Universidad Zaragoza

Trabajo Fin de Máster

Autenticación por huella dactilar para dispositivos Android mediante captura de imágenes

MEMORIA

Autor

Javier Granado Fornás

Director

José Elías Herrero Jaraba

Escuela de Ingeniería y Arquitectura

2014

“Como no sabía que era imposible lo hice”

Albert Einstein.

Agradecimientos

Cuando le comenté a Elías mi idea de proyecto, el tema le pareció muy interesante aunque lo veía complejo. Sin embargo después de hablar sobre el tema y con muchas dudas sobre su viabilidad, apostó por el proyecto y me dio el impulso que necesitaba para emprender definitivamente la tarea. Así, quedamos en que haríamos fotos de un montón de huellas y empezaríamos a trabajar con ellas en MatLab. Lo demás ya es historia.

Mi primer agradecimiento por tanto es para Elías, por la paciencia, el apoyo y el optimismo que siempre mostró hacia el proyecto aún cuando había muchas cuestiones por resolver. También por las reuniones por Skype desde nuestras respectivas casas, fuera de horario y los correos intercambiados a altas horas de la madrugada para poder compaginar el proyecto con mi horario laboral.

También quiero agradecer a mis compañeros de trabajo, las huellas que desinteresadamente me han "prestado" y que han servido para realizar este proyecto.

En este proyecto están también los esfuerzos de otras muchas personas. Ninguna de ellas ha tecleado ni una sola línea de código del mismo, ni falta que les hace. Son todas las personas que me han apoyado, aguantado, sufrido, criado, comprendido y algunas de ellas, por suerte para mí, todavía lo hacen. Me refiero a mis padres, a mis hermanas y sobre todo a mis hijos Marcos y Raquel, ojala que mi actitud y mi ilusión por elegir los retos y desafíos en la vida les sirvan de ejemplo, solo por eso ya habría valido la pena hacer este proyecto.

Mi último y mayor agradecimiento es para Rosa, mi mujer, que después de tantos años a mi lado sigue siendo la única persona en el mundo que siempre me ha apoyado en todo lo que he emprendido y que ha tenido que sufrir más que nadie mis "locuras" estudiantiles y mis noches sin dormir. Sin ella nunca lo hubiera conseguido.

Gracias a todos.

Resumen

En el presente proyecto se ha desarrollado una aplicación para dispositivos móviles con Sistema Operativo Android. Esta aplicación es capaz de realizar la autenticación de personas mediante la captura de las huellas dactilares de las mismas con la cámara del propio dispositivo.

El presente proyecto afronta el reto de no disponer de sensores de huella dedicados, que es la forma de capturar la huella a tratar, incluso en los dispositivos móviles de ultima generación (iPhone 5S, Galaxy S5), en los que se incorpora esta funcionalidad.

Otro importante reto en el presente proyecto es el de implementar en un dispositivo móvil, como teléfonos o tablets, los algoritmos necesarios para conseguir la autenticación.

La aplicación desarrollada tiene una doble vertiente. Por un lado es capaz de realizar la autenticación en tiempo real de un número limitado de usuarios, cuyas huellas se almacenan localmente en la memoria del dispositivo. El número de usuarios está limitado por el tiempo necesario para comparar la huella a identificar contra las demás huellas almacenadas.

Por otro lado la aplicación es capaz de generar una comparación cruzada de todas las huellas alojadas en una base de datos en *DropBox*[1], a la cual se "suben" todas las huellas capturadas por el dispositivo. La propia aplicación es capaz de crear una hoja Excel en la cual se refleja el resultado obtenido de la comparación de cada huella con todas las demás.

Para lograr este objetivo, se ha optimizado el algoritmo de autenticación para adoptar una solución de compromiso entre fiabilidad y tiempo de proceso.

En este proyecto se han utilizado distintas herramientas software. En la etapa inicial del proyecto se ha utilizado MatLab para verificar si era posible implementar un algoritmo que diera buenos resultados con las imágenes de las huellas capturadas con la cámara del dispositivo.

[1] Dropbox es un servicio de alojamiento de archivos multiplataforma en la nube, operado por la compañía Dropbox. Está disponible para Android, Windows Phone, Blackberry e IOS (Apple). Dropbox es un software que enlaza todas las computadoras mediante una sola carpeta, lo cual constituye una manera fácil de respaldar y sincronizar los archivos.

