPS, mejor es la tolerancia a movimientos del dedo, lo que se traduce en una mejor interacción con el usuario.
- Encriptación: El canal de comunicación entre el scanner y el host muchas veces es un punto de ataque frecuente por usuarios maliciosos, y asegurar el canal brinda un importante nivel de seguridad.
- Detección del Dedo: Algunos scanners pueden detectar automáticamente la presencia de un dedo sobre el sensor, sin tener que constantemente capturar y procesar marcos inútiles, lo que permite un mejor desempeño.

3.6. Proyectos Existentes

Con el tiempo, la biometría, y en especial las áreas que se preocupan de las huellas digitales, ha logrado resultado interesantes, lo que crea avances en la industria. A continuación se describen algunos proyectos,

privados y gubernamentales, que sirven como base para este proyecto, obteniendo de cada uno ellos ideas sobre diseño y estructuración, así como también información de resultados negativos, para no caer en ellos.

3.6.1. NIST Fingerprint Image Software - NFIS

Este proyecto es el resultado de años de investigación del FBI, en conjunto con el NIST (National Institute of Standards and Technology) y el DHS (Department of Homeland Security), resultando en una serie de programas independientes para sistemas operativos Unix, que permiten manipular imágenes de huellas digitales de diversa índole. Se orienta a usuarios muy familiarizados con el área de la biometría, sobretodo para ambientes de investigación y académicos. Una de las características más importantes de este sistema es que ayudó a definir una serie de estándares para el almacenamiento de la información que se obtiene de huellas digitales, catalogado como ANSI/NIST-ITL 1-2000 'Data Format for the Interchange of Fingerprint, Facial, Scar Mark & Tatoo (SMT) Information'. Este estándar a sido adoptado por todos las grandes agencias policiales de los Estados Unidos[38]. Sin embargo, hoy en día se ha masificado mucho en el sector privado, siendo incorporado a la gran mayoría de SDKs de los fabricantes de *scanners*. Otro producto que nació de las investigaciones hechas por el FBI es el formato de imagen WSQ (Wavelet Scalar Quantization).

NFIS se compone de una serie de paquetes principales, cada uno con una serie de programas pequeños (o comúnmente llamados de utilidad) que hacen distintas tareas que permiten trabajar y analizar huellas digitales. Estos paquetes abarcan procesos como la extracción de características de las huellas (minucias), limpieza de las imágenes, análisis de calidad y *matching*. Un elemento interesante es que permite clasificar huellas digitales según características comunes, lo que mejora el rendimiento del proceso de *matching*, pues se reduce la cantidad máxima de imágenes posibles de comparar. Esto aumenta considerablemente el desempeño de servidores de *matching*.

3.6.2. Finger Verification System - FVS

Este proyecto tiene objetivos y diseños muy parecidos al que se está presentando en este informe: busca los defectos de los SDK, e intenta crear una librería simple, pequeña, rápida y portable para que programadores puedan fácilmente agregar tecnología biométrica a sistemas de software[29]. El problema del Finger Verification System es que fue abandonado, sin siquiera tener algo que mostrar, como un ejecutable, documentación ni código fuente. Sin embargo, proporciona valiosas referencias técnicas para orientar a personas que están inicializándose en la biometría.

3.6.3. Fprint

Este proyecto, desarrollado por estudiantes de la Universidad de Manchester, apunta a ser una solución biométrica *open source* para Linux que permite unificar diferentes *scanners* bajo una API común, permitiendo a desarrolladores abstraerse del dispositivo. Incorpora una serie de elementos del NFIS como la extracción de minucias y *matching*.

Si bien Fprint es un proyecto que ha sido ampliamente adoptado en el mundo del software libre, ha sido muy criticado por el hecho que usa dependencias que restringen la portabilidad, tales como dispositivos incrustados y otros sistemas operativos. El diseño del proyecto está muy basado en el de los SDK, ya que sólo puede usarse con una lista de *scanners* y funciona como un sistema monolítico que no permite coexistir con otras implementaciones. Actualmente se está descomponiendo el sistema para hacerlo más modular y optimizar los usos de recursos computacionales, lo que generará una mayor flexibilidad.

3.6.4. Software Development Kits

Los SDKs son las alternativas más usadas para desarrollar aplicaciones biométricas de uso comercial, pues vienen como parte de los scanners, y son desarrollados y soportados por los mismos fabricantes. Estas herramientas proveen una forma rápida y confiable que ayuda al desarrollador a crear soluciones de alta calidad. Los SDK generalmente soportan una variedad de plataformas y lenguajes de programación tales como Java, Visual Studio .NET y C++, entre otros.

4. Metodología

Debido a la complejidad del tratamiento y análisis de huellas digitales, así como también a la imposibilidad de capturar todos los requerimientos de una sola vez, se optó por usar el paradigma evolutivo de software para la librería. Este paradigma permite desarrollar versiones cada vez más completas del sistema a medida que se descubran nuevas funcionalidades o surjan necesidades de cambiar y mejorar aquellas ya existentes. Las librerías, por naturaleza, no tienen un usuario final específico, y el diseño evolutivo ayuda a que una retroalimentación[30] pueda ir mejorando y desarrollando nuevas versiones de acuerdo a las necesidades que vayan surgiendo, a través de un ciclo de vida iterativo.

A nivel global, el sistema se descompone en partes, o componentes, y estas van evolucionando independientemente unas de otras, surgiendo iteraciones e incrementos. A medida que se va trabajando cada parte del sistema se desarrolla una iteración, y para cada incremento de estas iteraciones se aplican independientemente las etapas características de un desarrollo lineal o cascada, teniendo un análisis, diseño, implementación y pruebas. Con este paradigma de trabajo se logra una evolución paralela del sistema, adaptándose a las dificultades propias del problema. Para el prototipo, por su estricta dependencia a las funcionalidades de la librería, también se usó el paradigma evolutivo, iterativo e incremental. Cada vez que se haga un cambio a la librería, se estudia si la modificación vale la pena ilustrarla en la aplicación, y de ser así, se desarrollan las iteraciones, y para cada una, un incremento.

Pensando en las interacciones que el usuario pueda tener con la librería y el prototipo, se usó el paradigma de análisis y diseño orientado a objetos, usando UML como lenguaje de modelamiento, pues describen las especificaciones de los sistemas. En cuanto a la implementación, se establece una metodología estructurada, en donde se implementa el prototipo, la lógica de la librería y el acceso a los dispositivos de hardware (los scanners de huellas). Sin embargo, como parte del diseño, se define un estándar de creación de interfaces para lenguajes orientados a objetos que se puedan comunicar con la librería, y usar todas sus funcionalidades con otras tecnologías, como Java. Esta interfaz, por motivos de tiempo, no se implementó.

5. Rendimiento de Sistemas Biométricos

Es importante estudiar los rendimientos de los sistemas biométricos y las métricas para determinarlos. Al conocer la forma en que se evalúa y determina la calidad de los sistemas, se pueden analizar de mejor manera los algoritmos, y establecer con un mejor criterio, junto con su justificación, aquellos a usar en el proyecto. Las muestras obtenidas de alguna característica física de un usuario pueden variar significantemente a través del tiempo. Por ejemplo, en el caso de huellas digitales, factores tales como la ubicación del dedo sobre el sensor, la presión ejercida sobre él, o la condición en la que se encuentra la piel, son variaciones que siempre existen y causan problemas en los sistemas de comparación. Estas se denominan **variaciones intra-clase**. La siguiente figura muestra dos impresiones del mismo dedo de una persona, capturadas en días distintos. Es importante notar cómo difieren en cuanto a la rotación, traslación y distorsión. Por otro lado, dos huellas capturadas de personas distintas pueden ser bastante parecidas. Por ejemplo, existen personas que tienen una alta semejanza facial debido a factores genéticos (como padre e hijo, o gemelos). Esto se conoce como **similitudes inter-clase**.

Figura 5: Ejemplo de la variación intra-clase. Se ven dos muestras distintas del mismo dedo de una persona obtenidas en días distintos, con sus respectivas minucias marcadas. Debido a la diferencia de posicionamiento del dedo en cada captura y la distorsión, la cantidad y posición de las minucias difiere (33 y 26 en la izquierda y derecha, respectivamente).

Un sistema biométrico puede cometer dos tipos de errores: falsa aceptación (*false match*) y falso rechazo (*false non-match*). Cuando la variación intra-clase es alta, dos muestras de la misma característica biométrica pueden no ser reconocidas como iguales, lo que conlleva a un error de falso rechazo. Una falsa aceptación ocurre cuando dos muestras de distintos individuos son incorrectamente reconocidas como iguales, debido a una alta similitud inter-clase. Así, las medidas básicas de la precisión, o exactitud, de un sistema biométrico son el porcentaje de falsas aceptaciones (conocido como FAR - False Accept Rate) y el porcentaje de falsos rechazos (conocido como FRR - False Rejection Rate).

Un FRR de 2 % indica que, en promedio, 2 de cada 100 intentos legítimos de usar un sistema son rechazados. La mayoría de los errores de falso rechazo se debe a una mala interacción del usuario con el sensor biométrico, y pueden ser fácilmente corregidos al simplemente permitir al usuario usar el sensor nuevamente.

Este caso es similar a situaciones en que el usuario intenta autentificarse con su contraseña y se equivoca, y se da la posibilidad al usuario de ingresar su clave nuevamente. Un FAR de 0.01 % indica que, en promedio, 1 de cada 10.000 impostores son reconocidos como usuarios legítimos del sistema. Sin embargo, es importante enfatizar que la seguridad de un sistema biométrico, aunque no sea perfecta, es mejor que aquella basada en contraseñas o números PIN.

Existen dos tipos de errores, además de los descritos anteriormente. Si una persona no puede interactuar correctamente con el sistema biométrico (por ejemplo, por razones de interfaz de usuario, o por una mala calidad de la característica biométrica), el sensor no podrá procesar las características del individuo, por lo que el individuo no podrá enrolarse. Esto se denomina porcentaje de error de enrolamiento (o Failure to Enroll Rate - FTER). En algunos casos, una muestra particular no puede ser capturada o procesada con certeza. Este error es conocido como error en captura (o Failure to Capture Rate - FTCR).

En todo sistema biométrico se debe definir un umbral n que indica si una comparación es suficientemente cercana o no para establecer la identificación de una persona. Este umbral varía de acuerdo al sistema y dominio del problema. Cuando se hace una comparación, se obtiene un número (o *score*) S perteneciente al intervalo $[0, 1]$, que es comparado con el umbral para determinar si el usuario es genuino o un impostor. Si se establece un umbral alto, entonces el sistema se vuelve menos tolerante, y se requerirá más exactitud para que un usuario sea identificado, pero al mismo tiempo será más seguro ante usuarios que no pertenezcan al sistema.

El FRR y FAR están en función del umbral n, por lo que si se aumenta este número, el FAR se reducirá, pero el FRR se incrementará, y vice versa. Es por esto que, dado un sistema biométrico, no es posible disminuir simultáneamente ambos errores simplemente variando el umbral. Se debe establecer un nivel de equilibrio entre estos índices, de acuerdo al uso que se dará al sistema y las prioridades del problema. Es posible que no se sepa de antemano el uso específico que tendrá el sistema, por lo que no es claro el nivel del umbral a establecer, razón por la cual se recomienda reportar el rendimiento del sistema para distintos valores del umbral. Esto se logra a través de una *curva ROC - Receiver Operating Characteristic*. Esta curva muestra la relación entre el FAR y el número de aceptaciones genuinas, conocida como TAR (o *True Acceptance Rate*), que viene dada por la diferencia del 100 % con la FRR:

$$TAR(n) = 1 - FRR(n) \tag{1}$$

Desafortunadamente, el rendimiento de un sistema ante una población (base de datos) específica es aleatorio, y no puede ser medido, sino que sólo estimado a partir de datos empíricos, y las estimaciones del desempeño dependen mucho de los datos con los que fue probado[16].

5.1. Errores en Pruebas de Rendimiento

Desafortunadamente, no existe un estándar para realizar pruebas a sistemas biométricos, y es por esto que es fácil para fabricantes de scanners o diseñadores de algoritmos hacer pruebas de modo que favorezcan

sus propios sistemas. Los resultados de las pruebas son influenciados por diversos factores, tales como la calidad de la huella tomada, la población y la cantidad de dedos enrolados de un individuo, entre otros.

Al calcular los índices de FAR y FRR, se debe tener en cuenta el criterio de Doddington, el cual establece que para que la razón de error esté dentro del 25 % del verdadero índice de error (con alrededor de 90 % de certeza) hay que observar al menos 30 instancias de ese error. Entonces, si se dice que el FAR de un sistema es de 1 en 100.000, y sólo se hicieron 100.000 pruebas al sistema, entonces no es una cifra muy confiable, pues sólo se observó en una ocasión el error. Sin embargo, si se realiza la prueba tres millones de veces, y se obtiene el mismo valor de FAR, entonces esta cifra sí es mucho más precisa.

Es importante mantener la independencia entre las pruebas que se realizan a un sistema. Las pruebas de Bernoulli son aquellas que producen un resultado binario, verdadero/falso, o match/no match en el caso de la biometría. De esta manera, cada prueba es estadísticamente independiente de otra. Uno de los problemas de este tipo de pruebas es que debido a que cada huella tiene una probabilidad de distribución de error con otra huella, al someterla a una prueba, ya no puede volver a ser utilizada. Esto hace que sea muy costoso y poco práctico realizar una prueba Bernoulli[7].

6. Elección de Herramientas

Se definió una serie de métricas para seleccionar el software a usar como base para este proyecto. La primera fue el rendimiento del matcher, el cual debe desempeñarse de excelente manera, sobretodo con huellas capturadas de scanners biométricos. Por otro lado, el matcher debe estar basado en minucias, y no en correlación. Debido a la cercanía que tiene el extractor de minucias y el matcher, es muy importante que ambos sean parte del mismo proyecto y exista una cierta dependencia mutua, pues de lo contrario se puede afectar el rendimiento del sistema. Otro factor a considerar es la seriedad del proyecto. Deben existir estudios y casos de uso para poder tener certeza de que se cumplen las características que el proyecto dice tener.

Se decidió utilizar partes del NIST Biometric Image Software, o NBIS[28], pues éste es, lejos, proyecto más indicado para usar, de acuerdo a las métricas planteadas anteriormente. Este es una colección de pequeñas aplicaciones para el procesamiento de huellas digitales, desarrollado y publicado por el *Image Group* del US National Institute of Standards and Technology (NIST) para el FBI y el Department of Homeland Security (DHS). Actualmente es usado para el control de criminal y fronterizo, por consecuencia, es respaldado y utilizado por el gobierno estadounidense, lo que brinda una confianza inalcanzable por otros proyecto biométricos. Dado que fue creado[23] por empleados federales de los Estados Unidos, no está sujeto a protección de derecho de autor o propiedad intelectual (*copyright*), siendo de dominio público, de acuerdo con la Ley 17, sección 105, del Código de los Estados Unidos[5]. Como consecuencia, puede usarse sin ningún problema para este proyecto.

Se usaron dos componentes de este software para integrarlo al proyecto:

- `MINDTCT`: Es un sistema de detección y extracción de minucias.
- `BOZORTH3`: Es un sistema de comparación de huellas digitales basado en minucias, soportando sólo la comparación de dos huellas a la vez.

Para poder integrar NBIS al proyecto, se debe transformar y unificar el código fuente de todas las aplicaciones a usar, de modo que se puedan usar desde una librería y formen parte de la API (*Application Programming Interface*).

6.1. Estudio de Rendimiento de Bozorth

El NIST hizo una serie de estudios de rendimiento de su software usando bases de datos del gobierno de Estados Unidos, los que incluyen personas de distintas edades, sexo y procedencia. La procedencia de las huellas varía desde enrolamientos estándares con scanners biométricos, hasta huellas latentes, obtenidas de casos forenses. Estos estudios y pruebas fueron automatizados y denominados *Verification Test Bed - VTB*. Se usaron 16 computadores con las siguientes propiedades de hardware y software:

- Procesador Dual 1.8Mhz Intel Xeon
- 1GB RAM

- Sistema operativo Linux RedHat 7.2 instalado en cada nodo

Los estudios del NIST varian enormemente en el tamaño y la calidad de las bases de datos, con huellas desde 216 personas hasta alrededor de 600.000. Las estadísticas de rendimiento del algoritmo Bozorth muestran que con un FAR común de 1 %, en el mejor de los casos (con huellas de buena calidad - obtenidas de scanners) se logra un TAR de 99 %, y en el peor de los casos (sólo usando mecanismos offline de adquisición de huellas - como las latentes) el TAR es de sólo un 71 %.

A continuación se muestran los resultados de dos pruebas, una pequeña, y otra de gran escala. Ambas usan como métricas los índices de error de FAR, FRR y EER, entre otros; también se computa la curva ROC para analizar la dependencia entre los índices, y es una de las piezas fundamentales a medir en cada prueba.

6.1.1. Estudio de Verificación con SD14

La base de datos usada para esta prueba se denomina *NIST Special Database 14* y corresponde a 2.700 criminales de los archivos del FBI, obtenida mediante el uso de tinta, y posteriormente digitalizadas a imágenes con formato WSQ. El estudio consistió en la comparación de los dedos pulgares e índices derechos e izquierdos de cada individuo, por lo que, en total, se forman 4 conjuntos de 2.700 huellas cada uno.

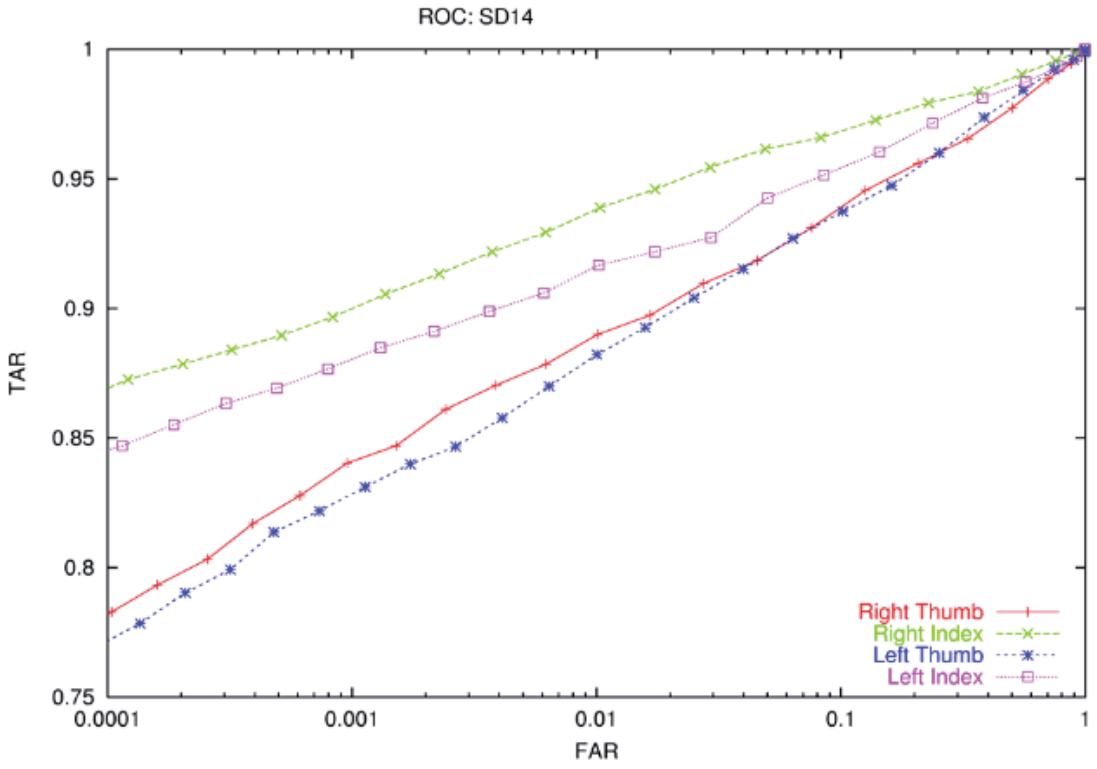

Figura 6: Curva ROC para la SD14 (Fuente NIST Verification Test Bed)

La figura 6 muestra la curva ROC resultante de las cuatro distintas matrices de 2700x2700, y se puede apreciar que el mejor resultado fue para el dedo índice derecho, pues se obtiene un 94 % de TAR y sólo un

1 % de FAR en comparación con los pulgares, así como también se comparten de mejor manera los dedos de la mano izquierda. En general estos resultados fueron relativamente malos (obviamente el ideal es que el TAR tienda a 99.99 %), sin embargo, hay que tener en cuenta que debido a la naturaleza de las huellas, y el limitado tamaño del universo de las muestras, no se puede esperar buenos resultados. Este es un ejemplo concreto de la importancia de tener huellas de buena calidad, y optimizadas, antes de enrolar usuarios en un sistema biométrico.

6.1.2. Estudio de Verificación con DHS10

Este estudio fue realizando usando la base de datos del *Department of Homeland Security*, la que comprende cientos de miles de huellas de criminales de los Estados Unidos. Se usaron, de manera aleatoria, 52.000 individuos, cada uno con las huellas de ambos pulgares, obteniéndose un total de 104.000 huellas. Esta base de datos consta principalmente de huellas obtenidas por scanners biométricos, por lo que son de buena calidad. El estudio se basó en comparar todas las huellas con todas las huellas del pulgar izquierdo. Los resultados se ilustran en la figura 7.

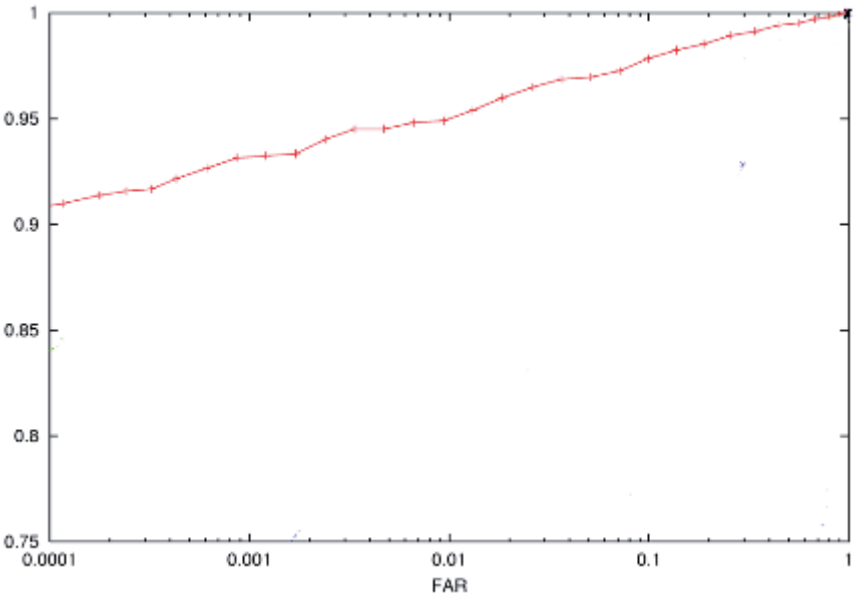

Figura 7: Curva ROC para la DHS10

Como se puede observar, el resultado es mucho mejor que el del SD14, más que nada por una superior calidad de las huellas usadas. La pendiente de la curva no es tan pronunciada, lo que implica resultados más parejos (sin tanta variación) en los distintos puntos del eje del FAR.