Posteriormente, se desarrolló la aplicación para Android bajo lenguaje de programación Java. La parte de tratamiento de la imagen se implementó haciendo uso de la biblioteca OpenCV, que es una biblioteca de uso libre de visión artificial originalmente desarrollada por Intel.

En el primer capitulo se hace una introducción, repasando el estado del arte de la biometría y la detección de huellas dactilares. También se recoge la motivación y los objetivos del proyecto.

En el segundo capitulo se recogen los primeros desarrollos del proyecto, que comprenden los métodos de tratamiento de la imagen de la huella realizados con MatLab como paso previo a la implementación posterior del algoritmo final en la aplicación Android.

En el capitulo tres se recoge el desarrollo de la aplicación Android, tratando aspectos como el uso de la biblioteca OpenCv y la implementación final del algoritmo en la aplicación.

En el capitulo cuatro se recoge la obtención de resultados y su interpretación.

En el capitulo cinco se detallan las conclusiones generales del proyecto.

En el capitulo seis se hace una reflexión sobre los trabajos futuros y los posibles desarrollos en los que se podría avanzar para mejorar el trabajo realizado.

El capitulo siete recoge la bibliografía consultada.

En el anexo A "Huellas Dactilares", se hace un repaso sobre la historia, características de las huellas, los tipos de sensores utilizados para su captura y las minucias.

En el anexo B "Estructura de la aplicación", se explican las distintas partes o "Activities" en las que está dividida la misma, con una explicación de cada una de ellas. Este anexo se puede entender también como una "guía de usuario" de la aplicación.

Índice

Índice de Figuras

Índice de Tablas

1. Introducción

1.1 Preámbulo

La autenticación biométrica es el estudio de los métodos automáticos basados en las características físicas o de comportamiento de las personas para el reconocimiento de las mismas. Existen numerosos enfoques, como el reconocimiento facial, de retina, de iris, de venas de la mano o la geometría de la palma de la mano. Sin embargo, la identificación mediante huellas dactilares sigue siendo uno de los métodos más fiables hoy en día.

Ni siquiera los gemelos comparten la misma huella dactilar, esto se debe a que los dibujos de la huella no están totalmente determinados por la información de los genes, que es la que comparten los gemelos idénticos. Las fuerzas intrauterinas, por ejemplo la que ejerce el flujo amniótico alrededor del feto durante el periodo de gestación, determinan el dibujo de las yemas de los dedos de pies y manos. Como cada hermano ocupa una posición diferente en el útero, las fuerzas intrauterinas son distintas y como resultado los dibujos de sus huellas dactilares, aunque se parezcan, son únicos.

La historia de la identificación por huellas dactilares se remonta a finales del siglo XIX y continúa siendo hoy en día una de las pruebas forenses más usadas. En 1980, el FBI creó el primer fichero informatizado de huellas de criminales. En 1983, el FBI creó el Centro Nacional de Información del Crimen, para compartir la información acerca de criminales entre los gobiernos federal y local.

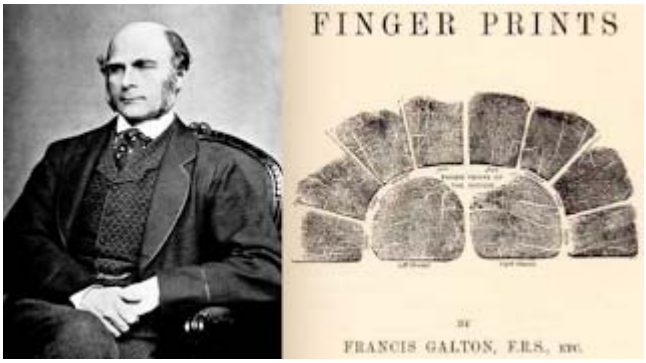


Fig. 1: Libro publicado por Francis Galton en 1892 llamado Finger Prints

A partir de este hecho, el FBI estandarizó los métodos de clasificación de las huellas dactilares. En 1989 todas las peticiones de reconocimiento de una huella se hacían por medio de un ordenador, acortando el tiempo de respuesta de 14 días a 1 día.

Hoy en día, la irrupción de la banca y el comercio electrónico, unida a la reducción en coste y tamaño de los sensores de huella dactilar, han hecho que la autenticación por huella dactilar sea una realidad en muchos dispositivos de la electrónica de consumo (ordenadores portátiles, SmartPhones, cerraduras electrónicas, etc.).

Fig. 2: lector de huella digital suprema biomini

Fig. 3: Terminal de pago Paytouch

Fig. 4: Cerradura Fingerprint Deadbolt lock de la empresa doljhink

Fig. 5: Arranque de moto Fingerprint Scan Bike de la empresa Greyp

1.2 Estado del arte

Las propiedades que debe cumplir en mayor o menor grado cualquier rasgo biométrico son [*Jain et al., 2004*] [14]:

- Universalidad: El rasgo biométrico existe en todos los individuos.
- Unicidad: El rasgo identifica unívocamente a cada individuo.
- Permanencia: El rasgo se mantiene invariable a corto plazo.
- Inmutabilidad: El rasgo se mantiene invariable a largo plazo.
- Mensurabilidad: El rasgo se puede medir.
- Rendimiento: El rasgo permite el reconocimiento con rapidez.
- Aceptabilidad: El rasgo es aceptado por la mayoría de la población
- Invulnerabilidad: El rasgo es robusto frente al acceso fraudulento.

La huella dactilar de una persona, es uno de los rasgos que cumple con estas propiedades en mayor grado, por lo que es el más utilizado en el reconocimiento biométrico.

El estudio de las huellas dactilares se puede enfocar desde dos vertientes, la autenticación y la identificación [*Maltoni et al., 2003*] [2]:

- En el proceso de autenticación los rasgos biométricos se comparan solamente con los de un patrón ya guardado, este proceso se conoce también como uno-para-uno (1:1). Este proceso implica conocer la identidad del individuo en el momento en el que se registra en el sistema con su huella dactilar a autenticar, por lo tanto, dicho individuo ha presentado algún tipo de credencial y durante el proceso de autenticación biométrica se comprueba si la huella de esa persona corresponde con la que hay almacenada en el sistema o no.

- En el proceso de identificación, los rasgos biométricos se comparan con los de un conjunto de patrones ya guardados. Este proceso se conoce también como uno-para-muchos (1: N). Este proceso implica no conocer la identidad presunta del individuo. La nueva muestra de datos biométricos es tomada del usuario y comparada una a una con los patrones ya existentes en el banco de datos registrados. El resultado de este proceso es la identidad del individuo, mientras que en el proceso de autenticación es un valor verdadero o falso.

El proceso de autenticación o verificación biométrica es más rápido que el de identificación biométrica, sobre todo cuando el número de usuarios (N) es elevado. Esto es debido a que la necesidad de procesamiento y comparaciones es más reducida en el proceso de autenticación. Por esta razón, es habitual usar autenticación cuando se quiere validar la identidad de un individuo desde un sistema con capacidad de procesamiento limitada o se quiere un proceso muy rápido.

Un ejemplo típico de sistema de identificación por huellas dactilares es el *Sistema Automático Integrado de Identificación por Huellas Dactilares* IAFIS (*Integrated Automated Fingerprint Identification System*), mantenido por el FBI y que contiene unas 70 millones de huellas dactilares. Estos sistemas requieren de costosos equipos en los que se ejecutan los algoritmos para la identificación de la persona propietaria de la huella introducida en el sistema.

http://www.fbi.gov/about-us/cjis/fingerprints_biometrics/iafis/iafis

Por otra parte se encuentran, como ya se ha mencionado, los dispositivos pertenecientes a la electrónica de consumo en los que se encuentra embebido un sensor de huella dactilar y el hardware/software asociado a él para realizar la autenticación de una persona o de un número de personas muy limitado. Entre estos dispositivos se encuentran las *SmartCards* (tarjetas inteligentes) [15], en las que se almacena la huella dactilar del usuario y por medio de un lector de huellas, se introduce la huella del usuario a autenticar en la tarjeta en la que se realiza el proceso de autenticación.

Otro ejemplo de este tipo de dispositivos, son los SmartPhones, que han apostado por introducir esta tecnología como elemento diferenciador. Así, últimamente están saliendo al mercado SmartPhones de gama alta que incorporan sensores de huella dactilar. Apple, en su *iPhone 5S (*Fig. 6*)* y Samsung en su *Galaxy S5* ya han lanzado SmartPhones al mercado con sensor de huella dactilar.