7. Proceso de Extracción de Minucias

A lo largo de los años en el desarrollo de la tecnología de huellas digitales, se han comparado dos huellas digitales usando características llamadas minucias. Esto es debido a su aceptación general en cortes de justicia asi como su gran precisión científica. Estas características son puntos en la piel de un dedo donde terminan crestas (partes obscuras dentro de una huella) o donde se bifurcan (partes más claras dentro de la huella). Generalmente existen alrededor de 50 minucias en una huella. Sin embargo, pueden existir más (llegando a los 100), o mucho menos (alrededor de 20). Esto depende de las condiciones de la piel de la persona, y puede variar de acuerdo al medio usado para obtener la huella, sexo, edad y profesión de la persona: por ejemplo, un trabajador de la construcción de 50 años tendrá mucho menos minucias que un estudiante secundario.

Para buscar y comparar huellas, se establecen puntos en un plano cartesiano bidimensional, junto con un grado de orientación para cada tipo de cresta. En la siguiente figura se muestran los dos tipos de principales de minucias: bifucaciones y términos de cresta.

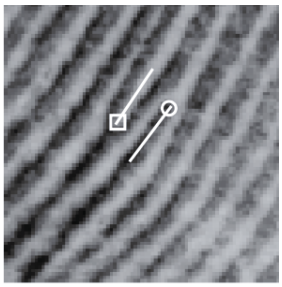

Figura 8: Minucias: Bifurcación (Cuadrado) y terminación (circulo)

La ubicación de cada minucia es representada por coordenadas dentro de la imagen, con unidades en pixeles, cuyo origen se encuentra en la parte superior izquierda de la imagen. La orientación de la minucia se representa en grados, con cero grado apuntando horizontalmente y a la derecha, e incrementándose en dirección contraria al reloj. La orientación de terminaciones de crestas se determina al medir el ángulo entre el eje horizontal y la línea que comienza en el punto de la minucia y continua hacia el medio de la cresta. La orientación de una bifurcación se determina midiendo el ángulo entre el eje horizontal y la línea que comienza en el punto de la minucia y continua hacia el medio de la bifurcación. Esto se ilustra abajo, en la Figura 9.

Antes de describir los algoritmos que se usan para extraer las minucias que se detectan en una huella, es importante describir cómo se analiza localmente una huella. La imagen se divide en una serie de bloques horizontales, verticalmente adyacentes, formando una malla, con dimensiones constantes para cada uno. Muchas veces se tiene que compartir información de los bloques vecinos, es decir, los bloques que son adyacentes al bloque analizado, y los resultados de un bloque ayudan a determinar los resultados de otro bloque. Para esto se usan ventanas, que constan de 9 bloques (dimensiones de 3x3), como se ilustra en la Figura 10. El uso de ventanas y el intercambio de información ayudan a minimizar la discontinuidad en los valores de los bloques, evitando cambios bruscos cuando se pasa de un bloque a otro.

La Figura 10 ilustra una ventana (en blanco) que abarca un bloque más pequeño (en gris). Se definen tres parámetros: el tamaño de la ventana L, el tamaño del bloque M y el desplazamiento, u *offset*, del bloque

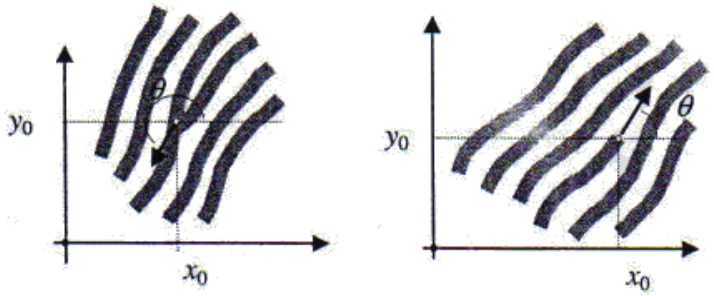

Figura 9: Orientación de Minucias. Izquierda: Ángulo θ para una bifurcación de cresta. Derecha: Ángulo θ para un término de cresta

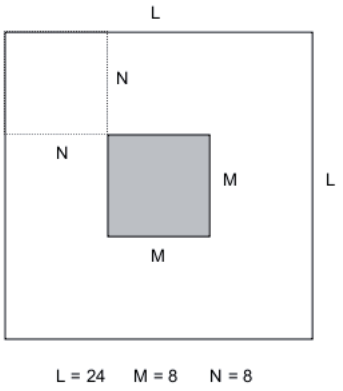

Figura 10: Ventana de 24x24 píxeles que abarca 9 bloques de 3x3.

desde origen de la ventana N. Como resultado, la imagen se divide en una malla de bloques de 8x8 píxeles, cada una asociada a una ventana de 24x24 píxeles. Es importante mencionar que, si bien los bloques no pueden estar sobrepuestos unos con otros, las ventas si pueden, razón por la cual dos ventanas pueden compartir los mismos bloques. Uno de los factores a considerar cuando se usan bloques y ventanas para analizar una huella es determinar cómo tratar las orillas de la imagen, pues, generalmente, las dimensiones de ésta no son múltiplos de las dimensiones de los bloques. Para solucionar este inconveniente, se agrega la cantidad necesaria de píxeles para cubrir la diferencia del bloque con el perímetro de la imagen. Estos píxeles son de una intensidad media de gris.

7.1. Mapa de Direcciones

Una de las fases más importantes en la detección y extracción de minucias es identificar las direcciones de los flujos de las crestas. Este mapa representa las áreas en la imagen con suficiente estructura de crestas para detectar puntos de terminaciones o bifurcaciones de crestas; es decir, aquellas bien formadas y claramente visibles. Todas las fases posteriores usan, de una forma u otra, el mapa de direcciones, razón por la cual es muy importante definir bien las direcciones de los flujos de cada cresta.

Cada bloque en una ventana es girada incrementalmente (de 0 a 180 grados), analizando cada orientación de los píxeles del bloque. Por defecto, se analizan 16 orientaciones distintas en un semicirculo, generando una incremento de 11,25 grados cada vez. En cada orientación, los píxeles de cada fila rotada en la ventana son sumados, formando un vector de las dimensiones de la ventana, es decir, los ángulos. Asimismo, las 16 orientaciones de cada bloque también se suman, formando un vector de 16 dimensiones. Entonces, si se toma el ejemplo de la Figura 10, con la ventana de dimensiones 24x24 píxeles, y los bloques de 8x8 píxeles, cada píxel de cada fila en la ventana es almacenado en un vector de 24 dimensiones.
Posteriormente, a cada dimensión del vector se aplica una formula que computa los cosenos y senos, y se suma el cuadrado de cada uno, obteniendo la dirección del píxel de cada bloque. Finalmente, se decide la orientación del bloque de acuerdo al mayor ángulo de cada uno de los píxeles del bloque. La siguiente ecuación se emplea para obtener la dirección de cada píxel i, j dentro de la imagen.

7.2. Binarización

La binarización es el proceso de convertir cada píxel en escala de gris a binario (blanco o negro), este es uno de los pasos más críticos en el proceso de extracción de minucias, pues el resto de los pasos usan la huella binaria. Esto mejora el contraste entre las crestas y valles dentro de la imagen, y, en consecuencia, facilita la extracción de minucias[36]. La Figura 11 muestra el resultado de este proceso, y se compara con la huella original, en escala de gris.

Figura 11: Comparación de una huella en escala de gris contra una binarizada

El valor de cada pixel se determina en base a la dirección que tiene el bloque que lo abarca, y en caso de que no tenga una dirección definida, automáticamente se asigna blanco.

Si se detectan flujos de crestas, entonces se usa una malla para analizar el píxel. Se comienza por ubicar el centro de la malla (las dimensiones de esta siempre son constantes: 7x9 píxeles) sobre el píxel de interés, y luego se gira para que las filas estén paralelas y las columnas perpendiculares a la dirección de la cresta. Esto se ilustra el la Figura 12.

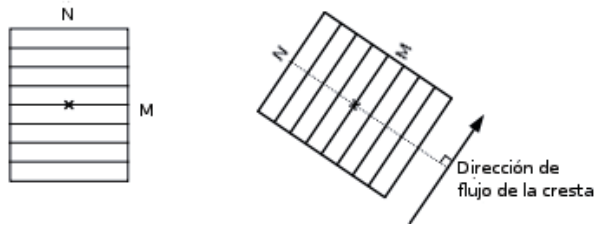

Figura 12: Se centraliza el pixel sobre una malla de dimensiones NxM y luego se gira para quedar paralelo al flujo de dirección.

Al ir rotando la malla, se suman las intensidades de gris que se encuentran en cada fila, para obtener la suma del píxel S_1. También se suman las intensidades de gris de toda la malla, osea rotandolo 360 grados S_2. Posteriormente, para calcular el valor binario a asignarle al píxel, se multiplica la suma del píxel S_1 por la cantidad de filas en la malla, y luego se compara este resultado con la suma de intensidades de gris de toda la malla, obteniendo una segunda suma, S_2. Luego, si la multiplicación de filas por S_1 es menor a S_2, entonces al píxel se le asigna un valor negro, de lo contrario se asigna blanco. Este procedimiento se realiza para todos los bloques de la imagen.

7.3. Detección de Minucias

Una vez obtenida la imagen binaria, se puede proceder a encontrar patrones de bifurcaciones y terminaciones que aparecen en los píxeles. Estos patrones son combinaciones de ciertos píxeles negros y blancos, que pueden repetirse un número indeterminado de veces. Por ejemplo, para determinar que ocurre una terminación de una cresta, debe existir una cantidad N de píxeles negros consecutivos, y luego al menos un píxel blanco (donde se indica que aparece un valle, por ende, término la cresta). La Figura 13 muestra la forma en se pueden cubrir todas los tipos de terminaciones de crestas con un simple patrón de píxeles de 2x3 . La columna del medio, marcada con un asterisco ('*'), puede repetirse una o más veces.

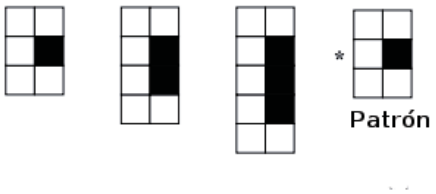

Figura 13: Ejemplo de patrón de píxeles para la terminación de una cresta.

Uno de los beneficios de usar estos tipos de patrones es que sirven para representaciones horizontales y verticales, y simplemente se debe rotar 90 grados hacia una u otra. En la imagen binaria se analizan píxeles consecutivos buscando las secuencias que coinciden con los patrones. A lo largo de años de investigación en el tema, se definieron una serie de patrones que cubren la totalidad de posibles representaciones de las minucias, las que se ilustran a en la Figura 14.

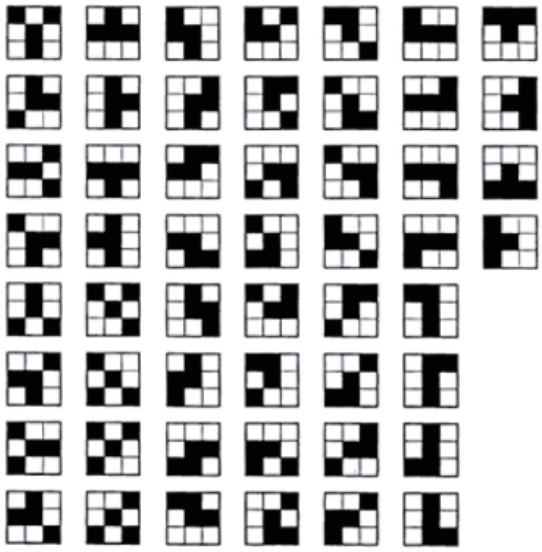

Figura 14: Patrones de píxeles para la terminación de una cresta.

7.4. Eliminación de Minucias Falsas

El mecanismo de detección de minucias presentado anteriormente asegura no ignorar ningún tipo de detalle que pueda perjudicar el proceso de extracción. Sin embargo, se dan muchas ocasiones en que se consideran minucias que realmente no lo son debido a problemas de binarización[31], y se perjudica el rendimiento del matching. Este paso busca eliminar ciertos puntos detectados en los patrones, pero que, sin embargo, no son minucias. Es muy importante eliminar estos puntos, y de hecho, si no se está seguro si un punto es una minucia legitima o falsa, lo mejor eliminarla. Se decide eliminar las minucias falsas ya finalizando el proceso de detección y extracción debido que hay menos puntos a considerar (solo aquellos que se obtuvieron en el paso anterior), y, por ende, cada punto puede ser analizado con mayor atención y detalle[24].

Es muy común que en huellas latentes existan manchas de tinta o suciedad de forma elíptica, que sean confundidas con terminaciones de crestas y, por lo tanto, catalogadas como minucia durante el proceso de detección. Sin embargo, deben ser eliminadas. Estos se denominan islas.

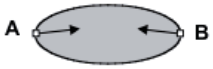

Figura 15: Eliminación de una isla.

Los requisitos para determinar una isla es una cierta distancia entre ambas minucias (alrededor de 16 píxeles), un cierto grado de dirección de ambas (deben estar prácticamente opuestas) y que el perímetro de la elipsis sea menor o igual a una cierta cantidad de píxeles. Si se cumplen todas estas condiciones, entonces se eliminan ambas minucias.

Los hoyos son pequeños vacíos dentro de una cresta producidos por suciedad o transpiración en el dedo. A diferencia de las islas, los hoyos se caracterizan por tener sólo una minucia, y con el perímetro de la elipsis más pequeño, de hasta 15 píxeles. La figura 16 ilustra este tipo de imperfecciones.

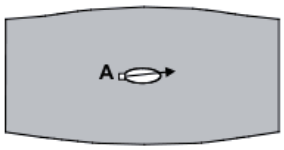

Figura 16: Eliminación de un hoyo.

Un gancho nace desde una cresta donde no alcanza a bifurcarse, y queda simplemente como una pequeña cresta conectada con otra cresta de mayor tamaño, como se ilustra en la figura 17. Al igual que en el caso de las islas, para determinar un gancho, la distancia entre ambas minucias debe ser de hasta 16 píxeles, y en direcciones de flujo opuestas. Otro factor para determinar un gancho es que las minucias de interés sean del mismo tipo, es decir, ambas deben ser terminaciones o bifurcaciones.

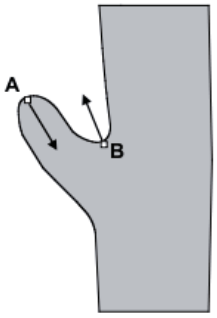

Figura 17: Eliminación de un gancho.

Otro error común son las superposiciones de puntos, debido a que la presión ejercida sobre el dedo en el sensor puede hacer que una cresta se superponga con otra. Los criterios para establecer la ocurrencia de una superposición son que ambas minucias deben estar a un máximo de 8 píxeles entre sí, así como sus direcciones opuestas, como se indica en la figura 18. Si estos dos factores se cumplen, luego se calcula la dirección de la línea que une ambas minucias, y se compara con la dirección de la primera minucia (de izquierda a derecha). Si la diferencia de ambas direcciones es menor o igual a 90 grados, entonces se eliminan ambas minucias.

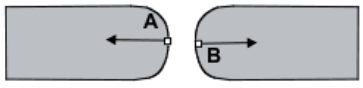

Figura 18: Eliminación de una sobre posición.

7.5. Obtención de Vecinos

En una huella, cada minucia contiene una serie de vecinos (cada una siendo una minucia proveniente de la misma huella). Cada sistema de identificación automática de huellas (*Automatic Fingerprint Identification System* - AFIS) contiene su propia topología de vecinos[38], y el NIST lo define como sigue:

- Para cada minucia en la huella, busca otras minucias que estan en el misma coordenada x, pero variando la y, osea verticalmente (en la misma columna de pixeles).
- Luego, sigue buscando, a la derecha de la minucia en questión, los demás puntos, en las demás columnas de pixeles.
- A medida que van encontrando candidatas a minucias vecinas, las inserta en una lista, la cual puede tener un máximo de cinco. Para cada minucia nueva a insertar, calcula su distancia euclidiana, entre la minucia en questión y su posible vecino, y la inserta ordenadamente, de menor a mayor distancia. En caso de que la lista ya contenga cinco minucias, se va eliminando el último, osea el de distancia más lejana. Para esto se usa la siguiente fórmula:

$$dist = \triangle(x_1, x_2)^2 + \triangle(y_1, y_2)^2 \tag{2}$$

Donde x_1, y_1 son las coordenadas de la minucia en questión y x_2, y_2 son las coordenadas de la minucia vecina. Así se logra tener hasta los cinco vecinos más cercanos de cada minucia, y como se puede ver, un vecino de una minucia no tendrá necesariamente a esa minucia como vecina, pues pueden existir otras más cercanas para esa.

Otro factor que se considera en esta topología, es la cantidad de crestas que intersectan la minucia con cada uno de sus vecinos. Para esto, se traza una línea recta entre ambos puntos, y como se trabaja con una huella binarizada, se cuentan la cantidad de píxeles negros que intersectan esa línea. La Figura 19 muestra el resultado de este proceso.

Figura 19: Una minucia (azul) con sus cinco vecinos (rojo)

8. Proceso de Matching

Gracias a la cercanía que existe con el área forense y la aceptación en los tribunales de justicia, el matching basado en minucias es el método más conocido y más usado hoy en día. Se pueden establecer dos representaciones para muestras de una huella digital, donde **H** y **H'** denota una representación de la huella 1 y de la huella 2, respectivamente. Cada huella tiene una serie de minucias asociadas a ella, representadas por **m** y **m'**, respectivamente. Como se estableció en el capítulo anterior, una minucia puede ser catalogada como un trio $m = \{x, y, \theta\}$, donde x e y son las coordenadas cartesianas y θ el grado de orientación de la minucia. Luego, se puede establecer la relación:

$$H = \{m_1, m_2, \ldots, m_q\}, m_i = \{x_i, y_i, \theta_i\}; i = 1 \ldots q \tag{3}$$

$$H' = \{m'_1, m'_2, \ldots, m'_w\}, m'_j = \{x'_j, y'_j, \theta_j\}; j = 1 \ldots w \tag{4}$$

donde q y w denotan la cantidad de minucias para H y H', respectivamente. Una minucia m'_j en H', y una minucia m_i en H, son consideradas iguales (que coinciden y provienen del mismo dedo, o huella) si se encuentran a cierta distancia entre sí, y a un cierto grado de dirección (también llamado orientación)[26]. Como es muy improbable que ambas minucias estén exactamente en el mismo punto geométrico y a un mismo grado, se define una cierta tolerancia r_0 y θ_0, respectivamente, para la distancia y grado de orientación.

$$distancia(m_i, m'_j) = \sqrt{(x'_j - x_i)^2 + {y'_j - y_i}^2} \leq r_0 \tag{5}$$

$$orientacion(m_i, m'_j) = min(|\theta'_j - \theta_i|, 360^0 - |\theta'_j - \theta_i|) \leq \theta_0 \tag{6}$$

En la Figura 20 se ilustra un ejemplo de cuatro minucias detectadas para las huellas H y H', junto con su grado de dirección. La imagen inferior izquierda muestra la forma en que se consideran coincidentes si dos minucias están dentro de un cierto espacio y ángulo.

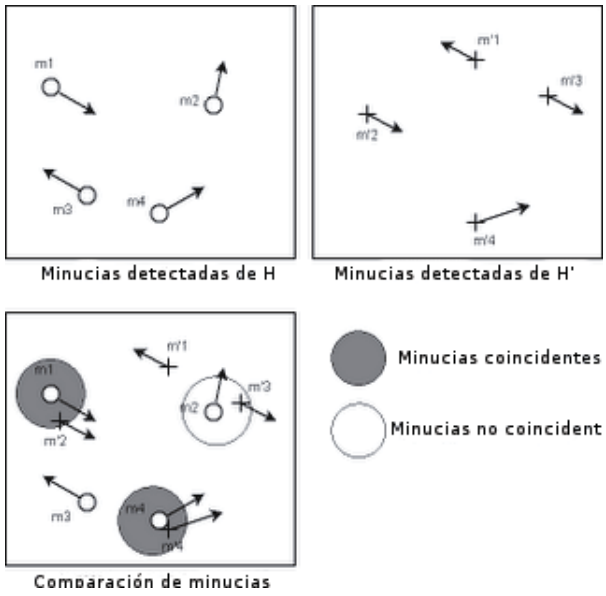

Figura 20: Distancia y orientación necesaria para que dos minucias sean coincidentes

8.1. Bozorth

El algoritmo escogido, llamado `Bozorth` (en honor a su diseñador, el agente del FBI Allan Bozorth), toma las minucias extraídas de dos huellas digitales y los compara de forma 1-1. Es importante tener en cuenta que:

1. Las minucias son exclusivamente en base a su ubicación y orientación, representadas por $\{x, y, \theta\}$.
2. El algoritmo está diseñado para ser tolerante a la rotación y traslación, lo que hace que sea más tolerante a las variaciones intra-clase.

Este algoritmo consta de tres fases principales: (i) construcción de tablas de comparación, que consiste en crear dos tablas, una para cada huella a comparar; (ii) construcción de una tabla de compatibilidad, la que se usa para almacenar las minucias que pueden resultar compatibles entre cada huella; (iii) recorrer la tabla de comparación, en donde se establece el resultado final del matching.

8.1.1. Tablas de Comparación

El primer paso consiste en analizar y computar las distancias y ángulos de cada minucia con las demás dentro de una huella. Estas medidas son almacenadas en una **tabla de comparación de minucias** y son lo que dan al algoritmo la independencia de rotación y traslación. La Figura 39 es una ilustración de dos

minucias: **k**, localizadas en la parte inferior izquierda, con una orientación hacia abajo y a la derecha; y **j**, en la parte superior derecha, con una orientación apuntando hacia arriba y a la derecha de la huella. La minucia k se encuentra en los puntos $\{x_k, y_k\}$, con un grado de orientación t_k. Asimismo, la minucia j se encuentra en los puntos $\{x_j, y_j\}$, con una orientación de t_j. La distancia entre k y j está representada por d_{kj}, y es la distancia más corta entre ambos puntos, por lo que se denota como una línea recta. Esta distancia se mantiene relativamente constante entre las dos huellas a comparar, sin importar cuánta rotación y desplazamiento existan en las muestras.

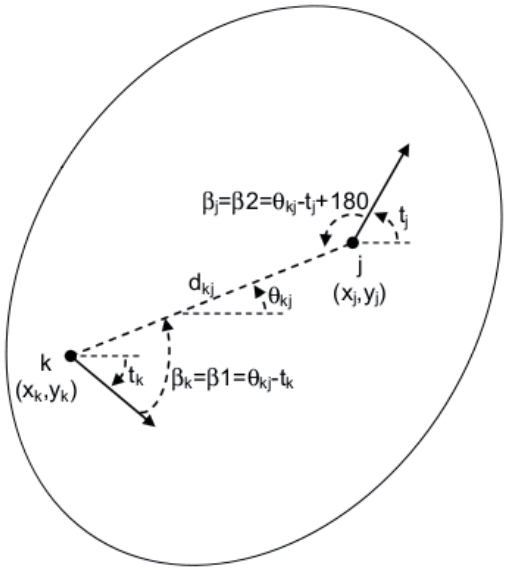

Figura 21: Análisis de dos minucias usando el algoritmo Bozorth

Para medir la rotación, se calcula el ángulo entre la orientación de cada minucia y la línea recta que *une* cada minucia. De esta manera, estos ángulos se mantienen relativamente constante, sin importar cuánta rotación exista. En la ilustración anterior, el ángulo θ_{kj} de la línea entre las minucias k y j se obtiene calculando el arco tangente de la pendiente de la línea. Los ángulos β_k y β_j son computados en relación a la linea al incorporar θ_{kj} en cada orientación t de la minucia. Es importante mencionar que las orientaciones están sujetas al intervalo]-180, 180], con 0^0 apuntando horizontalmente a la derecha y aumentando los grados contrario hacia el reloj. Para cada par de minucias se ingresa en la tabla la sectupla:

$$\{d_{kj}, min(\beta_k, \beta_j), max(\beta_k, \beta_j), k, j, \theta_{kj}\} \tag{7}$$

Las entradas almacenadas en la tablas de comparación son ordenadas en base a la distancia entre cada par de minucias d_{kj} en un orden de mayor a menor. Se construye una tabla para cada huella a comparar, por lo que en un matching 1-1 existirán dos tablas.