En estos casos, la utilidad de estos sensores se limita al desbloqueo de la pantalla y para realizar pequeños pagos al descargarse aplicaciones, en general de las propias tiendas online de los fabricantes, como AppleStore e iTunes de Apple o Galaxy apps de Samsung.

Fig. 6: iPhone 5S con sensor de huella integrado

A pesar del abaratamiento de estos sensores, muchos fabricantes no se han decidido a introducirlos en sus SmartPhones porque siguen suponiendo un coste elevado en relación al precio del SmartPhone. Además, al poco tiempo de su aparición, ya se descubrió que era posible "engañar" al sensor fabricando una huella de silicona obtenida de la huella dejada por el propio usuario en el cristal del sensor. Este hecho pone en entredicho la fiabilidad a la hora de utilizar este método como medio de pago. No obstante, a pesar de estos inconvenientes, sin duda esta tecnología todavía inmadura se acabará imponiendo y el pago seguro mediante autenticación por huella dactilar a través de un SmartPhone será algo cotidiano en poco tiempo. Todos los dispositivos de consumo que incorporan la funcionalidad de la autenticación por huella dactilar incluidos los SmartPhones, disponen como ya se ha mencionado de un sensor de huellas dedicado a leer la huella dactilar del usuario.

Los sensores de huella dactilar son básicamente de dos tipos:

- Los sensores ópticos (Fig. 7) y (Fig. 8), cuyo funcionamiento se basa en capturar la imagen de la huella mediante la luz reflejada por el dedo contra un CCD colocado debajo de un vidrio donde se apoya el dedo.

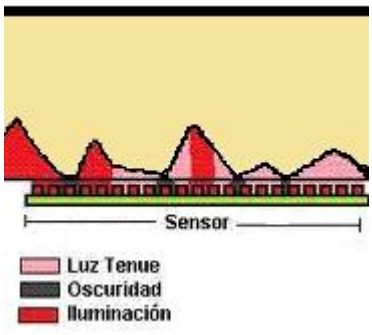


Fig. 7: Sensor Óptico

Fig. 8: Sensor óptico de la empresa Umanick

Los sensores de estado sólido, entre los que se encuentran los capacitivos (Fig. 9) y (Fig. 10), térmicos o piezoeléctricos. En estos sensores se mide alguna magnitud física como la variación de la capacidad, de la temperatura o de la presión, como en el caso de los piezoeléctricos.

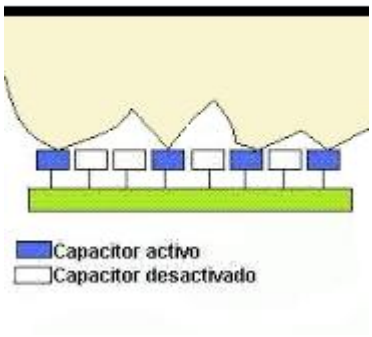


Fig. 9: Sensor Capacitivo

Fig. 10: Sensor Capacitivo Touch ID del iPhone 5S

1.3 Objeto del proyecto

1.3.1 Motivación

La idea de este TFM nació en el verano del año 2013 por la coincidencia de dos sucesos en el tiempo. El primero fue el hecho de estar pensando en desarrollar una aplicación para un teléfono móvil bajo el sistema operativo Android. Esta idea estaba alejada de mi experiencia profesional, en la cual desarrollo proyectos basados en microcontroladores y programación en lenguaje C.

La idea de adentrarme en la programación en lenguaje Java para Android y desarrollar una aplicación para teléfono móvil suponía un reto y una oportunidad de aprender y disfrutar con el TFM.

El segundo suceso fue el lanzamiento del primer SmartPhone con autenticación mediante sensor de huella dactilar incorporado.

La motivación de este TFM trata de responder a las siguientes preguntas:

- ¿Sería posible realizar la autenticación mediante huella dactilar tomando una imagen con la cámara de un SmartPhone?

- ¿Suponiendo que la imagen capturada de la huella dactilar tuviera calidad suficiente, sería posible implementar los algoritmos necesarios para llevar a cabo la autenticación en un SmartPhone?

- ¿Si fuera posible, no sería lógico que Apple hubiera explorado esta posibilidad, en vez de incorporar un sensor de huella con el incremento de coste que conlleva?

- ¿Habría potencia y capacidad de almacenamiento suficiente en un SmartPhone de ultima generación para ejecutar los algoritmos necesarios en un tiempo aceptable?

Tras una búsqueda sin éxito de alguna aplicación para SmartPhones que hiciera lo mismo, decidí que como mínimo sería un proyecto original.

El proyecto era interesante, divertido, original y complejo, por lo que parecía el proyecto mejor.

Así que la motivación de este TFM se puede resumir como un reto personal, una humilde aportación al mundo de la identificación biométrica y una forma de entender el TFM como algo importante con lo que disfrutar.

1.3.2 Objetivos

El objetivo del proyecto es desarrollar una aplicación para dispositivos móviles como SmartPhones o Tablets que sea capaz de realizar la autenticación de huellas dactilares de una persona mediante la captura de la imagen de la huella dactilar con la cámara de fotos del propio dispositivo, esto es, sin disponer de un sensor de huella dactilar dedicado.

La aplicación deberá ser capaz de realizar la autenticación de forma local, es decir, almacenando un número limitado de huellas en el propio dispositivo. En este sentido el *iPhone 5S* permite el registro de cinco usuarios diferentes. Se buscará una solución de compromiso ya que el número de huellas dependerá del tiempo de procesado necesario en el proceso de autenticación.

La aplicación también deberá ser capaz de realizar un proceso de "identificación" en el sentido explicado anteriormente. La aplicación deberá ser capaz de acceder a la base de huellas almacenadas en una ubicación remota y poder realizar la comparación de huellas dos a dos a elección del usuario de la aplicación mostrando en la pantalla del dispositivo el resultado de la comparación.

También deberá ser capaz de generar una hoja Excel con los resultados de la comparación de cada una de las huellas con todas las demás. Este proceso es muy importante ya que a partir de esa hoja Excel se podrán extraer conclusiones objetivas para el TFM.

En este proyecto se tendrán en cuenta factores críticos derivados del hecho de no disponer de un sensor de huellas dedicado. Estos factores

son el posicionado del dedo del usuario frente a la cámara, la inmunidad a la luz ambiente, el procesado de la imagen obtenida y el tiempo de procesado total empleado en la autenticación.

Se llegará a una solución de compromiso entre fiabilidad y tiempo de procesado, para lo cual se intentará optimizar el algoritmo empleado en todo el proceso, de modo que la aplicación pueda ser "usada" al menos en forma de demostración de una manera ágil.

El objetivo final es conseguir una tasa de falsas aceptaciones *FAR (False Aceptante Rate)* y falsos rechazos *FRR (False Rejection Rate)*, así como el *EER* del sistema (Equal Error Rate), que nos permita evaluar la fiabilidad del mismo.

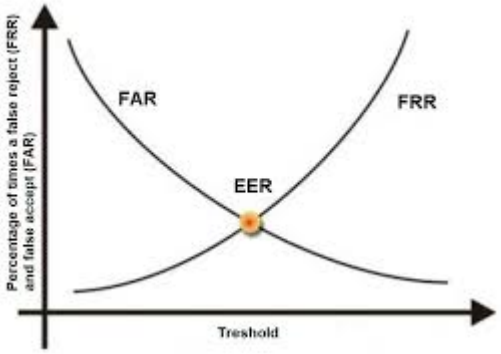


Fig. 11: FAR vs FRR

Si el umbral (Treshold) de decisión (Fig. 11), es decir, el valor que representa la coincidencia entre dos huellas, se fija en un valor muy bajo, el FAR será muy elevado, lo que significa que se aceptarán muchos falsos usuarios y se rechazaran muy pocos usuarios auténticos. Por el contrario, si el umbral se fija en un valor muy alto, el FRR será muy alto, lo que significa que se rechazaran muchos usuarios auténticos aunque se aceptaran muy pocos usuarios falsos.

Hay un punto en el que se cruzan las dos curvas, llamado EER (Equal Error Rate) o punto de igual error. Este punto suele usarse para caracterizar con un único número el rendimiento de un sistema biométrico, lo cual no significa que el sistema tenga que trabajar en ese punto.