8.1.2. Tabla de Compatibilidad

El siguiente paso consiste en buscar las entradas *compatibles* entre ambas tablas descritas anteriormente, y marcarlas como candidatas de minucias iguales al ingresarlas a una tabla única, denominada **tabla de compatibilidad de huellas**. La Figura 22 muestra dos huellas de un mismo dedo con diferencias de rotación, en cada huella. Al igual que en la sección anterior, se denotan dos minucias, siendo P la tabla de comparación para la huella superior, y G la tabla de la huella inferior.

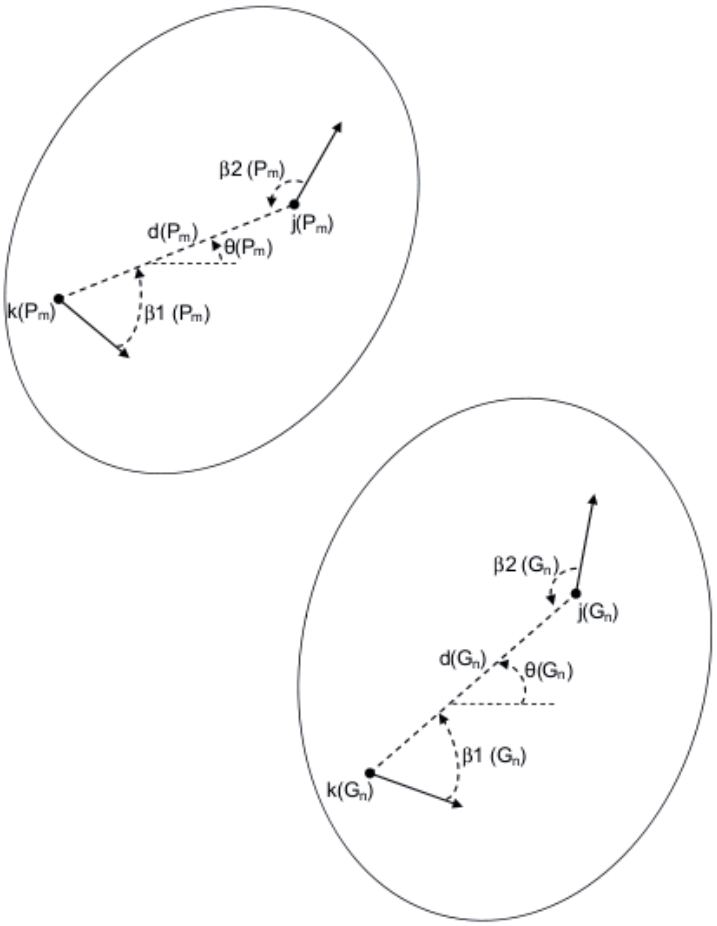

Figura 22: Análisis de dos minucias usando el algoritmo Bozorth

La m-ésima entrada en la tabla P se denota como P_m, y la notación individual, como k o j, para cada valor almacenado en la tabla, se representa con $k(P_m)$, mientras que la distancia entre las dos minucias guardadas en P_m se denota como $d(P_m)$. Se desarrollan tres pruebas para determinar si las entradas en las tablas P_m y G_n son compatibles. La primera consiste en probar si las distancias están dentro de la tolerancia r_0. Las otras dos pruebas verifican si los ángulos de las minucias están dentro de la tolerancia θ_0. Si la distancia y ángulos de ambas tablas están dentro de una cierta tolerancia, entonces se agrega a la tabla de tolerancia la quintupla:

$$\{\triangle_\beta(\theta(P_m), \theta(G_n)), k(P_m), j(P_m), k(G_n), j(G_n)\} \quad (8)$$

Es así como una tabla de compatibilidad incorpora dos pares de minucias, uno para la primera huella $k(P_m), j(P_m)$ y otro para la segunda huella, $k(G_n), j(G_n)$. Siguiendo la misma notación, $k(P_m)$ corresponde con $k(G_n)$ y $j(P_m)$ corresponde con $j(G_n)$. Esta tabla puede verse como una serie de asociaciones de compatibilidad entre dos pares de posibles minucias iguales. Estas asociaciones representan aristas de un grafo (conocido como **grafo de compatibilidad**.

8.1.3. Recorrer Tabla de Compatibilidad

Para determinar la cercanía que tiene una huella con otra, se recorre la tabla de compatibilidad, comenzando desde varios puntos. A medida que se va recorriendo la tabla, se unen distintas entradas, creando el grafo de compatibilidad, donde la unión de una entrada con otra representa una arista. Asimismo, cada entrada en la tabla de compatibilidad representa un nodo del grafo. Una vez completada esta fase, y con el grafo ya formado, se obtiene el resultado del match (*match score*) al sumar la cantidad de entradas unidas con otras. Entonces, a mayor cantidad de nodos unidos entre sí, mayor es el score, lo que hace más probable ambas huellas en cuestión sean del mismo dedo.

8.2. Comparación entre Matching Basado en Minucias y Correlación

Si bien las técnicas de matching basados en minucias, como el mostrado en este capítulo, son las más utilizadas, existe otro método muy común para comparar dos huellas, denominado **matching basados en correlación**. Esta técnica se basa, mayormente, en el estudio de patrones de imágenes más que de puntos, como en el caso de minucias.
La tarea de comparar dos huellas usando correlación comienza por ubicar el centro de la imagen (la cual no es necesariamente el centro de la huella) y extraer esa fracción de la imagen, formando una nueva, como se ilustra en la Figura 23. Posteriormente se compara la nueva imagen y se analizan las diferencias. Si bien no se entrará en los detalles de algoritmos de matching usando correlación, sí existen diversas métricas para comparar ambas técnicas.

Debido a que los matchers que usan minucias solo necesitan la información de los puntos, el peso de las características extraídas es muy bajo, de alrededor de 150 bytes para una huella con 40 minucias extraídas, lo que permite su fácil y rápido almacenamiento en todo tipo de medios. Por otro lado, los matchers basados en correlación necesitan de la imagen, con un peso como mínimo el doble del de las minucias (con una alta compresión). Esto implica un sacrificio en cuanto a calidad o almacenamiento, y en bases de datos importantes, el costo de almacenar miles de huellas puede ser muy alto. El flujo de información por redes es otro factor a considerar cuando se estudia el tamaño de almacenamiento entre ambos métodos, siendo el de correlación mucho menos óptimo, sobretodo entre redes de amplia cobertura.
Los cambios físicos en un dedo, como cortes, o cicatrices, causan problemas a los matchers que usan correlación, debido a que son muy sensibles a alteraciones en la imagen. Las técnicas que emplean el uso de

Figura 23: Centralización de una huella.

minucias no son tan afectadas por cambios físicos, pues basta con tener alrededor del 20 % de las minucias disponibles para hacer un buen match[26], lo que permite perder hasta un 80 % de las características de la huella, cosa que es muy difícil de lograr. En consecuencia, el uso de minucias es mucho más práctico en la vida real que el de correlación.
Otro problema importante para las técnicas de correlación es la falta de estandarización en la industria, mientras que existen diversos estándares ANSI para los que usan minucias. Se establece que el matching basado en minucias facilita el diseño de un sistema de verificación robusto, simple y rápido[17].

9. Análisis y Diseño de la Librería

9.1. Diagramas de Casos de Uso

Se diseñó un conjunto de diagramas de casos de uso, basados en los requerimientos funcionales y las distintas interacciones que el usuario pueda tener con el sistema. Se puede ver el diagrama de casos de uso general de las operaciones que efectúa el usuario con el sistema (Figura 24).

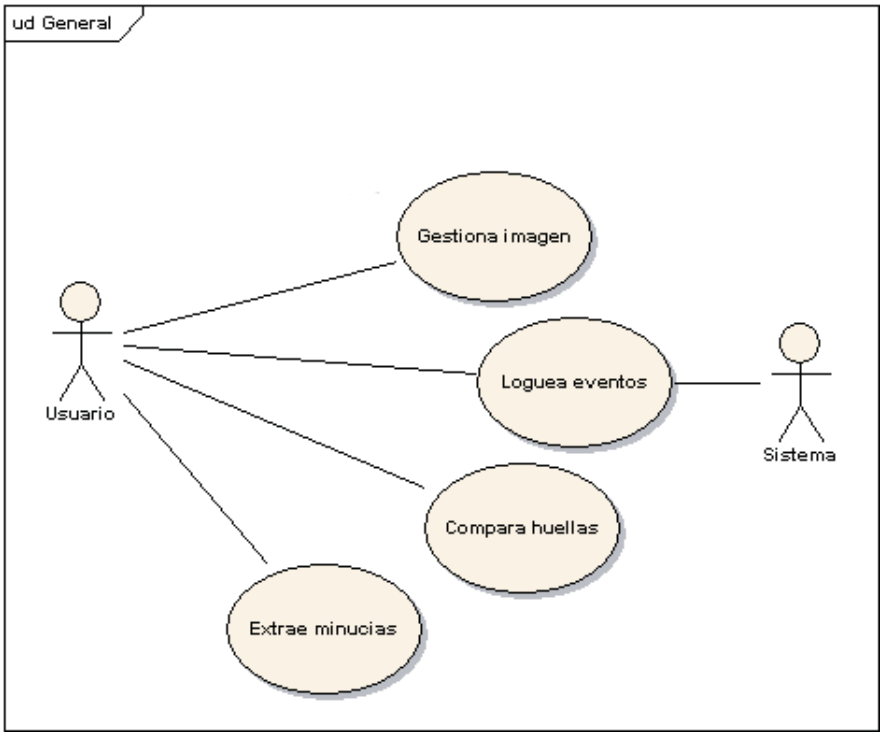

Figura 24: Diagrama General de Casos de Uso

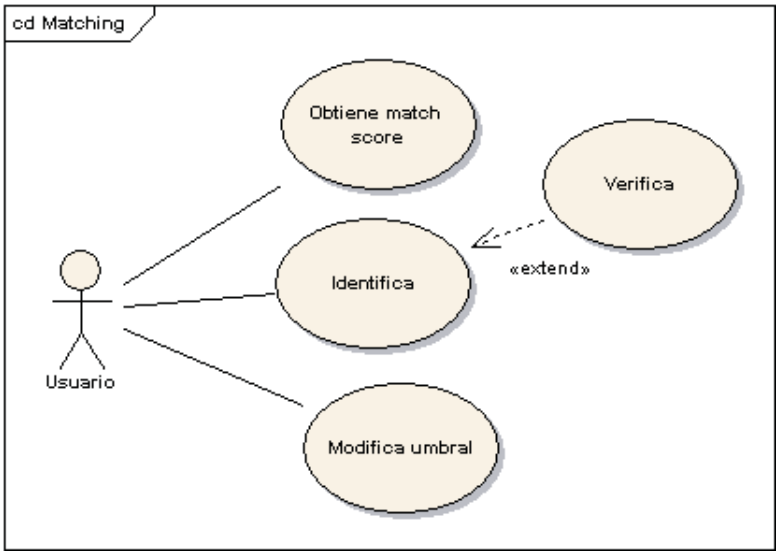

Figura 25: Diagrama de Casos de Uso de Matching

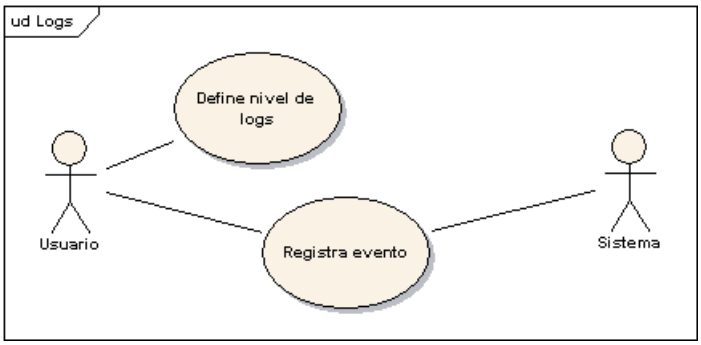

Figura 26: Diagrama de Casos de Uso de Logger

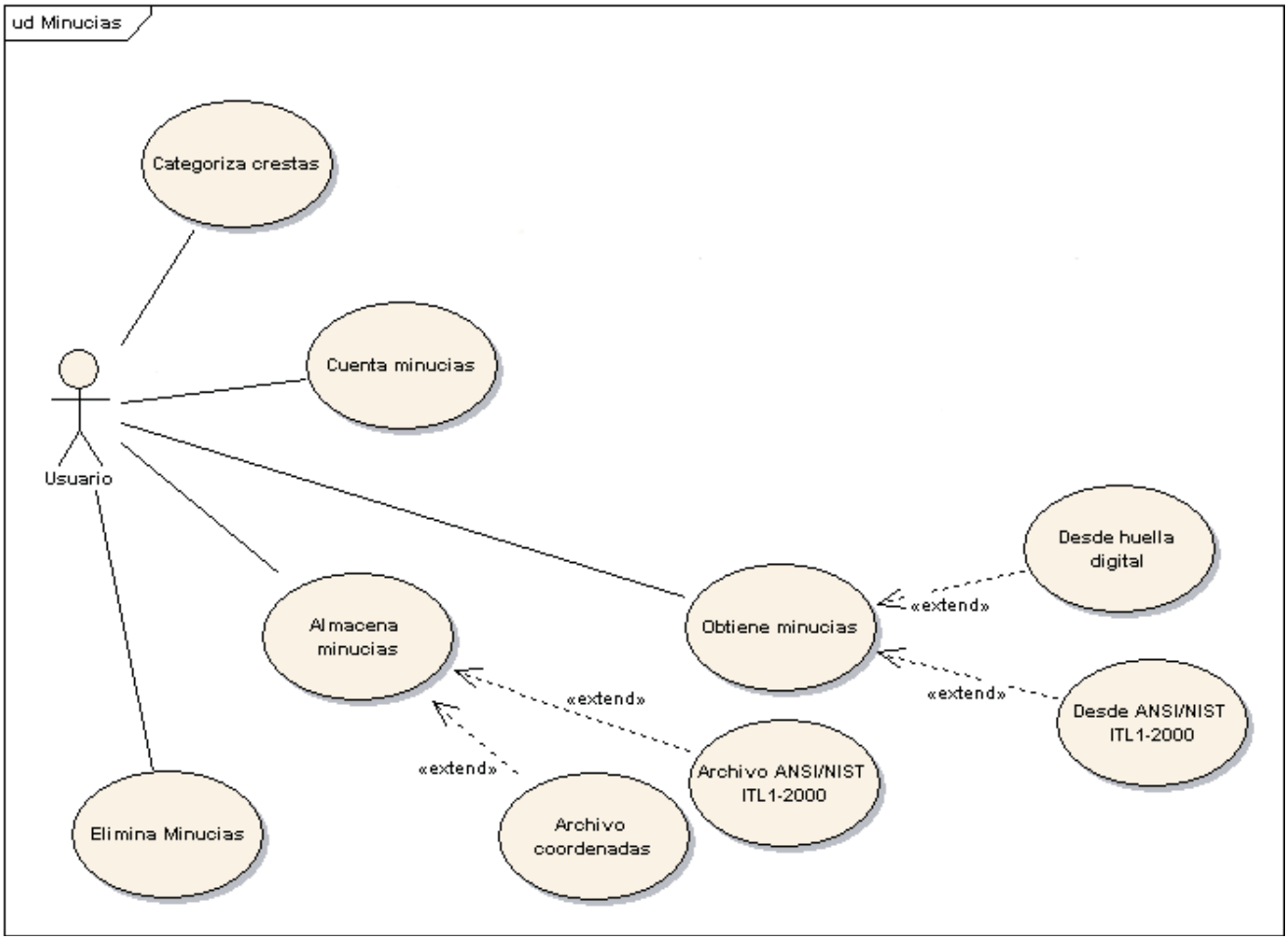

Figura 27: Diagrama de Casos de Uso de Minucias

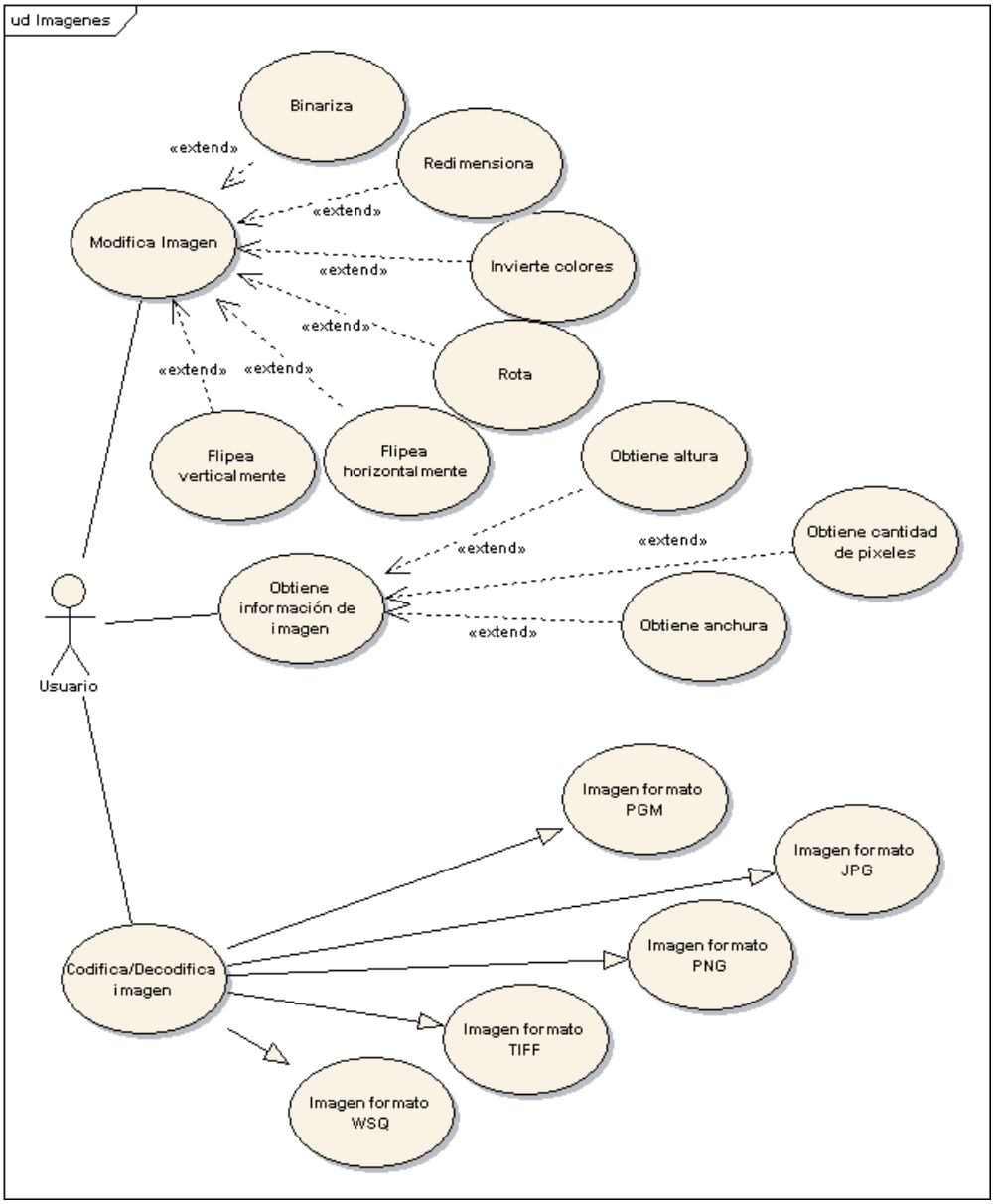

Figura 28: Diagrama de Casos de Uso de Imágenes

9.2. Diseño de la API

Este proyecto apunta a proveer mecanismos sencillos para integrar el tratamiento de huellas digitales en aplicaciones. Una API (*Application Programming Interface*) genérica bien diseñada asegura que desarrolladores puedan integrar cualquier software sin necesidad de conocer detalladamente el funcionamiento de los scanners ni de las huellas digitales. Se ha diseñado una API que se enfoca en la simplicidad, ofreciendo funcionalidades claves descritas en esta sección.

9.2.1. Verificación

La verificación consiste en una comparación 1-1 de una huella previamente enrolada contra una nueva capturada recientemente, resultando muchas veces en una toma de decisiones en tiempo real, tales como otorgar acceso a un sistema, si se considera que ambas huellas coinciden y provienen del mismo dedo de un individuo. La verificación es adecuada para loguearse en sistemas. Por ejemplo, el acceso a una estación de trabajo puede requerir el nombre de usuario y la captura de la huella.

9.2.2. Identificación

La identificación compara una muestra con una base de datos de huellas, produciendo comparaciones 1-N. En general, se usa la identificación para investigaciones y estudios de gran escala, y requiere muy poca interacción humana[19]. Comparado con la verificación, la identificación permite una autenticación más flexible. Para ingresar a un sistema, la identificación permite implementar un *login* que permita el usuario ingresar con cualquiera de sus dedos, incrementando la seguridad. Asimismo, se puede usar la identificación para escenarios donde no se sepa qué usuario está usando el sistema, por lo que la captura de una huella puede reemplazar el nombre de usuario y contraseña.

9.2.3. Extracción y Almacenamiento de Minucias

Es posible extraer las minucias de una huella y almacenarlas en diversos tipos formatos de archivos (planos). Gracias a que se sigue un estándar ANSI, pueden usarse estos archivos con minucias con otros algoritmos de matching; esta funcionalidad da una flexibilidad y libertad al usuario. Otra funcionalidad ligada a esto es poder obtener información sobre las minucias de una huella, como cantidad e información de las crestas, entre otros.

9.2.4. Gestión de Imágenes

Una huella digital es una imagen, y por ende es importante tener ciertos mecanismos básicos de manipulación de imágenes. La librería permite modificar la huella, como rotación, redimensión e inversión de colores, entre otras. Así también permite obtener información de las huellas respecto a su formato como imagen, como su calidad, dimensiones y píxeles, entre otras.

9.2.5. Arquitectura

Se plantea una arquitectura flexible que permite cubrir las diversas necesidades que el usuario pueda tener al trabajar con sistemas de huellas digitales. Como se muestra en la Figura 29 esta librería apunta a brindar al usuario soporte de distintos lenguajes y metodologías (como estructurado y orientado a objetos), así como la libre elección de scanners biométricos. Con este diseño, se permite la creación de aplicaciones nativas, desarrolladas en el mismo lenguaje de la librería, y por ende, más rápidas y con acceso directo al hardware que requiera la aplicación que se está desarrollando.