En aplicaciones de alta seguridad (control de accesos), el punto de trabajo suele situarse en valores bajos de FAR, para evitar que accedan impostores, a costa de tener alta FRR. Por el contrario, en aplicaciones forenses se trabaja en baja FRR para no perder individuos buscados, a

costa de una alta FAR. Las aplicaciones civiles suelen trabajar en un punto intermedio.

Para lograr este objetivo, se creara una base de huellas, que consistirá en diez huellas del mismo dedo de un total de 14 personas. Estas huellas servirán para hacer una comparación aleatoria entre ellas para simular lo que ocurriría si se introdujera una nueva huella y se comparara con toda la base de huellas existentes.

También es un objetivo de este TFM conseguir que la autenticación local sea lo más funcional posible y que su grado de fiabilidad sea aceptable. Por ultimo, también es un objetivo de este TFM que la aplicación sea estable y fácil de usar. Cuidando aspectos como la visualización de las imágenes así como la presentación de los datos al usuario.

2. Primeros desarrollos del proyecto

Antes de poder desarrollar la aplicación que sería ejecutada en el teléfono, se capturaron imágenes con la cámara del mismo para poder ser tratadas con MatLab. De este modo se pretendía hacer un estudio previo de los pasos a dar con la imagen capturada para posteriormente decidir que algoritmo implementar en la aplicación que correría en el teléfono.

En todo sistema de autenticación o identificación de huellas dactilares se pueden distinguir varias fases [1]:

Preprocesado (Preprocessing)

Realce (Enhancement)

Extracción de características (Feature extraction)

Comparación (Matching)

Para esta primera etapa del proyecto se han usado los algoritmos de Peter Kovesi, obtenidos de su sitio Web de la Universidad de Australia, dedicado a difundir conocimiento acerca de visión por computador y procesado de imágenes con MatLab.

P. D. Kovesi.

MATLAB and Octave Functions for Computer Vision and Image Processing

Centre for Exploration Targeting

School of Earth and Environment

The University of Western Australia

Available from: http://www.csse.uwa.edu.au/~pk/research/matlabfns/

En la Fig. 12, tenemos una de las imágenes capturadas al inicio del proyecto y sobre la que se aplicaran los algoritmos antes mencionados.

Fig. 12: Una de las imágenes (640x480 píxeles) capturadas al inicio del proyecto. Fue capturada con un Tablet Samsung Galaxy Tab 10.1.

2.1 Preprocesado (Preprocessing)

Los sensores de huella comerciales (Ver Anexo A), suelen entregar una imagen de la huella en escala de grises. Generalmente se representan las crestas con tonos más oscuros, mientras que a los valles se les asigna tonos claros. Esta forma de representar la imagen de una huella viene influenciada por los primeros sensores ópticos. En nuestro caso partimos de una imagen en color (Fig. 12), tendremos que realizar además una conversión a escala de grises en la primera fase (preprocesado).

2.1.1 Conversión a escala de grises

En primer lugar se efectúa una conversión a escala de grises (Fig. 13), ya que como hemos dicho, nos interesa el contraste entre crestas y valles y en este caso el color no nos aporta información relevante. Esta conversión se realiza a través de la función ***rgb2gray*** de MatLab.

2.1.2 Ecualización de Histograma

Mediante la función de MatLab ***adapthisteq*** se realiza una ecualización adaptativa del histograma[2] de la imagen (Fig. 14), que realza el contraste de la imagen dividiéndola en pequeños bloques y posteriormente realiza la interpolación bilineal de los bloques vecinos para eliminar los posibles bordes creados artificialmente. Con esto se consigue que todos los píxeles de un mismo nivel de gris se transformen a otro nivel de gris, separando lo máximo posible cada nivel dentro del histograma, maximizando el contraste de una imagen sin perder información de tipo estructural.