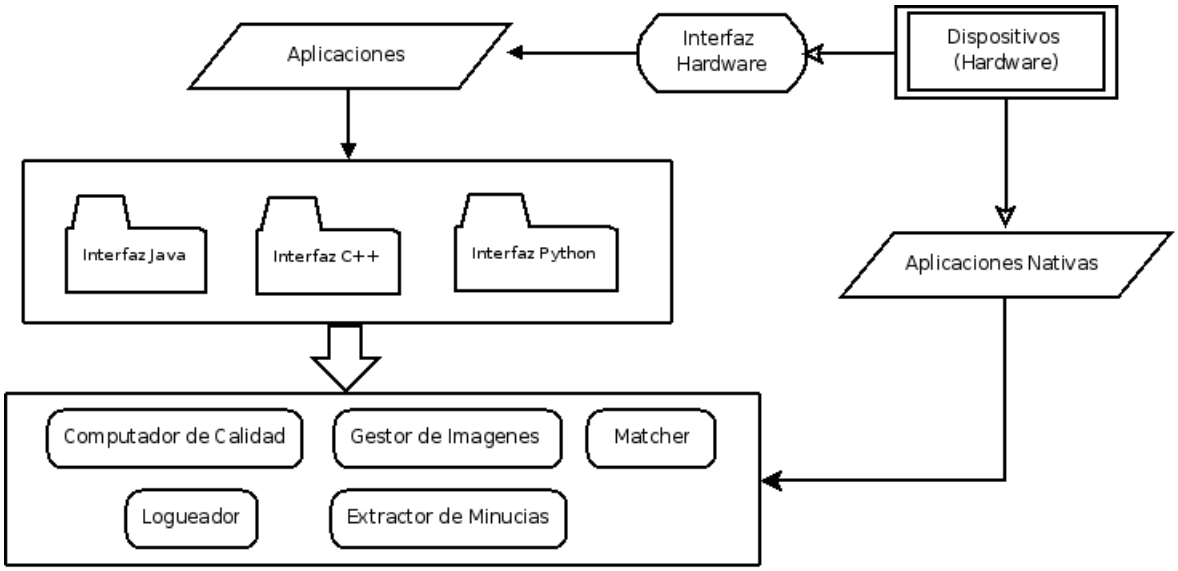

Figura 29: Arquitectura de la API

Por otra parte, también se pueden desarrollar aplicaciones en lenguajes externos, como Python y Java, a través de interfaces que sirven como puente comunicador entre el software y la librería. Esta funcionalidad permite extender el alcance del proyecto y ayuda al usuario a poder usar la metodología que más estime conveniente para su proyecto. Debido a que los drivers de scanners biométricos sólo pueden ser accesados a bajo nivel, debe implementarse separadamente una interfaz que comunique el dispositivo con el software externo.

Esta arquitectura también permite la combinación de lenguajes, donde pueden usarse mecanismos de bajo nivel para ciertas funciones de una aplicación y herramientas de un nivel más alto para otras tareas, como la creación de interfaces gráficas o de depuración.

En la Figura 30 se presenta un posible diagrama de clases para la construcción de interfaces para lenguajes orientados a objetos. Sólo es necesario guiarse por este diagrama y unir las clases a la librería por medio de software que haga de *puente* entre ambos, como SWIG. Distintos usuarios pueden reprenstar y organizar las clases como estime conveniente, este es solo una posible estructuración. Muchas herramientas abiertas tienen la posibilidad de construir interfaces para lenguajes distintos al nativo, ejemplo de esto pueden ser libGTK+, libSDL y libXML, todas parte del movimiento open source.

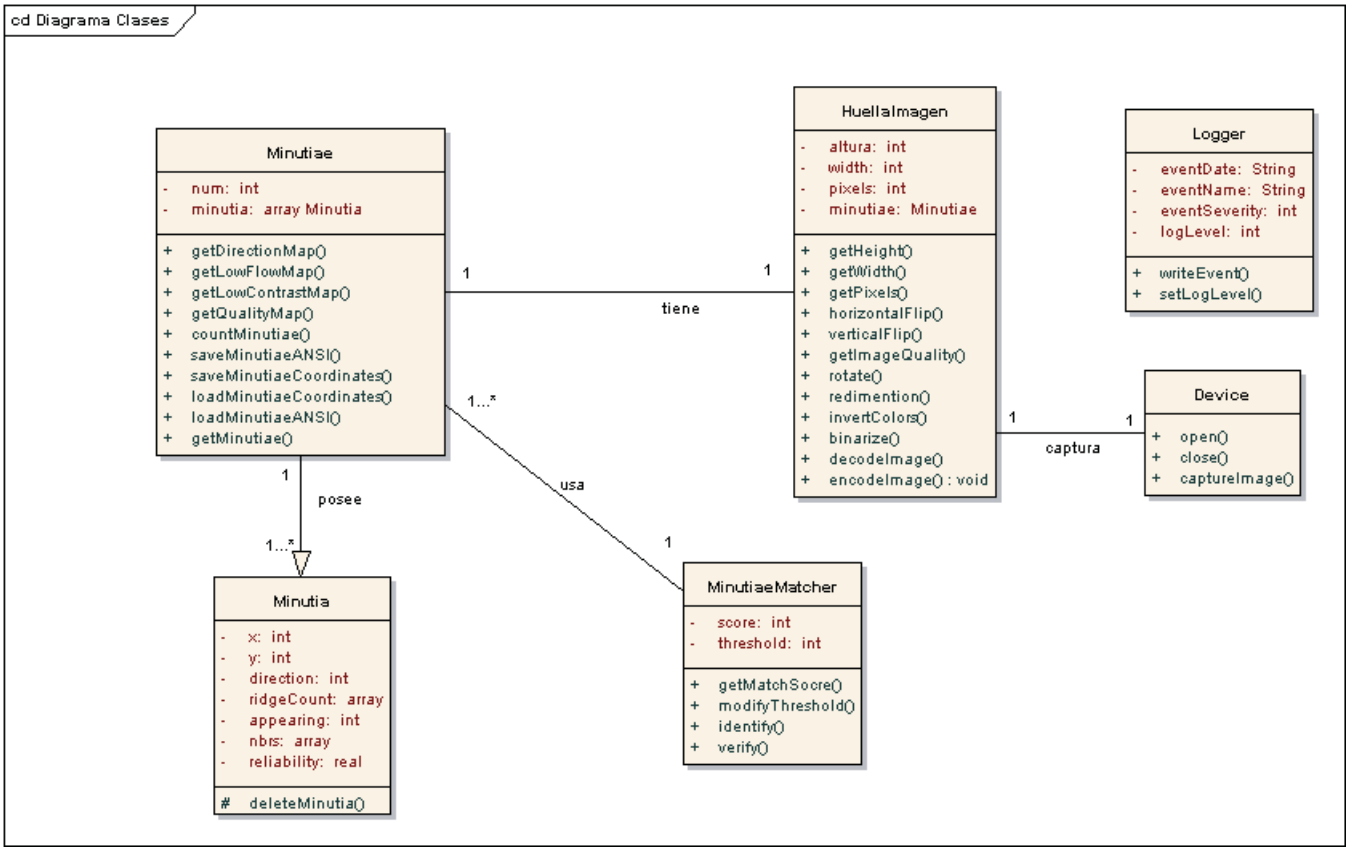

Figura 30: Diagrama de clases de una interfaz para lenguajes orientados a objetos

9.3. Almacenamiento de Datos

La recuperación y almacenamiento de huellas es una característica importante para todo sistema que use este proyecto. Se especificaron tres mecanismos para estas operaciones, cada uno aplicable a situaciones específicas. A continuación se describe el funcionamiento de cada mecanismo.

9.3.1. Compresión de Imágenes

Muchas veces se tienen imágenes de huellas digitales almacenadas en bases de datos multimedia o en bancos de imágenes, y se necesita una forma de pasar cada archivo a memoria para poder analizarlo biométricamente. También se necesita una forma de volver a almacenar los datos en un formato de imagen, una vez finalizado el proceso biométrico. Este componente tiene un decodificador que permite almacenar en memoria imágenes comprimidas con formatos estándares y que son comunes para almacenar huellas digitales, tales como JPEG y WSQ, entre otros. Existe también un codificador que vuelve el dato en memoria a una imagen, siguiendo los algoritmos de compresión definidos para el formato que se desee.

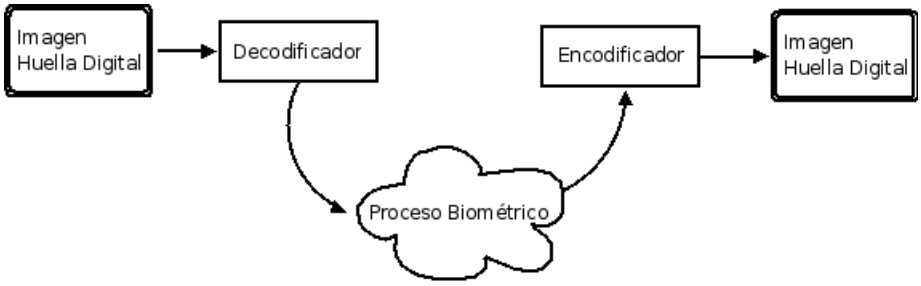

Figura 31: Ciclo de vida de ImgUtls

Si bien el decodificador y codificador se pueden usar juntos, es importante mencionar que son completamente independientes uno del otro. Por ejemplo, si la huella es capturada mediante un *scanner*, no se necesita el decodificador, pues queda inmediatamente en memoria. Lo mismo pasa con el codificador; la huella puede ser almacenada en una base de datos, sin ser guardada como imagen, por lo que no se usará.

9.3.2. Formato ANSI/NIST ITL 1-2000

La norma ANSI/NIST ITL 1-2000 'Data Format for the Interchange of Fingerprint, Facial, Scar Mark & Tatoo (SMT) Information' fue desarrollada para el intercambio de información y características biométricas entre distintas agencias del gobierno de los Estados Unidos[12]. Un archivo regido por esta norma define el contenido, formato y unidades de medida para el intercambio de huellas digitales (junto con otros tipos biométricos). A diferencia de las otras dos alternativas presentadas en este diseño, este formato es de texto plano (y no binario), consistente sólo en información sobre las minucias, como las coordenadas y grados de orientación, entre otros. Esto hace que sea mucho más liviano y portable, optimizando recursos y desempeño de los recursos computacionales. Como estos archivos son pasados directamente al matcher para la verificación o identificación, la complejidad y tiempo de procesamiento que se requiere son mucho menores.

9.3.3. Formato Binario

Algunas aplicaciones demandan capacidades de almacenamiento más flexibles, como almacenar las huellas en bases de datos, aplicar encriptación o compresión para enviarlos a través de una red. Este diseño provee la funcionalidad necesaria para acceder a una huella en *raw* (o en crudo) que representa una huella en memoria. Las imágenes raw son de una mejor calidad que aquellas con formato de imagen, debido a que no están comprimidas, manteniendo la mayor cantidad de detalles, lo que beneficia directamente el tratamiento de huellas digitales[8]. Así como también permiten un mejor control, pues las huellas en raw pueden ser manipuladas más finamente, como el balance de colores, saturación y brillo, entre otros. Esto lleva muchas veces a poder adaptar y mejorar la calidad. Este tipo de archivo también es el usado por los scanners al fotografiar un dedo y así capturar la huella de una persona.

La interfaz permite al desarrollador obtener una huella *raw* y almacenarla usando el mecanismo que estime conveniente; los datos pueden ser posteriormente recuperados y usados por el software para una verificación o identificación.

10. Implementación de la Librería

10.1. Estructura de la Librería

Como se ilustra en la Figura 32, se estructuraron los archivos y directorios en forma de árbol, divididos en las funcionalidades individuales de cada parte, como código fuente, archivos de configuración y pruebas, entre otros.

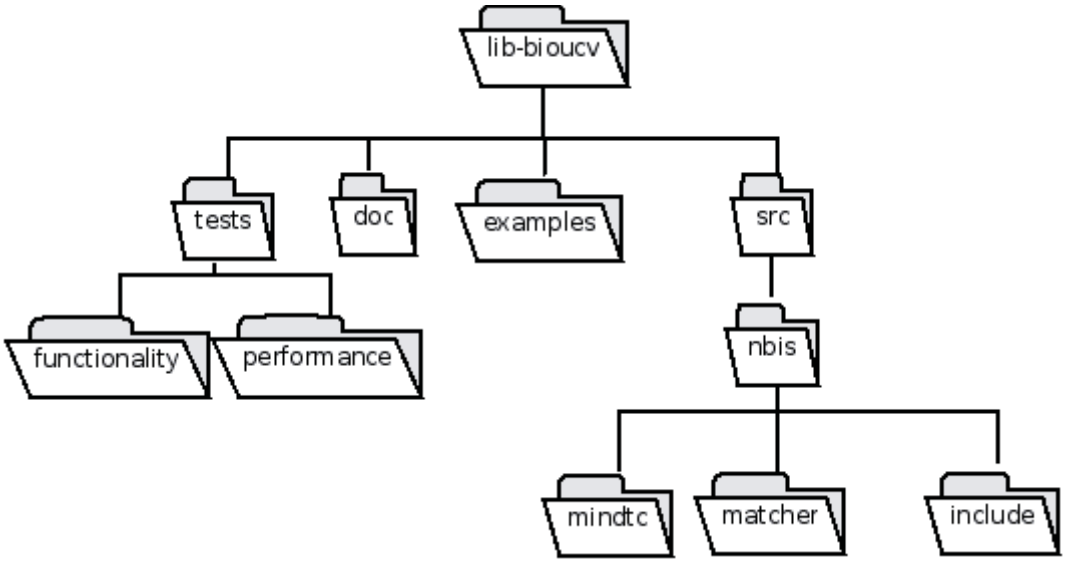

Figura 32: Estructuración de la librería

A continuación se describe el contenido de cada carpeta:

- lib-bioucv: La carpeta raíz, en donde se almacena la librería, contiene diversas subcarpetas y archivos de configuración para `autoconf` y `automake`.
  - tests: Contiene el código fuente de las distintas pruebas unitarias que se le hizo a las funciones de la librería.
    - functionality: Contiene las pruebas unitarias realizadas a las funciones.
    - performance: Contiene las pruebas de rendimiento de memoria realizadas a las funciones.
  - doc: Contiene el manual de referencia de la API de la librería.
  - examples: Contiene ejemplos sencillos para utilizar diversas funciones y estructuras de datos de la librería. Cada archivo tiene su propia documentación para compilarlo y ejecutarlo.
  - src: Contiene el código fuente del proyecto, y esta dividido en diversas subcarpetas, separadas por funcionalidad.
    - nbis: Código fuente del NIST Biometric Image Software, utilizado como la base biométrica de la librería.
      - matcher: Código fuente del comparador de huellas.

- ◇ mindctc: Código fuente del extractor y computador de minucias.
- ◇ include: Contiene los archivos de cabecera utilizados por el matcher y mindctc.

10.2. Herramientas

10.2.1. Lenguaje

Para la implementación de la librería se decidió usar el lenguaje de programación C (con su estándar ANSI), por las siguientes razones:

- Dado que la base de este proyecto está desarrollada en C, es más óptimo transformarla y extenderla manteniendo el mismo lenguaje en la que fue originalmente desarrollada. De lo contrario, pueden surgir problemas y el código puede volverse complicado, haciéndolo difícil de depurar y mantener en el tiempo.
- Muchos de los algoritmos de extracción de minucias y matching de huellas requieren una mayor velocidad en el cómputo de las instrucciones, y es por esto que es recomendable implementarlos en un lenguaje de más bajo nivel, como C.
- Facilita la integración con lenguajes de más alto nivel, como Java o Python. A partir de una librería desarrollada en C, es más fácil y rápido desarrollar interfaces para otros lenguajes que si fuera implementada con herramientas de alto nivel.
- Permite el acceso directo a los drivers de los dispositivos biométricos, facilitando la integración de distintos scanners.
- Permite que la librería sea utilizada con sistemas embedidos.

10.2.2. Dependencias

Este proyecto usa una serie de herramientas externas para su implementación. Cada una cumple un factor importante en la estructura de la librería:

- GCC: Se usó el compilador GNU Compiler Collection (gcc)[10] versión 4 para el desarrollo de este proyecto. GCC es un compilador portable, el cual corre sobre la gran mayoría de las arquitecturas computacionales existentes (x86, sparc y mips, entre otros) y es capaz de compilar programas escritos en varios lenguajes, como C, C++, Fortran y Java. Este compilador es incluido por defecto en la mayoría de las plataformas Unix, como Linux, BSD y OpenSolaris, lo que lo hace ideal para el desarrollo de este proyecto.
- MagicCore: Es una librería compuesta por interfaces de bajo nivel entre en lenguaje C y la librería de tratamiento de imágenes ImageMagick. Esta herramienta soporta más de 90 formatos de imágenes digitales como JPEG, PNG, TIFF y BPM, entre otros. El usar MagicCore ayuda al desarrollo de las funciones de procesamiento de imágenes que posee la librería.

- AutoTools: Esta herramienta permite configurar y compilar el proyecto para diferentes plataformas computacionales, manteniendo una uniformidad. Con AutoTools es posible comprobar que las dependencias del proyecto estén previamente instaladas antes de compilar, ahorrando tiempo y evitando errores al momento de compilar.
- LibTool: Esta herramienta provee una forma estándar para crear librerías (estáticas y dinámicas) de forma fácil.

10.3. Librería Dinámica Compartida

Como su título lo dice, este proyecto se basa en desarrollar una librería, lo que, a grandes rasgos, significa implementar una colección de archivos tipo objeto precompilados[11], los cuales son insertados en programas durante el proceso de *linking*, lo cual forma parte de la compilación. Una librería permite reutilizar software creado previamente, agilizando el proceso de desarrollo.

Existen diversos tipos de librerías, como estáticas, dinámicas y remotas, entre otros. Cada una tiene sus propias ventajas y desventajas, y para este proyecto, se optó por usar librerías dinámicas compartidas. Este tipo de librerías usa la extensión `.so`, lo que significa objeto compartido (*shared object*). Una de las principales ventajas de las librerías dinámicas es que permiten que los archivos ejecutables finales sean mucho más pequeños, pues se comparte una única copia de la librería entre todas las aplicaciones que lo utilicen simultáneamente. Esto se debe a que el sistema operativo usa parte de la memoria virtual para compartir la copia (en memoria física) asignando direcciones de memoria virtual para las distintas funciones de la librería que use cada aplicación.

Cada librería tiene su propia versión, para evitar problemas de compatibilidad con las aplicaciones que la usen. Entonces, si la librería es modificada, la versión actual que usa la aplicación no será la misma que tiene la nueva, por lo tanto no la usará y se quedará por defecto con la que esta usando actualmente. Para el caso de este proyecto, se denomina el archivo `libbioucv.so`, el cual, por motivos de simplicidad, es un enlace simbólico a la versión actual, que es `libbioucv.so.0` (cero indicando que es la primera versión).

10.3.1. Buenas Prácticas

Dado que este proyecto puede ser usado por un sin número de diferentes aplicaciones, hay que tener especial cuidado al diseñar e implementar la API y las funciones de uso interno, de manera que su mantención sea lo menos compleja posible. Deben tomarse algunas consideraciones, como buenas prácticas[6], al momento de implementar este proyecto. Sin esto, puede resultar que el proyecto se vuelva casi inmantenible y existan incompatibilidades entre diferentes versiones de la librería. Por ejemplo, si se necesita eliminar algún parámetro de una función que pertenece a la API, o agregar un campo a una estructura de datos, ¿qué impacto tiene esto en las aplicaciones que ya están usando la librería? ¿Se debe modificar? Esto puede ser costoso y complejo. Para mitigar estos riesgos, se decidió:

- Usar funciones específicas para obtener y setear variables, por ejemplo `get_variable()` y `set_variable()`.

- Usar un prefijo determinado para todas las funciones y tipos de datos que forman parte de la API de la librería. En este caso se usa `ucv_`, por ejemplo, `ucv_minutiae_t`. Esto ayuda a evitar posibles conflictos de nombres con otras partes de una aplicación.
- Usar sólo definiciones y declaraciones de funciones de la API en los archivos de cabecera (`archivo.h`) que el usuario puede ocupar. Es importante no mezclar funciones de la API y funciones internas dentro de un archivo cabecera que es visible al usuario, pues esto produce confusión encuanto a si las funciones están realmente disponibles o no para su uso.
- Asegurar que las funciones y variables propias del objeto no sean exportadas a la API. Esto se logra mediante el uso de `static`, y evita posibles problemas futuros. Esto también es ventajoso pues ayuda al compilador a optimizar la librería.

10.3.2. Compilación

Ya que este proyecto se implementa en forma de librería dinámica compartida, se deben tomar algunas consideraciones al momento de compilar el código fuente. A diferencia de aplicaciones normales (*stand alone*), una librería requiere usar ciertos parámetros especiales (o *flags*) para alterar el proceso normal de compilación.

Normalmente, al compilar una aplicación escrita en C, se efectúa una serie de pasos secuenciales para producir un archivo binario, o ejecutable:

i Preprocesamiento: este proceso expande las macros y aplica los archivos cabecera de la aplicación, por ejemplo los `#define` e `#include`.

ii Compilación: Transforma el código fuente (junto con los archivos preprocesados) al lenguaje ensamblador para la arquitectura específica (asm).

iii Ensamblamiento: Transforma el lenguaje ensamblador a código de máquina (1s y 0s), también conocido como archivos objetos.

iv Linking: Crea el ejecutable final en un archivo con binario, como ELF y COFF (Executable and Linker Format y Common Object File Format, respectivamente).

Para el proceso de compilación de este proyecto, además del compilador, se usó 'libtool' para la creación de la librería. Su proceso consiste en, primero, completar, para todos los archivos de código fuente, las tres primeras fases de una compilación normal, es decir, sólo llegar hasta la creación de los archivos objetos. Luego, crear archivos de texto plano con la ubicación de cada objeto creado. Una vez que se han compilado todos los archivos, se crea el archivo único compartido, para luego convertirlo en una librería propiamente tal. Este procedimiento se ilustra de la siguiente manera:

```
Para todo archivo con extensión '.c':
    Compilar el archivo en forma de objeto:
        gcc <opciones de compilación> -c -o <nombre_archivo>.o
    Crear el archivo con la ubicación de cada objeto:
```

```
        libtool <opciones> -o libbioucv_la-<nombre_archivo>.lo
Generar el archivo compartido a partir de los objetos:
    gcc -shared <opciones de compilación> archivo1.o archivo2.o
        ... archivoN.o -o libbioucv.so.<version>
```

Las opciones de compilación que se establecieron para cada archivos son las siguientes:

- `-Wall`: Esta opción habilita las opciones de advertencias más comúnmente usadas por el compilador, y se recomienda siempre utilizarla para evitar problemas semánticos y de rendimiento a la hora de correr la aplicación. Por defecto GCC no anuncia las advertencias.
- `-O2`: Esta opción habilita opciones de optimización realizadas por el compilador, como planificación de instrucciones. Esta opción es siempre recomendada, pues es la mejor alternativa costo vs peso para el archivo final.
- `-g`: Incluye información para depuradores (*debuggers*), como `GDB`.
- `-c`: Indica al compilador que debe crear un archivo objeto, en vez del ejecutable final.
- `-o`: Nombre del archivo de salida después de haber completado la compilación.

Estas opciones son algunas de las más comúnmente usadas en proyectos de software libre, y han sido, tradicionalmente, consideradas como buenas prácticas. Esta tarea se simplificó mucho gracias al uso de automake, el cual usa archivos denominados 'Makefile.am' para generar, a través de un script bash, el archivo Makefile que es el que finalmente se ejecuta.

10.4. Implementación de la API

Por diseño, se separó la librería en diferentes partes (o módulos) de acuerdo a la tarea específica que realiza cada una. Esto permite dividir el trabajo más fácilmente y mantenerlo ordenado para realizar cambios o entender la estructuración del proyecto. Existen elementos en la librería que son comunes a cada parte, como el caso de la información de la huella en memoria. Para esto se definió un `ucv_fingerprint_t`, el cual no es más que un puntero a un caracter sin signo: `unsigned char *`. Esta librería trabaja con huellas con hasta 8bits de nivel de gris (*grey scale*), lo que significa que deben tener valores entre 0 y 255, por lo cual se usa el concepto de caracteres sin signo, de lo contrario el rango de valores iría entre -128 y 127. Todas las funciones que pertenecen a la API se declaran como `PUBLIC_API`, la cual es una macro definida para una visibilidad normal, y si no se tiene, no se puede acceder a la función otra aplicación. Para esto se usan atributos especiales de GCC, como se ilustra a continuación:

```
#define PUBLIC_API __attribute__((visibility("default")))

PUBLIC_API void ucv_funcion()
{
        ...
}
```

10.5. Integración de NBIS

Al comenzar a implementar este proyecto se tenía una serie de archivos de código fuente correspondientes al NBIS, junto con el diseño de las interfaces de la API, las cuales son las que necesita el usuario para hacer uso de la librería.

Se decidió crear dos capas de abstracción internas. La primera, y de más bajo nivel, está constituida por parte del código del NBIS, la cual tiene los paquetes de comparación de huellas (bozorth) y el detector de minucias (mindtct). La siguiente capa consiste en aquellas funciones y estructuras de datos que el usuario puede utilizar como parte de la librería. Estas, en general, llaman a funciones de la capa más baja, haciendo uso de sus algoritmos y utilidades, como se ilustra en la Figura 33. Como se ve, una aplicación desarrollada por el usuario llama, por ejemplo, al comparador de huellas, mediante alguna función `ucv_matcher_`$< nombre >$ y esta, a su vez, llama a la función correspondiente del BOZORTH.

10.6. Gestor de Imágenes

El gestor de imágenes presenta un nivel de abstracción para el procesamiento de imágenes digitales necesarias en ambientes donde se trabaja con huellas digitales. Como se mencionó anteriormente, las imágenes soportadas por la librería son de 8bits de escala de gris. Este tipo de imágenes está constituida por 8 bits de información por pixel, y usa de 0 hasta 255 escalas de grises, en formato decimal (y 00000000 hasta 11111111 en binario), logrando hasta 256 distintos tonos de gris por píxel:

```
00000000: negro
```

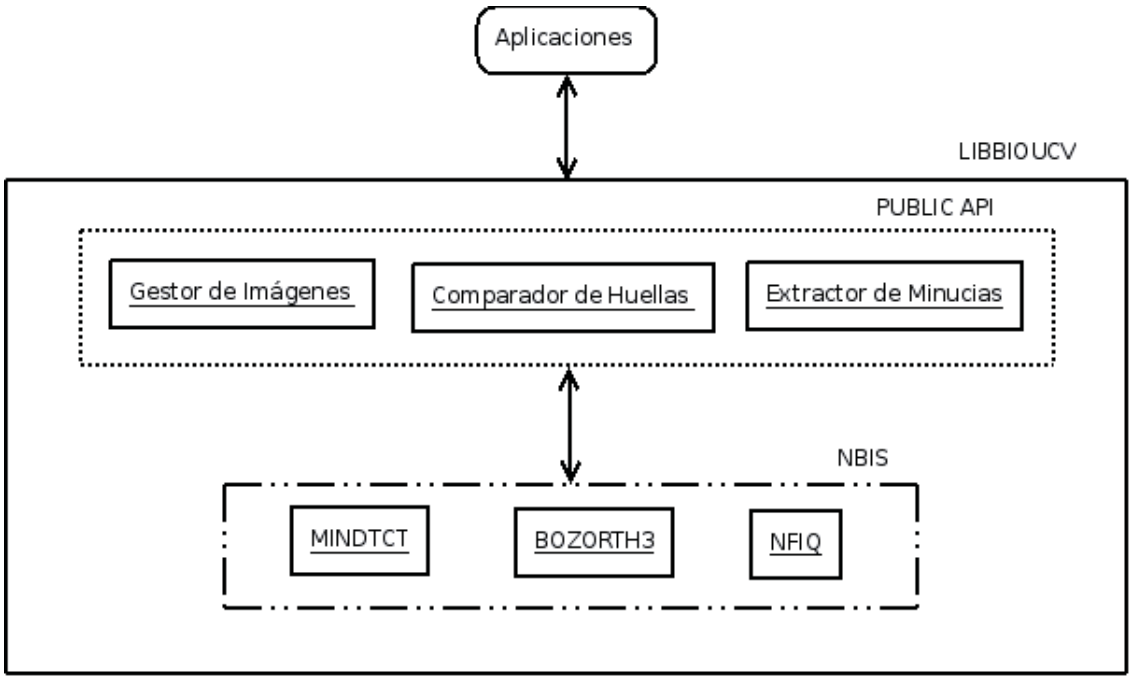

Figura 33: Estructura jerárquica de la librería

```
...
11111111: blanco
```

Cada imagen en la librería está representada por `ucv_image_t`, una estructura que contiene la información mínima que se requiere para capturar y modificar una imagen. Está compuesta por las dimensiones en píxeles (alto y ancho) al igual que la información sin procesar (*raw data*). Al decidir crear un gestor de imágenes se pensó en un mecanismo de entrada y salida conveniente para los usuarios, en donde se pueda alimentar la librería con una imagen de entrada, poder manipularla, ingresarla al extractor de minucias, etc. para luego almacenarla nuevamente en el sistema de archivos bajo algún formato de compresión.
Se desarrollaron diversas funciones para gestionar una imagen, y se utilizó ImageMagick para muchas de estas, como por ejemplo, tomar un archivo y obtener la información *raw*, asi como la escala de gris y las dimensiones. La función `ucv_img_to_data()` toma una imagen (dado por la ubicación en el sistema de archivos) de formato TIF, PGM, JPG o PNG (entre las más comunes para el manejo de huellas digitales) y obtiene la información necesaria para generar un `ucv_image_t` y luego la retorna. Esta una una de las funciones de entrada de la librería, ya que una vez que se usa, se puede, virtualmente, hacer todo el proceso biométrico (manipular la imagen → obtener las minucias → compararlo con otras huellas). Como parte de la familia de funciones de manipulación o procesamiento de imágenes digitales, se implementaron las siguientes funciones:

- `ucv_img_vflip()`: Gira la imagen verticalmente (rotarla 180 grados). El algoritmo se basa en ir tomando las ultimas filas e intercambiarlos por la primera. Por ejemplo, en una imagen de 374x388, reemplaza la fila 388 por la 1ra, luego la 387 por la 2da, luego la 386 por la 3ra y así hasta recorrer toda la imagen.

- `ucv_img_hflip()`: Gira horizontalmente la imagen usando un algoritmo similar al de la inversión vertical. Itera por la imagen columna a columna reemplazando cada extremo. Por ejemplo, en una imagen de 374x388, reemplaza la columna 374 por la 1ra, luego la 373 por la 2da, luego la 372 por la 3ra, y así hasta recorrer toda la imagen.
- `ucv_img_invert_colors()`: Invierte los colores de la imagen. El algoritmo utilizado simplemente itera por la imagen, píxel por píxel, restando 255 a cada uno (o `0xff` en hexadecimal)[8]. Si se utilizara un histograma para ver las intensidades de gris por cada píxel de una huella original y aquella invertida, se observaría la inversión a nivel de pixelaje, como se ilustra en la Figura 34.

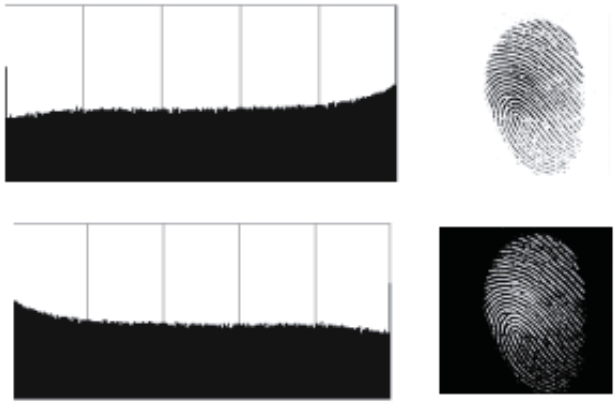

Figura 34: Comparación de histogramas de una huella original y de una con sus colores invertidos

- `ucv_img_binarize()`: Binariza la imagen, usando algoritmos del MINDTC del NBIS.

La Figura 35 ilustra cómo funcionan algunas de las funciones de manipulación de imágenes.

Figura 35: Ejemplo de algunas funciones del manipulación de imágenes

La primera imagen es la entrada para el uso de la librería, se pasa un archivo TIF con una huella digital y se crea la estructura usando la función `ucv_img_to_data()`. Luego, una vez en memoria, se procede a invertir la imagen llamando a `ucv_img_vflip()` para posteriormente invertirle los colores usando `ucv_img_invert_colors()`. Es importante mencionar que en cualquier punto se puede llamar a alguna función para exportar la imagen en memoria a un archivo de compresión, como JPG o PNG.

También es necesario poseer mecanismos para obtener información acerca de la imagen. Para esto se crearon: `ucv_img_get_width()` que retorna la anchura de la imagen, `ucv_img_get_height()` que retorna la altura de la imagen y `ucv_img_get_data()` que retorna la huella sin procesar (obtenida desde el archivo de entrada).

Dado que se tiene una forma para ingresar una huella a la librería, también se necesita una forma para exportarla a un archivo de imagen. Para esto se crearon las siguientes funciones: `ucv_img_data2pgm()`, `ucv_img_data2jpg()`, `ucv_img_data2tif()` y `ucv_img_data2png()`. Cada una de estas funciones toma una estructura `ucv_image_t` y crea un archivo usando la compresión solicitada con la huella digital. La primera usa la especificación de PGM, la cual consta de escribir a la primera línea de un archivo un cierto *string* con las dimensiones de la imagen y posteriormente, en las siguientes líneas, escribir la información *raw* de la imagen un byte a la vez hasta completar *altura x anchura* bytes. Las otras tres funciones usan ImageMagick para comprimir la huella para cada formato (en realidad soporta más de 90 tipos de archivos de compresión[25], pero sólo se trabaja con algunos para este proyecto). Estas funciones son realmente *wrappers* (o macros) de la función `ucv_img_data2img_format()`, la cual llama una rutina de ImageMagick para convertir los datos a archivos comprimidos. Esta relación se ilustra en la Figura 36. Internamente ImageMagick usa distintas librerías independientes de imágenes digitales, como para compresión y descompresión utiliza `libZ`, así como utilidades específicas para el tipo de archivo que se está utilizando, como el caso de `libJPEG`, `libTIF` y `libPNG`.

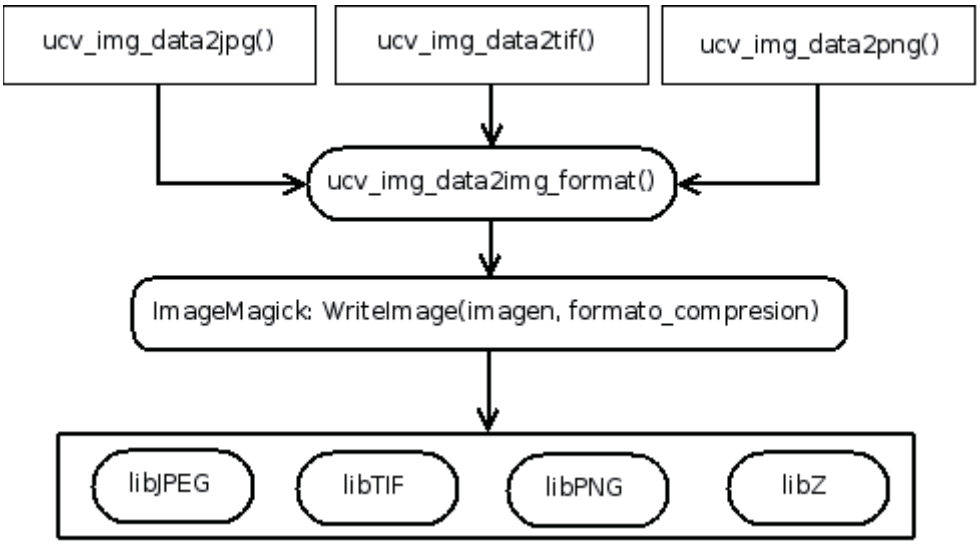

Figura 36: Funcionamiento interno de las funciones de exportación de imágenes

Una vez que se ha terminado de trabajar con la imagen, es importante liberar la memoria utilizada (como buena práctica de programación) y destruir todo dato asociado a ella. Para esto, se llama a `ucv_img_free()`. Si el usuario no usa esta función, tendrá espacio en memoria sin utilizar, cosa que puede resultar en drásticas disminuciones de rendimiento de un sistema, sobretodo si se está trabajando con muchas imágenes.

10.7. Extractor de Minucias

El detector y extractor de minucias es una de las partes fundamentales de la librería, sin ella no se podría realizar el matching. El rendimiento del matcher está directamente ligado a la eficiencia del extractor de minucias. Esta parte de la librería está estrechamente ligada al código fuente del MINDTCT del NBIS, pues usa algoritmos y estructuras de datos necesarios para capturar la información. Para el extractor de minucias se estableció el tipo de dato `ucv_minutiae_t`, el cual es una estructura que almacena información respecto a cada minucia detectada a una huella, junto con la cantidad de minucias detectadas. El tipo de dato `ucv_minutia_t` (singular), que es incluido como un arreglo de punteros a la estructura `ucv_minutiae_t` (plural), y se refiere específicamente a una minucia, donde se incluye información como las coordenadas (x, y), la dirección, el tipo de minucias (si es un fin de una cresta o una bifurcación) y la cantidad de vecinos, entre otras. Lógicamente se puede ver la relación en la Figura 37.

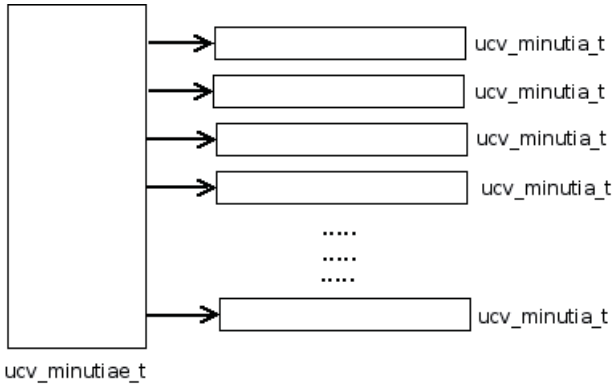

Figura 37: Relación entre estructuras que contienen las minucias detectadas a una huella digital

El tipo de dato `ucv_minutiae_t` que representa todas las minucias de una huella, se define como sigue:

```
typedef struct minutiae {
        int num; /* cantidad de minucias detectadas en la huella */
        ucv_minutia_t **list; /* matriz de punteros a cada minucia de la huella */
} ucv_minutiae_t;
```

El tipo de dato `ucv_minutia_t`, que representa una sola minucia dentro de una huella, se define como sigue:

```
typedef struct minutia {
         int x;
         int y;
         int direction;
         int type;
         int appearing;
         int feature_id;
         int *nbrs;
```

```
        int *ridge_counts;
        int num_nbrs;
} ucv_minutia_t;
```

- `x`: Coordenada (dentro de un plano cartesiano) x de la minucia.
- `y`: Coordenada (dentro de un plano cartesiano) y de la minucia.
- `direction`: Dirección u orientación de la minucia. Se representa en grados (de 0 a 360), con cero grado apuntando horizontalmente y a la derecha, e incrementándose hacia la izquierda. La orientación de terminaciones de crestas se determina al medir el ángulo entre el eje horizontal y la línea que comienza en el punto de la minucia y continúa hacia el medio de la cresta. La orientación de una bifurcación se determina midiendo el ángulo entre el eje horizontal y la línea que comienza en el punto de la minucia y continua hacia el medio de la bifurcación.
- `type`: El tipo de minucia, NBIS sólo trabaja con término de crestas (`RIDGE_ENDING`) y bifurcaciones (`BIFURACTION`).
- `appearing`: Si la minucia esta apareciendo (`#define APPEARING 1`) o desapareciendo (`#define DISAPPEARING 0`). Este parámetro indica la dirección donde nace o resalta la minucia hacia el patrón (ver Figura 38). si es de izquierda a derecha, entonces se considera como apareciendo; en otro caso se designa como desapareciendo. Estas características están predefinidas en los patrones de pixeles de 2x3 a buscar durante el proceso de detección de minucias.
- `feature_id`: Índice en la estructura de los patrones de minucias (véase Figura 38). Dado que se tienen 10 posibles patrones de píxeles, y puede repetirse el tipo, y si aparece o desaparece, este campo diferencia un tipo de minucia con otra, haciéndola única. Por ejemplo, una minucia puede ser BIFURCATION y DISAPPEARING, junto con el `feature_id` se especifíca la minucia individual, y se diferencia de todas las demás. Puede tomar los siguientes valores: $\{0, 1\}$ o $\{1, 0\}$.
- `nbrs`: Puntero a los vecinos de la minucia.
- `ridge_counts`: Cantidad de crestas que atraviesan una línea recta (imaginaria) entre la minucia con su minucia vecina más cercana.
- `num_nbrs`: Cantidad de vecinos de la minucia, la cantidad máxima está dada por `MAX_NBRS` (`#define MAX_NBRS 5`).

La función que detecta y captura las minucias de una huella es `ucv_minutiae_get()`, la cual toma una huella en memoria (junto con sus respectivas dimensiones) y crea una estructura `ucv_minutiae_t`, la cual es devuelta por la librería. De forma interna, se realiza diversos pasos en un orden específico, estos se describen por lo siguiente:

1. Inicialización: Se comienza por calcular la cantidad de pixeles a agregar a la imagen, de forma que se pueda dividir en bloques perfectos para así trabajarla como un cuadrado - esto lo realiza la función `get_max_padding()`. Posteriormente se pasa a rellenar la imagen con estos pixeles, llamando a

`pad_uchar_image()`, la cual hace una copia de la imagen a una más grande y agrega los píxeles necesarios alrededor (o del perímetro si se considera que la imagen es un cuadrado).

2. Generación de Mapas: El siguiente paso consiste en generar los mapas de dirección, de flujos y contrastes, entre otros. Para esto, se usa la función `gen_image_maps()`, la cual computa los mapas. Se comienza por generar el mapa de dirección, el que representa el flujo dominante de crestas para cada bloque. Cada bloque válido tiene un valor positivo (incluyendo cero), y a cada inválido se asigna -1. Posteriormente, se obtienen los mapas de bajo contraste y bajo flujo. En ambos casos aquellos bloques con bajo contraste y flujo son considerados como bloques inválidos en el mapa de direcciones.

   Luego se eliminan las direcciones que son inconsistentes con los bloques vecinos (`remove_icon_dirs()`) y se tratan aquellos bloques donde se detectan cambios bruscos con las direcciones de los vecinos (`smooth_direction_map()`). Después se asignan los bloques que componen el margen de la imagen (los píxeles de relleno) como inválidos usando la función `set_margin_blocks()`. Finalmente, se crea el mapa de alta curvatura a partir del mapa de direcciones usando `gen_high_curve_map()`.

3. Binarización: Ya con los mapas creados, el siguiente paso es binarizar la imagen. Para esto se llama a `binarize_image()`, la cual recorre el mapa de dirección de la huella, bloque por bloque, buscando aquellas inválidas, la cual asigna como pixel blanco (`#define WHITE_PIXEL 255`). Si el bloque es válido, entonces se usa la función `dirbinarize()`, la cual determina el valor binario de un píxel de escala gris basado en la dirección de las crestas. Este procedimiento consiste en ubicar el centro de la malla sobre el píxel de interés y luego se gira para que las filas estén paralelas y las columnas perpendiculares a la dirección de la cresta, sumando las intensidades de gris. También se suman las intensidades de gris de toda la malla, es decir, rotándola 360 grados. Para calcular el valor binario del píxel, se multiplica la primera suma por la cantidad de filas en la malla y luego se compara con la segunda suma; si es menor, entonces se asigna el píxel negro (`#define BLACK_PIXEL 0`). A continuación se ilustra parte del código de este algoritmo:

```
/* calcular el centro (0 grados de orientacion) de la fila en la malla */
dcy = (dirbingrids->grid_h-1)/(double)2.0;
....
/* para cada fila en la malla... */
for(gy = 0; gy < dirbingrids->grid_h; gy++){
   /* inicializar suma de pixeles de las filas a 0 */
   rsum = 0;
   /* para cada columna en la malla ... */
   for(gx = 0; gx < dirbingrids->grid_w; gx++){
      /* acomular el proximo pixel de la fila rotada en la malla */
      rsum += *(pptr+grid[gi]);
      /* Bump grid's pixel offset index. */
      gi++;
   }
   /* acumular la suma de filas en la suma de pixeles de la malla */
```

```
      gsum += rsum;
      /* si la actual fila es la fila del centro de la malla,
         entonces almacenar la suma de filas separadamente */
      if(gy == cy)
         csum = rsum;
   }

   if((csum * dirbingrids->grid_h) < gsum)
      /* setear pixel binario a negro */
      return(BLACK_PIXEL);
   else
      /* de lo contrario setear el pixel binario a blanco */
      return(WHITE_PIXEL);
```

4. Detección: Una vez que se tiene la huella binarizada, se usa `detect_minutiae()` para recorrer la imagen, bloque por bloque, buscando aquellos válidos y detectar posibles minucias. Se comienza por buscar minucias horizontalmente en la imagen (`scan4minutiae_horizontally()`), la cual escanea toda la huella binaria de forma horizontal, detectado posibles puntos de minucia, los cuales tienen una orientación, por naturaleza, vertical. Los posibles candidatos a minucias se detectan obteniendo dos píxeles y comparándolos con los patrones predeterminados para ver si son bifurcaciones o términos de crestas. La siguiente estructura define todos los posibles patrones.

```
 typedef struct feature_pattern{
   int type; /* si es una bifurcacion o fin de una cresta */
   int appearing; /* si aparece o desaparece de acuerdo al patrón */
   int first[2]; /* puede ser: {0, 0};{0, 1};{1, 0};{1, 1} */
   int second[2]; /* puede ser: {0, 0};{0, 1};{1, 0};{1, 1} */
   int third[2]; /* puede ser: {0, 0};{0, 1};{1, 0};{1, 1} */
 } FEATURE_PATTERN;
```

   Se puede ver el uso de esta estructura al asociarlo a los patrones de búsqueda de minucias, como se ilustra en la siguiente Figura 38.

   Si se encuentra un posible candidato, entonces se usa `process_horizontal_scan_minutia()` para detectar si realmente será considerada como una minucia válida. Esta función toma el punto de la minucia y determina su dirección y la agrega a la lista de minucias de la huella. Para esto se asegura que la minucia no esté en un bloque inválido:

```
/* si el punto de la minucia está en un bloque cuya direccion es invalida ... */
 if(dmapval == INVALID_DIR)
    /* entonces, ignorar el punto */
    return(IGNORE);
```

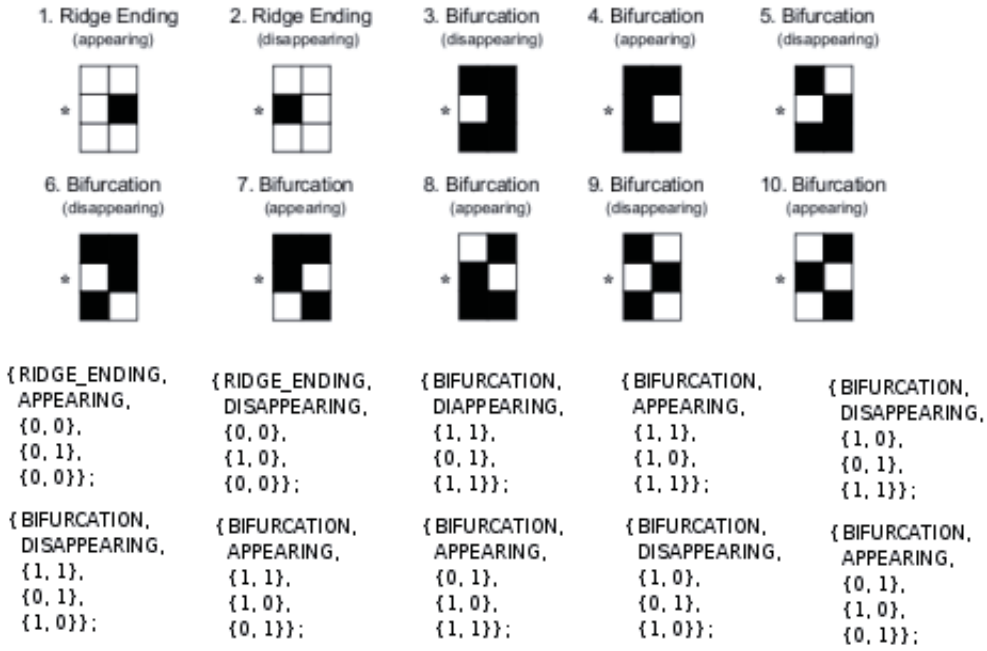

Figura 38: Secuencia de pasos para desarrollar cada función

El siguiente paso consiste en crear la minucia con los valores determinados durante el algoritmo, usando la función `create_minutia()`, que simplemente crea una estructura (`ucv_minutia_t`) y asigna los atributos correspondientes.