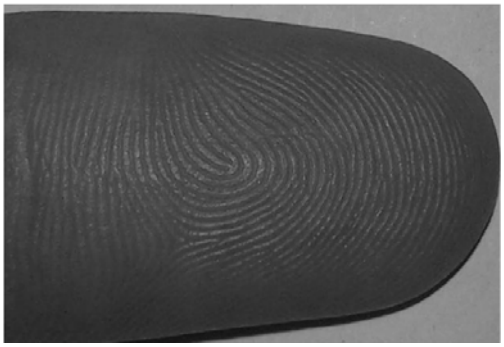


Fig. 13: Conversión a escala de grises

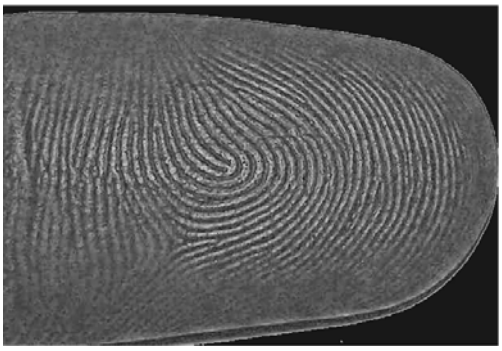


Fig. 14: Ecualización adaptativa de histograma

2.2 Realce (Enhancement)

En esta fase se consigue realzar el contraste entre las crestas y los valles. Esto se consigue mediante la aplicación de filtros de Gabor.

La aplicación de estos filtros se basa en determinar de manera local, es decir, dividiendo la imagen en bloques, la orientación de las crestas y la frecuencia espacial de las mismas, es decir, la inversa del número de crestas que cortan un segmento de una unidad de longitud perpendicular a su dirección.

Una vez que se ha determinado la orientación y la frecuencia espacial de las crestas para cada bloque, se genera un filtro de Gabor por cada bloque y se filtra la imagen. El diagrama [1] (Fig. 15), explica los procesos llevados a cabo en la etapa de realce.

[2] un **histograma** es una representación gráfica de una variable en forma de barras, donde la superficie de cada barra es proporcional a la frecuencia de los valores representados

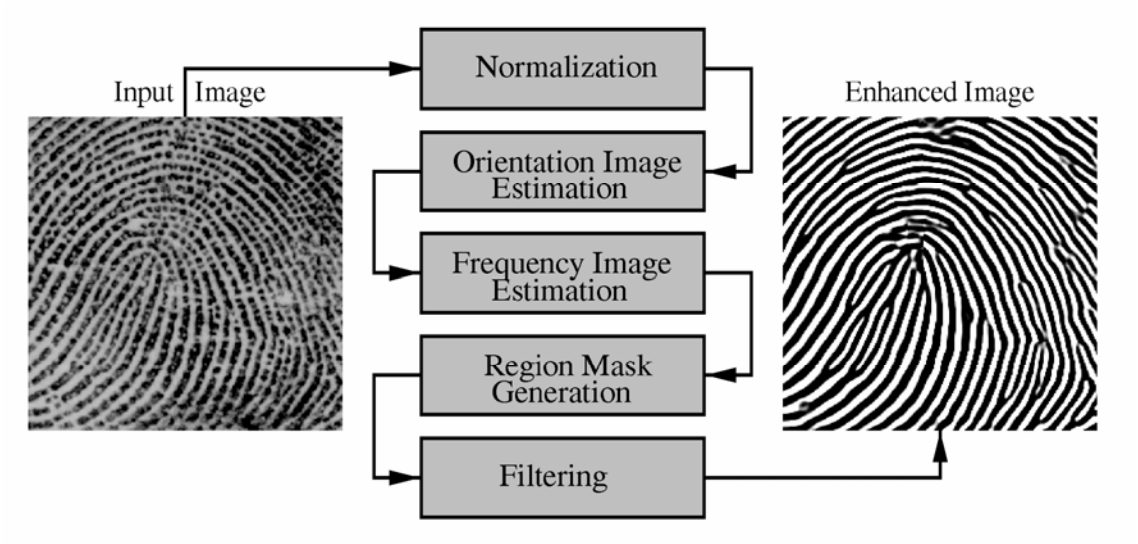


Fig. 15: Diagrama de flujo del algoritmo de realce. Fuente: [*Hong et al., 1998*] [1]

2.2.1 Normalización

El proceso de normalización se realiza para mejorar los pasos siguientes. Es un proceso que se realiza con cada uno de los píxeles.