Después de la detección de las minucias de la huella, se tiene que eliminar las minucias falsa para no influenciar el proceso de comparación (*matching*). Esto lo realiza `remove_false_minutia()`. Esta función realiza una serie de pruebas sobre la estructura que contiene las minucias detectadas por el proceso anterior, entre las que se incluyen:

- Eliminar islas y lagos - `remove_islands_and_lakes()`: Esta tarea consiste en tomar pares de minucias y eliminar lagos (un pixel blanco alrededor de varios píxeles negros) e islas (un píxel negro alrededor de varios píxeles blancos).
- Eliminar agujeros - `remove_holes()`: Se toma una minucia y se buscan pequeños vacíos dentro de una cresta las cuales están dentro de un perímetros de 15 píxeles.
- Eliminar puntos cercanos a bloques inválidos - `remove_pointing_invblock()`: Se recorre la estructura de las minucias comparando cada minucia para determinar si están a 6 píxeles de algún bloque con dirección inválida.
- Eliminar ganchos - `remove_hooks()`: Este procedimiento detecta aquellas minucias en el lado de una cresta o un valle. Un gancho nace desde una cresta donde no alcanza a bifurcarse, queda simplemente como una pequeña cresta conectada con otra cresta de mayor tamaño. Al igual que en el caso de las islas, para determinar un gancho, la distancia entre ambas minucias debe ser de hasta 16 píxeles, y en direcciones de flujo opuestas. Otro factor para determinar un gancho

es que las minucias de interés sean del mismo tipo, es decir, ambas deben ser terminaciones o bifurcaciones.

5. Suma de Crestas: El paso final consiste en sumar las crestas de las minucias usando `count_minutiae_ridges()`. Esta función toma una lista de minucias, y para cada una, determina los vecinos más cercanos y cuenta la cantidad de crestas entre la minucia y cada uno de los vecinos. Para buscar los vecinos de una minucia, se usa la función `find_neighbors()`, la cual toma el punto, ve a qué bloque pertenece, y busca las crestas de los bloques adyacentes. Se puede ver parte del código del algoritmo para sumar las crestas como sigue:

```
/* buscar todos los vecinos y almacenar la informacion en nbr_list */
if((ret = find_neighbors(&nbr_list, &nnbrs, lfsparms->max_nbrs,
                         first, minutiae))){
   free(nbr_list);
   return(ret);
}

/* si no se encontraron vecinos ... */
if(nnbrs == 0){
   /* entonces terminar inmediatamente el proceso */
   return(0);
}

/* contar las crestas entre la actual y sus vecinos */
nbr_nridges = (int *)malloc(nnbrs * sizeof(int));
if(nbr_nridges == (int *)NULL){
   free(nbr_list);
   fprintf(stderr, "ERROR : count_minutia_ridges : malloc : nbr_nridges\n");
   return(-450);
}

/* Por cada vecino encontrado ... */
for(i = 0; i < nnbrs; i++){
   /* contar las crestas entre la minucia y sus vecinos */
   ret = ridge_count(first, nbr_list[i], minutiae, bdata, iw, ih, lfsparms);
   /* caso de error ... */
   if(ret < 0){
      /* liberar memoria usada hasta este momento */
      free(nbr_list);
      free(nbr_nridges);
      return(ret);
   }
```

```
        /* de lo contrario, la suma de crestas fue exitosa, por ende guardarla */
        nbr_nridges[i] = ret;
    }
```

Una vez que se han detectado y extraído las minucias de una huella, y se tiene un `ucv_minutiae_t`, se pueden realizar diversas funcionalidades que incluyen esta parte de la librería. Para obtener la cantidad de minucias de la huella se invoca a `ucv_minutiae_get_amount()`, el cual lee la estructura y retorna el número correspondiente (mayor o igual a cero). Así también se puede determinar si una minucia es una bifurcación o el término de una cresta usando `ucv_get_minutiae_type()`.

Existe la opción de almacenar las minucias en un archivo de texto plano, la cual disminuye drásticamente la cantidad de recursos a usar, si se compara con usar una imagen. La función `ucv_minutiae_save_to_file()` recorre la estructura y escribe la información, de la siguiente manera:

```
<cantidad_minucias_detectadas> <cantidad_maxima_de_minucias>
<coordenada_x> <coordenada_y> <ex> <ey> <direccion> <fiabilidad> <tipo> ... <cantidad_vecinos>
```

Para leer un archivo con este formato, se usa `ucv_minutiae_load_from_file()` la cual retorna un `ucv_minutiae_t` que se crea leyendo cada línea (a partir de la segunda, pues la primera es sólo información general de todas las minucias de la huella) y va creando una estructura de una minucia por cada línea leída.

Para liberar la memoria utilizada durante el proceso de detección y extracción de minucias, se debe utilizar `ucv_minutiae_free()`, la cual libera cada minucia dentro de la estructura.

10.8. Matcher

Esta parte de la librería presenta el comparador de dos o más huellas digitales basados en sus minucias. Cuando se comparan dos huellas de forma $1:1$ se realiza una verificación, y cuando se compara una huella con una serie de otras huellas de forma $1:N$ se realiza una verificación. Este proyecto usa como base de verificación el algoritmo del FBI denominado `BOZORTH` (la cual fue integrada al proyecto NBIS), y a partir de este se implementó el proceso de identificación. Para ambos procesos la librería presenta `ucv_match_verify()` y `ucv_match_identify()`. La primera función recibe un par de minucias de entrada para realizar el matching uno a uno y devuelve el resultado de la comparación representada por el tipo de dato `ucv_match_result_t` la que puede ser `MATCH_FAIL: 0` o `MATCH_SUCCESS: 1`. La segunda función toma una estructura de minucias a verificar y un *pool* (grupo) de minucias para compararlas, y hace uso del mecanismo de identificación n veces (donde n es la cantidad de minucias pertenecientes al grupo la cual se quiere comparar).

Internamente, el matching usa el NBIS para tomar las minucias y producir un score, la cual, posteriormente se evalúa de acuerdo a un umbral para decidir si pertenecen a una misma huella o no. Este algoritmo usa un máximo de 200 minucias, pero para efectos prácticos, tomas solo 150 de estas (en una huella de excelente calidad es posible llegar hasta 90, por lo que no afecta en el rendimiento). Sólo se basa en las coordenadas

geométricas de la ubicación de la minucia, junto a su grado de orientación dentro de la imagen, definiendo la tripleta: (x, y, θ). Esta información es representada por la siguiente estructura:

```
#define MIN_BOZORTH_MINUTIAE 0
#define MAX_BOZORTH_MINUTIAE 200
#define DEFAULT_BOZORTH_MINUTIAE 150

struct xyt_struct {
         int nrows;
         int xcol[MAX_BOZORTH_MINUTIAE];
         int ycol[MAX_BOZORTH_MINUTIAE];
         int thetacol[MAX_BOZORTH_MINUTIAE];
};
```

Como se ve, esta estructura tiene todas las coordenadas x (`xcol`), y (`ycol`) y el grado θ (`thetacol`) de las minucias de la huella, junto con la cantidad de minucias detectadas (`nrows`). Para convertir una estructura de minucias `ucv_minutiae_t` a `xyt_struct` se usa `minutiae2xyt()`, la que itera minucia por minucia, obteniendo las coordenadas y grado de orientación para almacenarlos ordenadamente en una estructura `xyt_struct`, la cual es retornada por la función. Como es de suponer, este paso es necesario para ambas minucias a comparar. Posteriormente se comienza a computar el resultado de la comparación de acuerdo al siguiente algoritmo:

10.8.1. Tablas de Comparación

El primer paso consiste en analizar y computar las distancias y ángulos de cada minucia con las demás dentro de una huella. Estas medidas son almacenadas en una **tabla de comparación de minucias**. Para esto se usa la función `bz_comp()`, la cual toma la cantidad de minucias junto con las coordenadas geométricas y grados de orientación de cada una. Itera por cada minucia obteniendo la distancia entra esa y todas las otras minucias de la huella. El siguiente fragmento de código muestra que se está comparando una minucia k con una j, y la distancia entre ambas se obtiene al sumar el cuadrado de las pendientes:

```
for ( k = 0; k < npoints - 1; k++ ) { /* minucia k */
        for ( j = k + 1; j < npoints; j++ ) { /* minucia j */
                dx = xcol[j] - xcol[k];
                dy = ycol[j] - ycol[k];
                distance = SQUARED(dx) + SQUARED(dy);
        ....
```

Luego, para medir la rotación, se calcula el ángulo entre la orientación de cada minucia y la línea recta que *une* cada minucia. De esta manera, estos ángulos se mantienen relativamente constantes, sin importar cuánta rotación exista. Este ángulo (llámese θ_{kj}) de la línea entre las minucias k y j se obtiene calculando el arco tangente de la pendiente de la línea:

```
                /* minucias k, j tienen la misma coordenada x */
                if ( dx == 0 ) theta_kj = 90;
                else {
                        double dz = ( 180.0 / PI ) * atanf((float)dy /
                                                        (float)dx );
                        if ( dz < 0.0 ) dz -= 0.5;
                        else dz += 0.5;
                        theta_kj = (int) dz;
                }
....
```

La Figura 39 ilustra dos ángulos más que se requieren para este algoritmo: β_k y β_j, los cuales son calculados en relación a la linea que une las minucias. Se incorpora el ángulo θ_{kj} en cada orientación t de la minucia:

```
beta_k = theta_kj - thetacol[k];
beta_k = IANGLE180(beta_k);
beta_j = theta_kj - thetacol[j] + 180;
beta_j = IANGLE180(beta_j);
```

Es importante mencionar que las orientaciones están sujetas al intervalo]-180, 180], con 0 grados apuntando horizontalmente a la derecha y aumentando los grados en dirección contraria a las manecillas del reloj. Esto lo realiza la macro `IANGLE180`, la cual simplemente agrega o resta 360 grados si el ángulo es menor o mayor a 180, respectivamente.

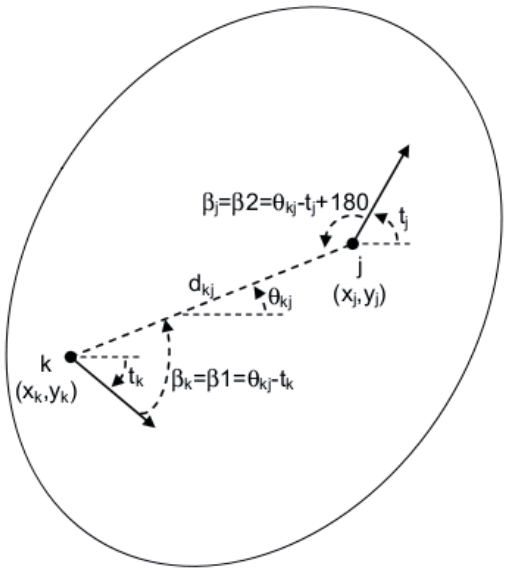

Figura 39: Análisis de dos minucias usando el algoritmo Bozorth

Finalmente, para cada par de minucias, se ingresa en la tabla la sectupla:

$$\{d_{kj}, min(\beta_k, \beta_j), max(\beta_k, \beta_j), k, j, \theta_{kj}\} \tag{9}$$

Las entradas almacenadas en la tablas de comparación son ordenadas en base a la distancia entre cada par de minucias d_{kj} en un orden de mayor a menor. Se construye una tabla para cada huella a comparar, por lo que en un matching $1-1$ existirán dos tablas.

10.8.2. Tabla de Compatibilidad

El siguiente paso consiste en buscar las entradas *compatibles* entre las dos tablas descritas anteriormente, y marcarlas como candidatas de minucias iguales al ingresarlas a una tabla única, denominada **tabla de compatibilidad de huellas**. Para esto se utiliza la función `bz_match()`, la cual realiza dos pruebas:

1. Distancias: Si se tienen dos tablas de comparación para dos minucias a verificar, P y G respectivamente, esta prueba consiste en verificar que las distancias en las entradas de la tablas estén en cierto rango de tolerancia T_d. Para esto se toma cada distancia en la tabla P y se la compara con cada distancia en la G, calculando la diferencias entre ambas generando un delta: $\triangle_{distancia}(distancia(P_m), distancia(G_n))$ (donde m y n son la m-ésima y n-ésima entrada o índice en cada tabla. Luego se prueba esta diferencia el rango de tolerancia:

```
#define TK 0.05

        /* para cada entrada en la tabla P ... */
        for ( k = 1; k < probe_ptrlist_len; k++ ) {
                /* almacenar la distancia en la variable ss */
                ss = scolpt[k-1];

                /* para cada entrada en la tabla G ... */
                for ( j = st; j <= gallery_ptrlist_len; j++ ) {
                        /* almacenar la distancia en la variable ff */
                        ff = fcolpt[j-1];
                        /* calcular el la direfencia (delta) de las distancias */
                        dz = *ff - *ss;

                        /* calcular la tolerancia en funcion de la
                           suma de las distancias en cada tabla */
                        fi = ( 2.0 * TK ) * ( *ff + *ss );

                        /* comparar el cuadrado (por los negativos)
                           del delta y de la tolerancia */
                        if ( SQUARED(dz) > SQUARED(fi) ) {
```

```
                /* si no se está dentro del límite
                   de la tolerancia, pasar a la
                   siguiente iteracion en j */
                   continue;
}
else { /* continuar con las otras pruebas de
          los ángulos*/
```

Como se ilustra en el fragmento de código, el algoritmo itera por cada entrada en la tabla P y G calculando las diferencias en las distancias y comparándolas con la tolerancia. Esta tolerancia está definida en función de las distancias tratadas en ese momento y en el factor `TK` de 0.05. Si está dentro de la tolerancia, entonces se proceden con las pruebas descritas a continuación. De lo contrario, se pasa a la siguiente iteración en la tabla G.

2. Ángulos: Este paso es muy similar a las comparaciones de las distancias, pero respecto a los dos ángulos (beta) que hay en cada entrada de cada tabla. Se comienza calculando la diferencia entre el primer beta de cada tabla generando un delta $\triangle_\beta 1$, el cual debe estar dentro de una tolerancia T_β. Si es que se pasa la prueba, se continúa con la iteración actual y se pasa a comparar la diferencia entre el segundo beta de cada tabla, obteniendo el $\triangle_\beta 2$ con la misma tolerancia T_β.

Finalmente, si la distancia y ángulos relativos de cada par de minucias están dentro de una tolerancia aceptable, entonces se agrega la siguiente entrada a la tabla de compatibilidad:

$$\{\triangle_\beta(\theta(P_m), \theta(G_n)), k(P_m), j(P_m), k(G_n), j(G_n)\} \tag{10}$$

Es así como una tabla de compatibilidad incorpora dos pares de minucias, uno para la primera huella $k(P_m), j(P_m)$ y otro para la segunda huella, $k(G_n), j(G_n)$. Siguiendo la misma notación, $k(P_m)$ corresponde con $k(G_n)$ y $j(P_m)$ corresponde con $j(G_n)$. Esta tabla puede verse como una serie de asociaciones de compatibilidad entre dos pares de posibles minucias iguales. Estas asociaciones representan aristas de un grafo (conocido como **grafo de compatibilidad**.

10.8.3. Recorrer Tabla de Compatibilidad

Con la tabla de compatibilidad ya creada, se tiene un serie de asociaciones compatibles entre dos pares de minucias. Si se ve desde un punto de vista de grafos, cada par $k(P_m)$ y $k(G_n)$ en cada entrada de la tabla representan dos nodos unidos, y si se tuvieran las siguientes entradas en la tabla:

$$\{\triangle_\beta(\theta(P_1), \theta(G_3)), k(P_1), j(P_1), k(G_3), j(G_3)\} \tag{11}$$

$$\{\triangle_\beta(\theta(P_1), \theta(G_5)), k(P_1), j(P_1), k(G_5), j(G_5)\} \tag{12}$$

$$\{\triangle_\beta(\theta(P_1), \theta(G_{103})), k(P_1), j(P_1), k(G_{103}), j(G_{103})\} \quad (13)$$

$$\ldots \quad (14)$$

Significaría que la 1ra entrada de la tabla P es compatible con las entradas 3, 5 y 103 de la tabla G, obteniendo el siguiente grafo:

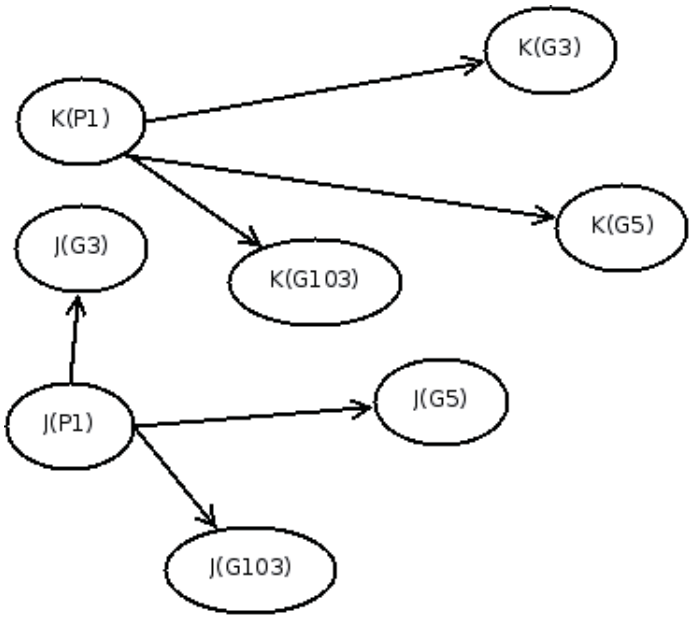

Figura 40: Parte de un posible grado de entradas de la tabla de comparación

Este algoritmo lo realiza la función `bz_match_score()`, y recibe la cantidad de entradas en la tabla de compatibilidad (la tabla misma es una variable global, por lo que lo usa directamente sin necesidad de ser pasada en la función), junto con las dos estructuras `xyt_struct` correspondientes a cada huella. El primer paso consiste en verificar que exista una cantidad mayor a 8 minucias en cada estructura, de lo contrario el resultado del match devuelto será de cero:

```
if ( pstruct->nrows < MIN_COMPUTABLE_BOZORTH_MINUTIAE ) {
        if ( gstruct->nrows < MIN_COMPUTABLE_BOZORTH_MINUTIAE ) {
                return ZERO_MATCH_SCORE;
}
```

En lugar de comenzar desde la primera entrada en la tabla de comparación, se inicia desde varios puntos de partida hasta tener el grafo completamente creado. Posteriormente se van sumando todas las asociaciones compatibles obteniendo el resultado del match (en este caso del ejemplo anterior sería de 3 .

Una vez que se obtiene el resultado de la comparación, es necesario comparar este dato con un umbral. Si este resultado es mayor o igual al umbral, entonces las minucias provienen de la misma huella, de lo contrario, son huellas distintas. Este parámetro indica si una comparación es suficientemente cercana o

no para establecer la identificación de una persona. Este umbral varía de acuerdo al sistema y dominio del problema. NIST, con su estudio de resultados de verificación[41], recomienda un umbral de alrededor de 100, por lo cual se incluyó un valor de 105 por defecto: `#define UCV_DEFAULT_THRESHOLD 105`. Sin embargo, para efectos de usabilidad y darle mayor libertad al usuario, se estableció un mecanismo donde el usuario puede asignar el umbral que estime conveniente usando la función `ucv_match_set_threshold()`, la cual modifica la variable externa `ucv_threshold`. Asimismo, puede desetablecerlo, asignándole -1, usando `ucv_match_unset_threshold()`. Internamente, para saber que umbral usar, la librería consulta el valor de `ucv_threshold`, y si éste es cero o negativo, usa `UCV_DEFAULT_THRESHOLD`. Si se establece un umbral alto, entonces el sistema se vuelve menos tolerante, y se requerirá más exactitud para que un usuario sea identificado, pero al mismo tiempo será más seguro ante usuarios que no pertenezcan al sistema. Esto se ilustra en la Figura 41.

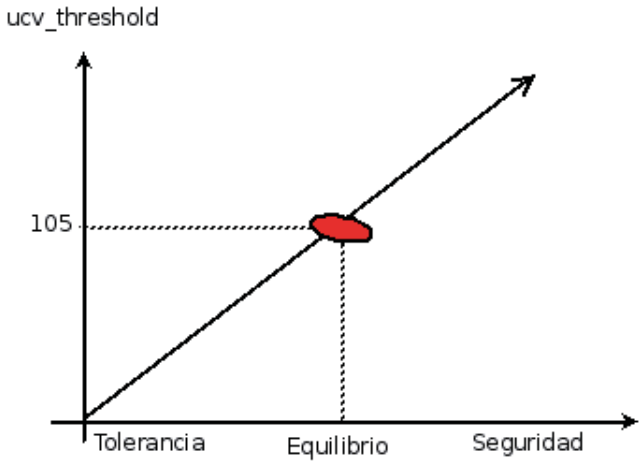

Figura 41: Relación Tolerancia/Seguridad en un sistema biométrico

.

11. Desarrollo del Prototipo

11.1. Análisis y Diseño del Prototipo

El objetivo principal del prototipo es ejemplificar algunas de las funcionalidades de la librería. Para este fin se diseñó una aplicación gráfica, en donde se visualiza, de mejor forma, las operaciones que se pueden realizar a una huella digital. Esta aplicación ofrece las siguientes funcionalidades:

- Enrolar o registrar usuarios escaneando su dedo y almacenando la las minucias pertenecientes a la huella digital, junto con el nombre y la ubicación en el sistema de archivos de la imagen de la huella.
- Eliminar usuarios previamente enrolados.
- Ver todos los usuarios registrados en el sistema.
- Escanear una huella y compararla con aquellas ingresadas previamente. Para comprobar que el usuario existe o no en la base de datos, se compara la huella capturada con todas las otras huella existentes en el sistema, haciendo así el proceso de identificación, o matching 1-N. Se calcula la cantidad de tiempo en que demora realizar cada operación. De igual manera se muestra el umbral definido, en base al dispositivo a usar, y la comparación con el resultado del matching, lo que dará un resultado exitoso o negativo.
- Ver la huella en escala gris o binarizada, junto con sus minucias e información básica, como el tipo (fin de cresta o bifurcación), cantidad de vecinos, la confianza y si aparece o desaparece dentro de la imagen. Cuando se realiza el proceso de matching, en caso de una comparación exitosa, se puede ver aquellas minucias que hicieron el matching para cada huella.
- Customizar la ubicación en el sistema de archivos donde guardar la imagen de la huella que se obtiene a través del scanner, junto con la información de conección a la base de datos.

Para almacenar la información de cada huella, se diseñó e implementó el modelo relacional de la Figura 42.

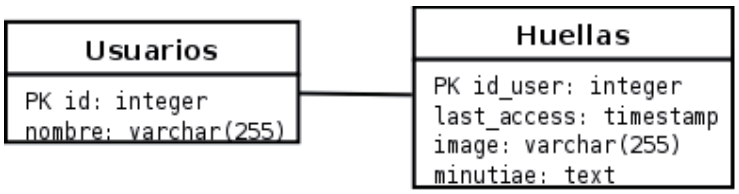

Figura 42: Modelo relacional usado para el prototipo

Como se puede ver, la tabla Usuarios y Huellas tienen una relación de 1-1. Sin duda el dato más importante son las minucias. En lugar de diseñar un modelo basado en las minucias pertenecientes a una huella, se hizo basado en el usuario mismo. Esto hace que todas las minucias se almacenen en un texto, con el mismo formato que usa la librería para almacenarlas en un archivo. Esto brinda al usuario más flexibilidad al momento de usar la librería, pues no está sujeto a un modelo relacional, y al mismo tiempo, permite mantener

la simplicidad de la librería pues reutiliza los algoritmos para parsear el formato de minucias.

Otro factor importante que se incorpora en este modelo relacional, es el campo *last_access* (último acceso). Este permite ordenar las entradas de usuarios de acuerdo al último uso que tuvo, y se refleja en una considerable optimización al momento de un matching (especialmente en bases de datos con más de 1.500 huellas).

11.1.1. Diagramas de Casos de Uso del Prototipo

Haciendo un análisis de este sistema, basado en los requerimientos funcionales, puede crearse un diagrama de casos de uso general, de las operaciones que efectúa el usuario con el prototipo (Figura 43).

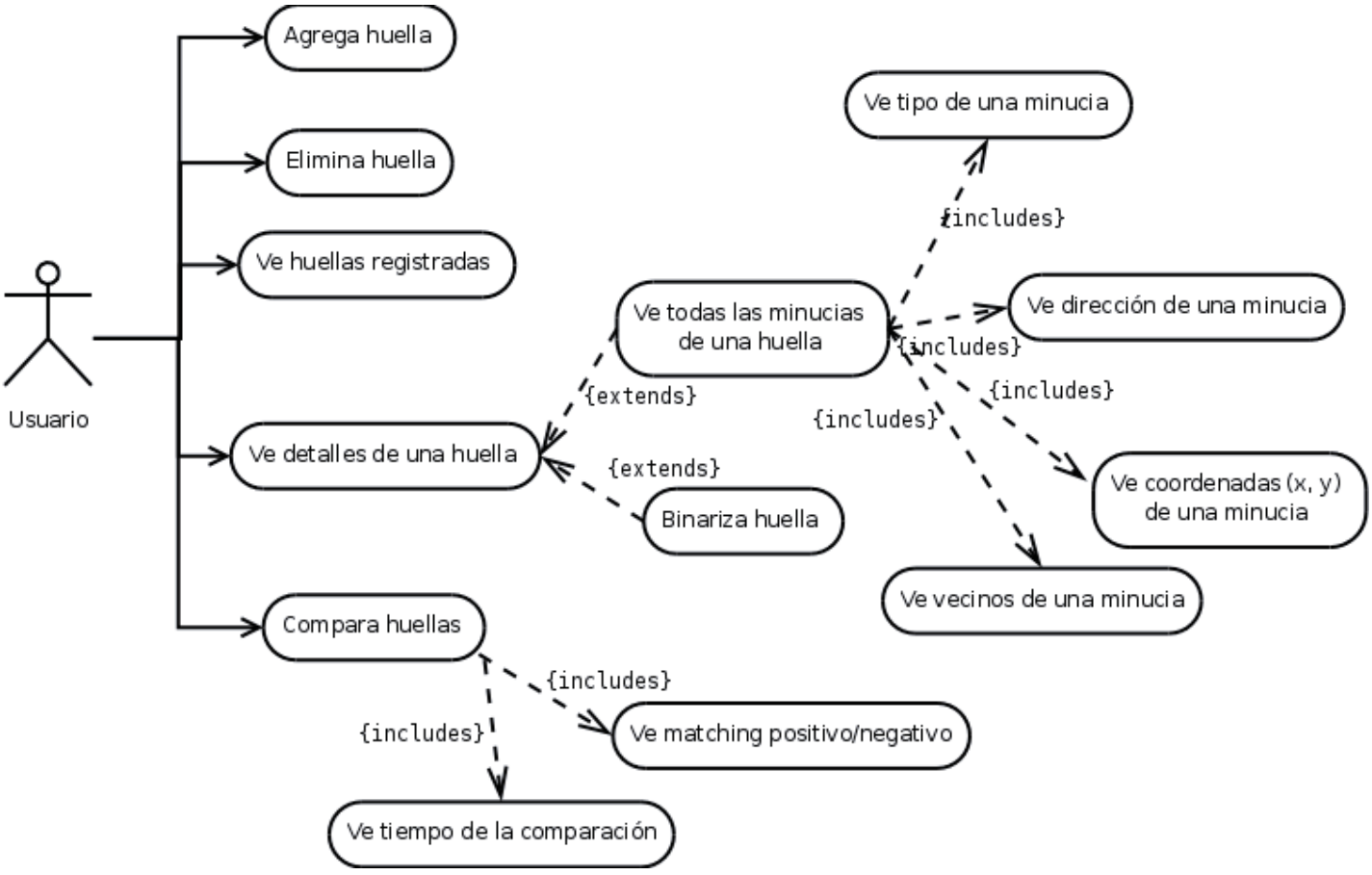

Figura 43: Diagrama de Casos de Uso General del Prototipo

11.2. Interfaz de Usuario

La interfaz de usuario definida para el prototipo es simple pero intuitiva en su uso. Se presenta una serie de botones en la parte superior de la pantalla el cual consta de solo botones:

- Preferencias: Configuración de la ubicación de las imágenes en el sistema, asi como umbral de comparación y credenciales (usuario, contraseña y dominio) para la connección a la base de datos.
- Ayuda: Una ayuda en línea que describe todos los procesos, botones y configuraciones de la aplicación.
- Acerca: Acerca del autor y propósito de la aplicación.
- Salir: Salir de la aplicación.

Abajo de la lista de botones se tienen dos secciones que se describen a continuación:

1. Buscar: En esta sección se divide la pantalla en tres columnas, a los extremos, una imagen A (la imagen escaneada recientemente) y una imagen B (la imagen con la cual se comparó exitósamente la imagen A), y en el centro, información del proceso biométrico general. Esto incluye un botón 'Find', 'Scan' y 'Stop'. El primero comienza la comparación de huellas previamente enroladas con el botón 'Scan', mientras que el último cancela la comparación.

   Debajo de estos tres botones se puede ver el resultado de la comparación ('MATCH' o 'NO MATCH'), junto con el tiempo que tomó realizar la operación, el match score y el umbral usado para la comparación.

   Volviendo a las imágenes de los extremos, las configuraciones son iguales para ambas. En la izquina izquierda de cada una hay una opcion para ver todas las minucias o solo aquellas que hicieron matching positivo, asi como ver la imagen normal (escala de gris) o binarizada (blanco y negro). Abajo de cada imagen se muestra la cantidad de minucias detectadas y la cantidad que hicieron match. Una de las características de esta aplicación es la posibilidad de hacer click en cualquier minucia y ver la información específica de cada una, mustrando las coordenadas (x, y), su dirección en grados, el porcentaje de confianza, el tipo (apareciendo o desapareciendo) y la cantidad de vecinos. Abajo de este último, se puede especificar ver los vecinos de la minucia seleccionada.

   Cabe destacar que las minucias se pueden diferenciar de distintos colores y formas respecto a la confianza y tipo (término o bifuración), respectivamente. Los colores son basados en RGB, en donde, la imagen de la izquierda colorea los puntos con mayor confianza (cerca del 100 %) son rojos (`255, 0, 0`) y se agregan intensidades de verde y azul de acuerdo a como varía, proporcionalmente, su confianza. Lo mismo se aplica para la imagen de la derecha, solo que en vez de rojo siendo el de mayor confianza, se muestra en azul (`0, 0, 255`). Para ambos, se ilustran los términos de crestas como circulos rellenos, y para las bifurcaciones circulos sin relleno.

2. Administrar Huellas: Esta sección lista todas las huellas ingresadas en el sistema, junto con los botones 'Editar' y 'Eliminar'. El primer permite reingresar la huella de un usuario, asi como cambiar el nombre del individuo, el segundo elimina el usuario del sistema.

Se puede vizualizar el prototipo en la Figura 44.

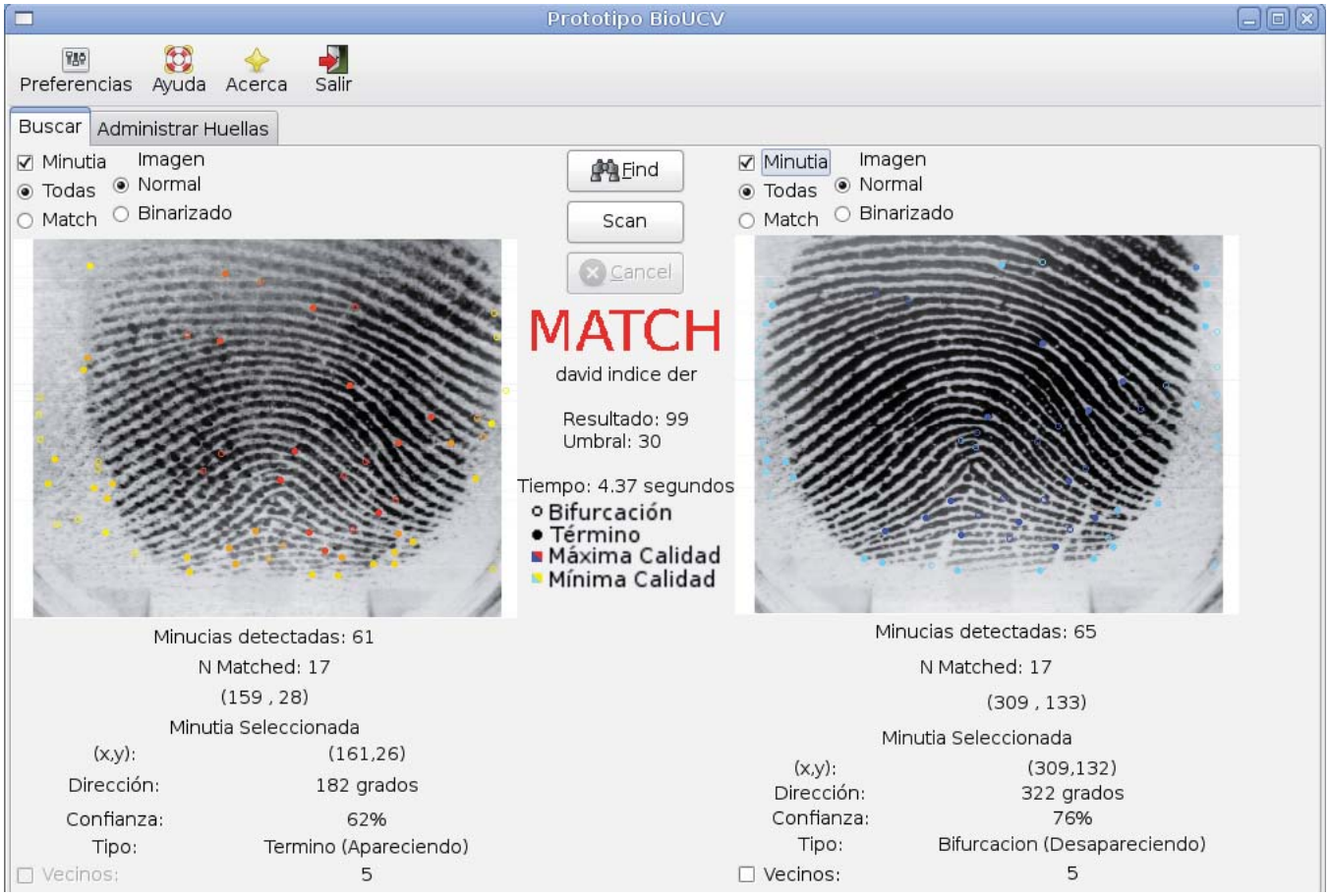

Figura 44: Pantallazo de una comparación positiva en el prototipo

11.3. Lector de Huellas Digitales

El prototipo usa un dispositivo USB para capturar huellas en tiempo real de algun usuario. El hardware que se decidió usar es el *Microsoft Fingerprint Reader*, el cual tiene un chipset *Digital Persona U.are.U 4000B*. Debido a que la empresa fabricante, DigitalPersona, provee un SDK propietario, y sin posibilidades de usarlo en ambientes no Windows/Macintosh, se usó un driver libre para Linux (licensiado bajo la *GNU General Public License (GNU GPL)* desarrollado por el proyecto fprint. El código fue adaptado para usarse con la librería, conectándolo con las interfaces desarrolladas, para funcionar en armonía. Asimismo se corrigieron una gran cantidad de bugs y problemas de memoria que podian causar segmentaciones de memoria (escribiendo en direcciones no reservadas)[15]. Este parche ya fue integrado en el proyecto, y hoy en día el autor de esta tesis ya es el mantenedor.

11.3.1. Imágenes Capturadas

La Figura 46 ilustra una huella capturada por el dispositivo, y se pueden mencionar una serie de características:

- El dispositivo inicialmente captura la imagen al reves y con las crestas blancas y fondo negro, posteriormente, por software, se invierte la imagen y los colores, reflejando lo que se ve en la Figura.
- Se puede ver un fondo con mucho ruido o suciedad, esto se debe a que el sensor que captura la imagen embedida en el dispositivo es muy delicado y suceptible a la superficie que se usa. Esto causa que se detecten muchas minucias falsas, y gracias a su eliminación, puede volverse más confiable. Se realizaron pruebas con otros dos dispositivos para comparar la cantidad de minucias falsas detectadas y eliminadas, los resultados se pueden ver en la Figura 45. Se escanearon 100 veces un dedo índice derecho para una persona de 25 años usando los dispositivos Secugen Hamster III, UPEK y Microsoft Fingerprint Reader. Como se ve, el hardware de Microsoft, es el que más minucias falsas detecta.

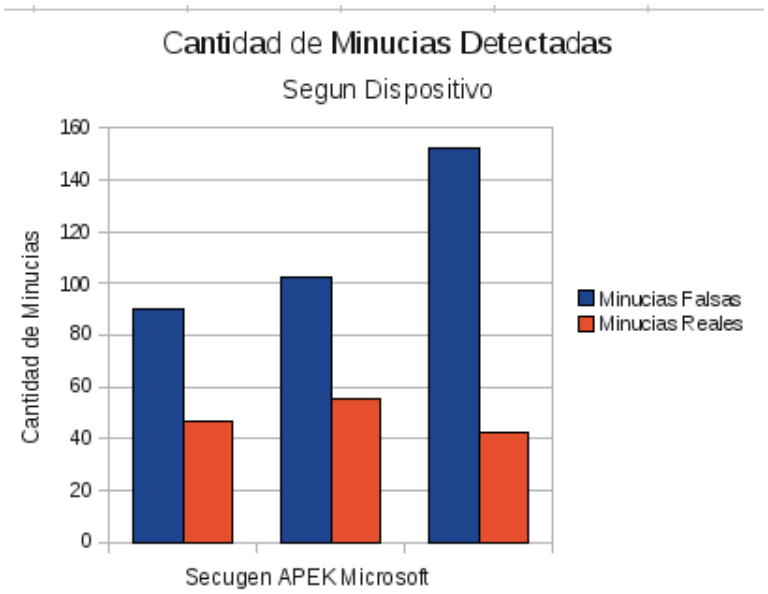

Figura 45: Resultados de la detección de minucias con tres dispositivos distintos

Se probó el lector 100 veces para una persona de 25 años, arrojando una buena cantidad de minucias detectadas (45.3), y lo mismo para una persona de 50 años, con menos cantidad de minucias (39.1). Esto demuestra que si bien varía la cantidad de acuerdo a la edad de una persona, el dispositivo refleja esto de buena manera, y es capaz de entregar imágenes verídicas de un dedo. Se pueden ver otras pruebas de comparación de huellas realizadas con este dispositivo en las Figuras 48 y 49.

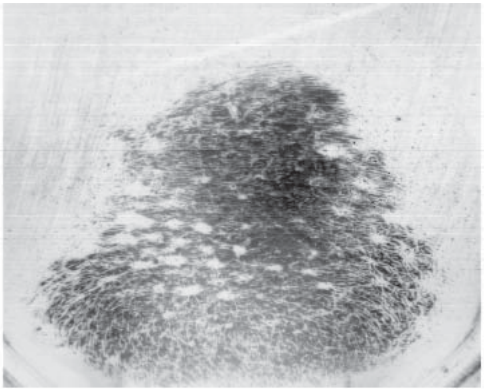

Figura 46: Huella digital capturada por el dispositivo Microsoft Fingerprint Reader

11.4. Implementación del Prototipo

Este proyecto usará una serie de herramientas externas para su implementación. Cada una cumple un factor importante en la estructura del prototipo:

- MagicCore: Es una librería compuesta por interfaces de bajo nivel entre en lenguaje C y la librería de tratamiento de imágenes ImageMagick. Esta herramienta soporta más de 90 formatos de imagenes digitales como JPEG, PNG, TIFF y BPM, entre otros.
- AutoTools: Esta herramienta permite configurar y compilar el proyecto para diferentes plataformas computacionales, manteniendo una uniformidad. Con AutoTools es posible comprobar que las dependencias del proyecto estén previamente instaladas antes de compilar, ahorrando tiempo y evitando errores al momento de compilar.

Se implementó esta aplicación en el lenguaje C y para la interfaz gráfica se usó GTK+. Este permite visualizar de mejor manera las operaciones que se le pueden ir haciendo a las huellas digitales, tales como binarización, ubicación de minucias y matching, entre otros. Como base motor de base de datos se decidió utilizar PostgreSQL, pues es una opción libre y altamente confiable.

12. Pruebas

Las pruebas de software son una de las partes más importantes al desarrollar este proyecto, y, como debe funcionar con una amplia gama de aplicaciones, es necesario asegurarse que exista la menor cantidad de *bugs* posible. Las pruebas se dividieron en dos categorías: (i) de funcionalidad: asegura que cada función hace lo que dice hacer, y (ii) de rendimiento: que la librería haga buen uso de los recursos computacionales (especialmente si se piensa en escenarios de sistemas embedidos). La Figura 47 ilustra los pasos secuenciales que se adoptaron para programar cada función de la librería que forma parte de la API.

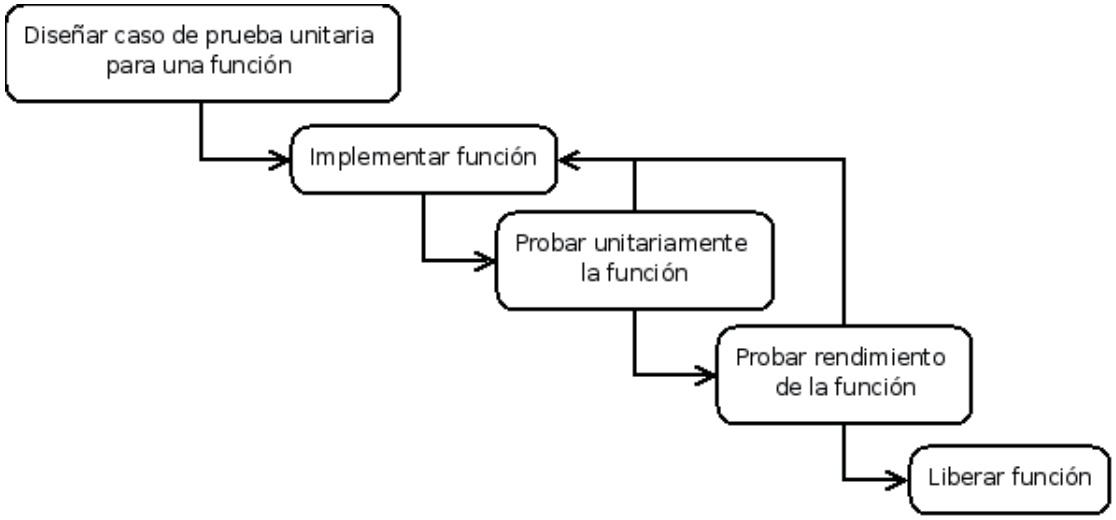

Figura 47: Secuencia de pasos para desarrollar cada función

Se comienza por diseñar un caso de prueba para una función en particular, por ejemplo, los parámetros y salidas, todo basado en el manual de referencia creado anteriormente (aunque cuando esté sujeto a cambios futuros, pero las interfaces no deberían cambiar). Luego se implementa la función en el lenguaje. Una vez que compila sin errores ni *warnings*, se procede a probarla usando el caso de prueba ya creado. Si no supera la prueba, se vuelve a la fase de programación anterior hasta que pase la prueba. Una vez que se sabe que la función hace lo que se espera, se pasa a probar el rendimiento computacional, y si se determina que algo puede ser mejorado (como disminución de cantidad de memoria que se usa), entonces se vuelve a la fase de programación. Una vez concluida, se sigue con la prueba unitaria nuevamente, posteriormente a las pruebas de rendimiento, y finalmente, se considera como completada la función. Es importante seguir esta metodología para cada función, pues asegura que cada una haga lo que se espera ella, y de buena forma.

12.1. Pruebas de Funcionalidad

Cada vez que se desarrolla un software, es fundamental asegurarse que se comporte como se espera, y realice las tareas que se definieron en las fases anteriores, como por ejemplo, el diseño. Para este proyecto se decidió usar pruebas unitarias (o *unit testing*) para asegurar que cada función se comporte como se espera, y que cada parámetro de salida y entrada sea acorde con lo que se establece en el manual de referencia. Una unidad es la parte *testeable* más pequeña de una aplicación, como métodos o funciones[22]. Si bien esta técnica es muy usada para lenguajes orientados a objetos, el concepto es aplicable a cualquier paradigma, y, por ende, a la programación estructurada.

Al decidir utilizar esta técnica por sobre otras, se tomó en cuenta la naturaleza de cambio propia de una librería, y la complejidad de la tecnología con la que se trabaja. Al construir una prueba unitaria, la función a probar debe seguir un comportamiento estrictamente definido para pasarla. Estas pruebas facilitan los cambios en el software, pues al crear una prueba, ésta se puede reutilizar tantas veces como sea necesario, y se puede probar si los cambios hechos a una función la satisfacen. Otro beneficio al usar pruebas unitarias es que, al diseñar los casos de pruebas, se puede trabajar en conjunto con el manual de referencia de la API, asegurando así que concuerden.

Existen muchos *frameworks* para pruebas unitarias en distintos lenguajes, y se optó por usar uno denominado CUnit[1]. Este es un proyecto de software libre que consiste en una librería estática para construir y administrar pruebas unitarias en C. Incorpora varios casos de pruebas predefinidas, lo que ayuda a simplificar el proceso de *testing*. Además, la documentación y ejemplos son ilustrativos para todo tipo de usuarios que requieren probar sus aplicaciones.

Con la ayuda de CUnit se establecieron conjuntos de pruebas (denominadas *suites*) agrupadas por módulo. Una *suite* contiene un conjunto de pruebas referentes a cada función, y a medida que estas se van implementando, se van agregando a la *suite*. Como se mencionó anteriormente, las pruebas están en el directorio `tests/functionality/` del código fuente, compuesto por diversos archivos siguiendo la notación: `<modulo>-suite.c`. Se crearon las siguientes *suites*:

- Image: Contiene las pruebas de las funciones que pertenecen al gestor de imágenes. Todo el trabajo con imágenes no depende de ninguna otra parte de la librería, por lo que se puede probar de forma independiente.
- Minutiae: Contiene las pruebas de las funciones que pertenecen al extractor de minucias. Esta *suite* solo cubre aquellas funciones que pueden funcionar de manera independiente, pues algunas requieren de trabajo del gestor de imágenes.
- MinutiaeImage: Contiene las pruebas de las funciones que comparte el gestor de imágenes con el extractor de minucias.
- MatchingMinutiae: Contiene las pruebas de las funciones que comparte el extractor de minucias y comparador de huellas.

- MatchingImage: Contiene las pruebas de las funciones que comparte el gestor de imágenes con el comparador de huellas.

A continuación se ilustra un ejemplo de salida de pruebas unitarias al módulo `Image` de la librería. Se corrieron diez pruebas para diez funciones, dos de las cuales fallaron, por motivos de cambios en el código.

```
Suite: Image Suite
  Test: test of ucv_img_to_data() ... passed
  Test: test of ucv_img_get_height() ... passed
  Test: test of ucv_img_get_width() ... passed
  Test: test of ucv_img_get_data() ... passed
  Test: test of ucv_img_get_height + ucv_img_get_width() ... passed
  Test: test of ucv_img_vflip() ... passed
  Test: test of ucv_img_data2pgm() ... passed
  Test: test of ucv_img_hflip() ... FAILED
    1. image-suite.c:124  - CU_ASSERT_EQUAL(len_0,len_1+1)
  Test: test of ucv_img_invert_colors() ... passed
  Test: test of ucv_img_free() ... FAILED
    1. image-suite.c:137  - !img

--Run Summary: Type      Total     Ran  Passed  Failed
               suites        1       1     n/a       0
               tests        10      10       8       2
               asserts      11      11       9       2
```

12.2. Pruebas de Rendimiento

Siempre es importante usar bien los recursos computacionales que se tienen, y parte fundamental de esto es programar aplicaciones que lo hagan. Si una librería tiene fallas, por ejemplo, de memoria, entonces las aplicaciones que hagan uso de ella se verán inmediatamente afectadas de igual manera (sin considerar las fallas propias de ésta). Es muy fácil desarrollar software que haga mal uso de la memoria (por mencionar una), sobretodo en lenguajes de bajo nivel, como C. Es por esto que es importante probar el rendimiento computacional que tiene cada funcionalidad de la librería. Para esto se utilizó 'valgrind'[33], una aplicación para depurar programas de acuerdo al comportamiento en cuanto al uso de memoria, *cachè* y CPU, entre otras características. Esta herramienta dispone de diversos módulos, cada uno cumpliendo una tarea particular, y para este proyecto se usaron:

- `memcheck`: un depurador de memoria que ayuda a detectar bugs de bajo nivel como:
  - Uso de memoria no inicializada.
  - Liberación de memoria no utilizada.
  - Memoria asignada pero no liberada (conocido como *memory leaks*).

  - Lectura y escritura de memoria no asignada

- `massif`: un medidor de memoria que ayuda a detectar niveles muy altos de consumo comparados a la ejecución normal de una aplicación. Esta herramienta ayuda a eliminar cuellos de botella y posibles consumos excesivos de espacio *swap*.

Como se indicó anteriormente, cada vez que se concluyen las pruebas unitarias para una función, se hace una prueba de rendimiento que abarca dos ejecuciones de valgrind, una con memcheck y otra con massif. A continuación se ilustra una ejecución de memcheck ante el prototipo desarrollado durante este proyecto:

```
dave@laptop:~/bioucv/lib-bioucv/tests$ valgrind --tool=memcheck ./test_minutiae prints/test.tif
==7220== Memcheck, a memory error detector.
==7220== Copyright (C) 2002-2007, and GNU GPL'd, by Julian Seward et al.
==7220== Using LibVEX rev 1732, a library for dynamic binary translation.
==7220== Copyright (C) 2004-2007, and GNU GPL'd, by OpenWorks LLP.
==7220== Using valgrind-3.2.3-Debian, a dynamic binary instrumentation framework.
==7220== Copyright (C) 2000-2007, and GNU GPL'd, by Julian Seward et al.
==7220== For more details, rerun with: -v
==7220==
Minucias detectadas: 33
==7220==
==7220== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 53 from 1)
==7220== malloc/free: in use at exit: 0 bytes in 0 blocks.
==7220== malloc/free: 9,160 allocs, 9,160 frees, 32,735 bytes allocated.
==7220== For counts of detected errors, rerun with: -v
==7220== All heap blocks were freed -- no leaks are possible.
```

Esta salida, entre otras cosas, muestra que no existen errores en el uso de memoria, y que durante el tiempo de ejecución del programa se hicieron 9.160 asignaciones de memoria (reservando un total de 332.735 bytes) y se liberó la misma cantidad de bloques, haciendo imposible que existan *memory leaks*. Es importante que no existan errores de memoria en esta librería, pues si se piensa que trabaja generalmente con bases de datos de huellas (por ejemplo, 1600 en el caso del prototipo), entonces si no se liberan todos los bloques de memoria, habría una perdida de `1600 * cantidad de bytes no liberados` por ejecución. Esto puede resultar en un serio problema para cierto tipo de plataformas.

12.3. Pruebas de Estrés

Estas pruebas permiten ver el desempeño de la librería desarrollada en este proyecto en situaciones de alta demanda. Se diseñó una prueba de matching 1-N, usando un universo de 1600 huellas digitales provenientes del *Fingerprint Verification Contest - FVC2000*[41]. Se tomaron aleatoriamente 50 muestras de este universo, y se les hizo un matching positivo y negativo. Esto se probó en dos máquinas de muy distintas capacidades: la primera, un servidor de alta capacidad, mientras que la segunda fue un computador personal (fácilmente obtenible por cualquier usuario). A continuación se describen las características de cada una.

- Máquina 1:
  - Procesador: Sun Ultra SPARC 64bit
  - Memoria RAM: 8 Gb
  - Sistema Operativo: Sun Solaris 10
- Máquina 2:
  - Procesador: Intel Pentium(R) M 1.60GHz
  - Memoria RAM: 512 Mb
  - Sistema Operativo: Debian GNU/Linux

Las pruebas miden el tiempo que toman en comparar una huella contra una base de datos de 1.600 huellas. Se hace uso exclusivo de las funciones desarrolladas en este proyecto: `ucv_minutiae_load_from_file()` y `ucv_matcher_verify()`. La lógica que se implementó para realizar estas pruebas consta de dos partes: la primera es leer imágenes de huellas digitales, y extraer sus minucias para almacenarlas en archivos de texto plano (usando la función `ucv_minutiae_save_to_file()`). El segundo paso consiste en leer los archivos de minucias, pasarlas a memoria, y, finalmente, compararlas contra una imagen de huella digital. Estos pasos se pueden ver a continuación:

```
1.- Tomar un directorio con N cantidad de huellas digitales.
2.- Leer las N imágenes y pasarlas a N archivos de minucias.
3.- Cargar en memoria los N archivos de minucias.
4.- Comparar una huella cualquiera con las N minucias almacenadas
en memoria.
```

La Figura 48 ilustra el resultado de las pruebas con matching positivo. Al probarlo en la Máquina 2, los valores fluctuan entre los 0 y 7 segundos, pues depende del momento dentro del proceso en que las huellas hicieron una comparación positiva. Si se logró al principio de la lista, entonces tendrá un valor muy cercano a los $]0, 1]$ segundos, sin embargo, si se encuentra cerca del final, estará en el rango de $[6, 7]$ segundos. Algo muy similar ocurre en la Máquina 1, sólo que con rangos de $]0, 4]$ segundos. La media del tiempo demorado en hacer un matching positivo para la Máquina 1 es de 2.05 segundos, mientras que para la Máquina 2 es de 3.01. Esto ilustra que la enorme diferencia de memoria RAM entre ambas máquinas no impacta tan drásticamente en el comportamiento de la aplicación (salvo algunos *peaks* donde se perciben tiempos de 7 segundos y 4 segundos, para las Máquinas 2 y 1, respectivamente).

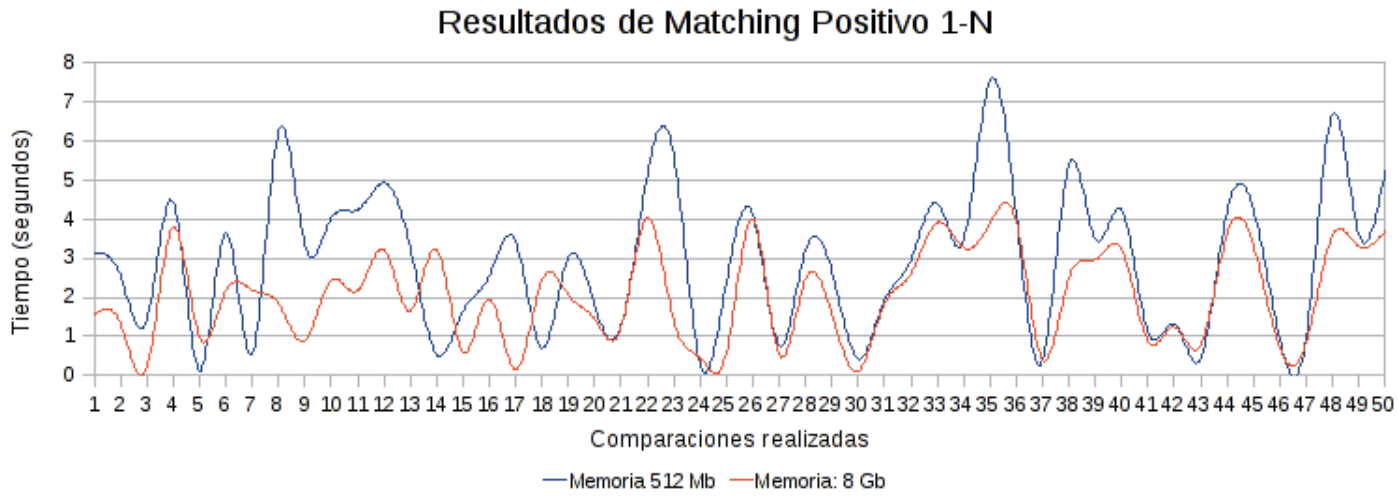

Figura 48: Resultados de un matching positivo contra una base de datos de 1600 huellas digitales

La Figura 49 ilustra el resultado de las pruebas con matching negativo. El objetivo de esto es ver cómo se comportan las funciones en el peor caso: un matching negativo requiere comparar las 1.600 huellas (en el positivo, al encontrar un match, termina el proceso inmediatamente). El promedio de tiempo tardado para la Máquina 1 es de 2.77 segundos, mientras que para la Máquina 2 es de 5.83 segundos. A diferencia de la primera prueba de matching positivo, aquí se puede ver una drástica mejora en los tiempos, gracias a la diferencia de memoria RAM de cada computador.

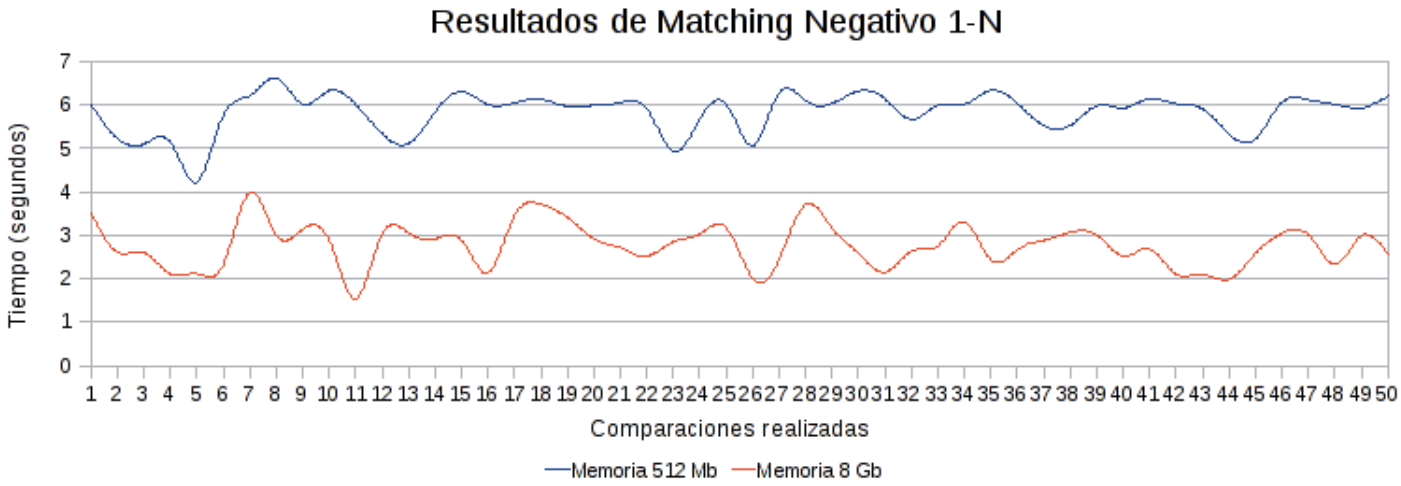

Figura 49: Resultados de un matching negativo contra una base de datos de 1600 huellas digitales

Con estos datos, se puede concluir que si bien las funciones de la librería se comportan dentro de los tiempos esperados, sí existe una diferencia significativa respecto a la cantidad de RAM utilizada en cada máquina, especialmente en las comparaciones negativas (*no match*).

13. Conclusiones

El contar con una herramienta de fácil integración de tecnologías de huellas digitales a aplicaciones de diversa índole, presenta, sin duda, una enorme ventaja para desarrolladores de software. Este es el tipo de usuario al cual apunta este proyecto. El prototipo desarrollado presenta un claro ejemplo de cómo se logra, en sólo un par de líneas de código, tener un sistema que pueda realizar todo el proceso biométrico, vale decir, desde que la huella se captura con un dispositivo, hasta que se realiza una comparación de huellas (*matching*). El campo de las aplicaciones que pueda hacer uso de este proyecto es muy grande: desde sistemas que abran puertas (embedidos) hasta servidores de matching (máquinas dedicadas a procesar y comparar grandes cantidades de huellas digitales, generalmente utilizadas en campos de investigación forense).
Técnicamente, cabe destacar que la librería desarrollada en este proyecto nace del código fuente del NBIS, el cual proporciona los algoritmos básicos de extracción de minucias y matching. Sin embargo, lo innovador de este proyecto es su conversión a una librería (pues, hasta ahora, era sólo parte de un colección de pequeñas aplicaciones utilitarias). A esta librería se le agrega una capa de abstracción que incluye un manejo de imágenes que permite una fácil y rápida integración con dispositivos, así como la posibilidad de realizar una comparación de huellas 1-N (el algoritmo usado sólo permite comparaciones 1-1). Una de las áreas más interesantes de este proyecto fue la modificación del algoritmo de Bozorth para poder ver el camino exacto que utiliza en cada par de huellas para calcular el resultado del match. Dado la falta de documentación, y lo mal comentado del código (además de variables poco descriptivas), esta tarea se basó principalmente en realizar un proceso de ingeniería reversa. Actualmente se envió un *patch* con los cambios realizados a Michael D. Garris (autor del extractor de minucias utilizado), el cual está evaluando para integrarlo a una nueva versión del NBIS.

Desde un punto de vista de usabilidad de la librería desarrollada, además de presentar una API con interfaces bien definidas y documentadas (reflejados en el manual de referencia), permite la integración de diferentes algoritmos biométricos. Esto se debe a que como fue diseñado modularmente, por lo que es muy sencillo cambiar las funcionalidades, por ejemplo, para utilizar un método de matching distinto. Esta característica no es casual, pues se pensó en el usuario a la hora de integrar el NBIS a la librería.

Después de completar este proyecto, y meditar sobre el trabajo realizado, queda la satisfacción de haber desarrollado un proyecto desafiante e interesante. Existe la certeza que esta nueva herramienta podrá ayudar a muchos desarrolladores de software a simplificar la tarea de integrar huellas digitales en sus aplicaciones. Por esto mismo es que se quiere dar un enfoque de código abierto y liberarlo, para que así la comunidad de software libre pueda beneficiarse y aportar nuevas ideas a esta librería. Se pretende continuar con esta tarea específica después del transcurso de este trabajo académico. Para el trabajo futuro se desea incorporar la librería en sistemas embedidos, pues, como es portable, es muy sencillo de usarse en sistemas operativos unix pequeños (como μ-Linux o Minix3) y hardware de bajo consto. Esto brinda una nueva ventana de posibilidades biométricas para un sin fin de nuevos proyectos privados y comerciales. Otra área para continuar este trabajo es la seguridad, el 2006 un hacker Finlandés, Mikko Kiviharju, presentó avances en ataques a lectores de huellas digitales que consiste en la captura de paquetes que se envian entre el bus USB y la imagen generada[20]. Sería de interés para la investigación y el uso de este proyecto, desarrollar una capa de encriptación entre ambos canales, basándose en un arquitectura OpenSSL, pues esto es exactamente lo que ocurre con SSH y Telnet (este último no envía información encriptada).

Referencias

[1] Benner, Philipp; CUnit, http://cunit.sourceforge.net

[2] Blank, Leland T., Tarkin, Anthony J.; *Ingeniería Económica 4ta Edición*, McGraw Hill, Santa Fé de Bogotá, Bogotá, Colombia, 1999.

[3] Chikkerur, Sharat; *Online Fingerprint Verification*, Center for Unified Biometrics and Sensors, Universidad de Buffalo, Nueva York, Estados Unidos.

[4] Código fuente del NIST Biometric Image Software.

[5] Copyright Law of the United States of America and Related Laws Contained in Title 17 of the United States Code, http://www.copyright.gov/title17/92chap1.html#105

[6] Drepper, Ulrich; *Good Practices in Library Design, Implementation, and Maintenance*, Red Hat Inc., Estados Unidos, Marzo, 2002.

[7] Fidelica Microsystems Inc., *Interpreting Fingerprint Authentication Performance*, Milpitas, California, Estados Unidos, Diciembre, 2001.

[8] Fraser, Bruce; *Understanding Digital Raw Capture* Whitepaper, Adobe Systems Inc., San Jose, California, Estados Unidos.

[9] Free Software Foundation, http://www.gnu.org

[10] GNU Compiler Collection, Free Software Foundation, http://gcc.gnu.org

[11] Gough, Brian; *An Introduction to GCC: For the GNU Compilers gcc and g++*, Network Theory Limited, Bristol, Inglaterra, Marzo, 2004.

[12] Hapeman, Dale; Hutchinson, James B.; *Biometric Collection, Transmition and Storage Standards Technical Reference*, US Department of Defence Biometrics Task Force Standards Team, Julio, 2006.

[13] IBM Exploratory Computer Vision Group, International Business Machine, http://research.ibm.com/ecvg

[14] Identix Corporation Inc., *Minutia Vs. Pattern Based Fingerprint Templates* White Paper, Marzo, 2003.

[15] Internet Email Archive, [fprint] FDU 2000 patch, Davidlohr Bueso, http://www.mail-archive.com/fprint@reactivated.net/msg00306.html

[16] Jain, Anil K.; Pankanti, Sharath; Bolle, Ruud M.; *Introduction to Biometrics*, IBM Thomas J. Watson Research Center, Yorktown Heights, Nueva York, Estados Unidos.

[17] Jain, Anil K.; Pankanti, Sharath; *Fingerprint Classification and Matching*, Handbook for Image and Video Processing, A Bovik (Ed.), Academic Press, Septiembre, 2000.

[18] Jain, Anil K.; Ross, Arun; Prabhakar, Salil; *Fingerprint Matching Using Minutiae and Texture Features*, International Conference on Image Processing, Atenas, Grecia, Octubre, 2001.

[19] Jain, Anil K.; Pankanti, Sharath; Hong, Lin; Ross, Arun; Wayman, James L.; *Biometrics: A Grand Challenge*, Proceedings of International Conference on Pattern Recognition, Cambridge, UK, Agosto, 2004.

[20] Kiviharju, Mikko; *Hacking fingerprint scanners*, Black Hat 2006 Conference, Las Vegas, Nevada, Estados Unidos, 2006.

[21] Krishnan, Rama; *Fingerprint Capture Challanges and Opportunities*, IDENT - Biometrics Quality Lead, Department of Homeland Security, Estados Unidos.

[22] Kolawa, Andrew; *Why, When, How: Unit Testing White Paper*, http://www.parasoft.com/jsp/redirector.jsp/WWH_UnitTesting_W

[23] Licencia NIGOS, http://www.itl.nist.gov/iad/894.03/nigos/NIGOS_licdis_061906.pdf

[24] Luper, David; *Image Processing in Biometrics: Special Focus on Fingerprint Recognition*

[25] Manual de Referencia de ImageMagick: http://www.imagemagick.org

[26] Maltoni, David; Maio, Dario; Jain, Anil K.; Prabhakar, Salil; *Handbook of Fingerprint Recognition*, Springer, ISBN 0-387-95431-7, Nueva York, Estados Unidos, 2005

[27] Modi, Shimon K; Elliot, Stephen J.; *Impact of Image Quality on Performance: Comparison of Young and Elderly Fingerprints*, Universidad de Purdue, West Lafayette, Indiana, Estados Unidos.

[28] NIST Biometric Image Software, http://fingerprint.nist.gov/NBIS/index.html Marzo, 2008.

[29] Patel, Shivang; *Fingerprint Verification System* - http://fvs.sf.net, 2002.

[30] Pressman, Roger S.; *Ingeniería del Software: Un Enfoque Práctico* 5ta Edición, McGraw Hill, ISBN 84-481-3214-9, Madrid, España, 2002.

[31] Popović, Brankica M.; Masković, Ljiljana; *Fingerprint Minutiae Filtering Based on Multiscale Directional Information*, Agosto, 2007.

[32] Samet, H.; *The Design and Analysis of Spatial Data Structures*, Addison-Wesley, Reading, Massachussets, Estados Unidos, 1990.

[33] Seward, Julian (jseward@acm.org), Valgrind, http://www.valgrind.org

[34] Swarns, Rachel; *Senator? Terrorist? A Watch List Stops Kennedy at Airport*, New York Times, Agosto, 2004.

[35] Tabassi, Elham; Wilson, Charles L.; Watson, Craigh I; *Fingerprint Image Quality* NISTIR 7151, National Insitute of Standards and Technology, Gaithersburg, Maryland, Estados Unidos, Agosto, 2004.

[36] Thai, Raymond;, *Fingerprint Image Ehnhancement and Minutiae Extraction*, School of Computer Science and Software Engineering, University of Western Australia, Australia, 2003.

[37] Universidad de Bologna, SFINGE, http://biolab.csr.unibo.it, Bologna, Italia.

[38] Watson, Graigh I., Garris, Michael D., Tabassi, Elham., Wilson, Charles L., McCabe, Michael R., Stanley, Janet; *User's Guide to NIST Fingerprint Image Software 2 (NFIS2)*, National Institute of Standards and Technology, Gaithersburg, MD. Estados Unidos.

[39] Wilson, L.B. & Clark, R.G; *Comparitative Programming Languages*, Addison-Wesley, ISBN 0-201-18483-4 Wokingham, Inglaterra, 1988.

[40] Wilson, Charles L.; Watson, Craigh I.; Hicklin, A.; Ulery, B.; Korves, H.; Zoepfl, M.; Grother, P.; Michaels, R.; Otto, S.; Bone, M.; *Fingerprint Vendor Technology Evaluation (FpVTE)*, Technical Report NISTIR 7123, Junio, 2004.

[41] Wilson, Charles L.; Watson, Craig I.; Garris, Michael D.; *Studies of Fingerprint Matching Using the NIST Verification Test Bed (VTB)* NIST IR 7020, National Institute of Standards and Technology, Gaithersburg, Maryland, Estados Unidos, Julio, 2003.

[42] Yi-Sheng, Michael Yao; Pankanti Sharath; Haas, Norman; Ratha, Nalini; Bolle, Ruud M.; *Quantifying Quality: A Case Study in Fingerprints*, IBM Thomas J. Watson Research Center, Yorktown Heights, Nueva York, Estados Unidos.