El proceso, básicamente divide la imagen en bloques y calcula la media y la varianza de la imagen según la ecuación (1) [*Hong et al., 1998*] [1].

$$G(i,j)=\begin{cases} \mathrm{M}_0+\sqrt{\dfrac{VAR_0(\mathrm{I}(i,j)-\mathrm{M})^2}{VAR}} \Rightarrow \mathrm{I}(i,j)>\mathrm{M} \\ \mathrm{M}_0-\sqrt{\dfrac{VAR_0(\mathrm{I}(i,j)-\mathrm{M})^2}{VAR}} \Rightarrow othrerwise \end{cases} \quad (1)$$

Donde I(i, j), denota el valor de nivel de gris del píxel (i, j) de la imagen I. M y VAR denotan la varianza y la media estimada de I y G(i, j) denota el valor normalizado del píxel (i, j).

M_0 y VAR_0, son los valores de media y varianza deseados, respectivamente. La normalización (Fig. 16) reduce las variaciones de los valores en escala de grises a lo largo de las crestas y los valles, lo cual facilita los procesos siguientes.

Fig. 16: Normalización de la imagen

2.2.2 Estimación local de la orientación y frecuencia

Partiendo de la imagen normalizada del proceso anterior, se lleva a cabo el cálculo de la orientación y frecuencia de las crestas.

La estimación de la orientación de las crestas y de la frecuencia (Fig. 17) de las mismas es necesario, como ya se explicó, para la generación de los filtros de Gabor. Esta estimación se hace localmente, es decir, dividiendo la imagen en bloques en los que se estima la orientación de las crestas calculando el gradiente G_x y G_y de cada pixel dentro del bloque y calculando posteriormente el arco tangente G_x / G_y. La orientación del bloque será la orientación dominante de todos los píxeles del bloque.

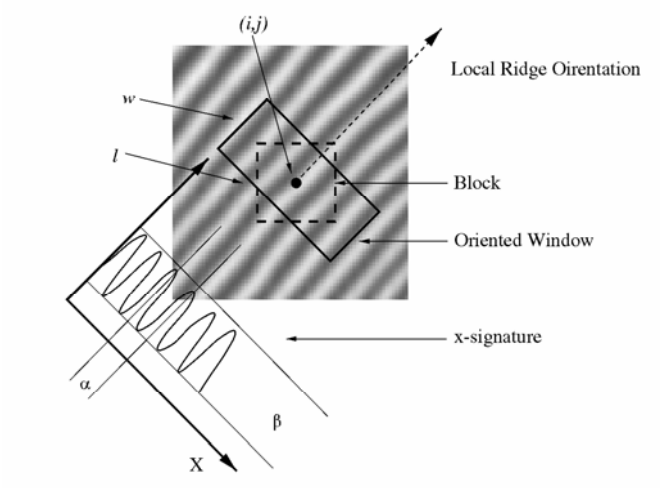


Fig. 17: Orientación de las crestas y frecuencia [1]

La frecuencia de las crestas es el número de crestas que cortan un segmento de una unidad de longitud perpendicular a su dirección.

2.2.3 Filtro de Gabor

Con la imagen dividida en bloques y teniendo bien definidos la frecuencia y orientación de las crestas y los valles, se puede eliminar el ruido de la imagen mediante la aplicación de filtros "sintonizados" a la frecuencia y orientación correspondiente de cada bloque.

Los filtros de Gabor (Fig. 18) tienen propiedades de frecuencia y orientación selectivas por lo que son adecuados en este proceso.

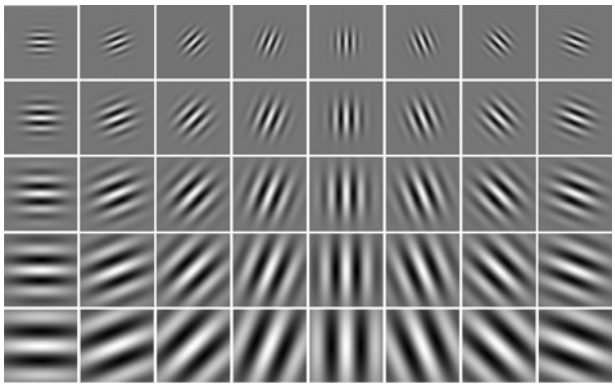


Fig. 18: Representación gráfica de 40 filtros de Gabor. Imagen extraída de [3].

En la imagen (Fig. 19), se observa la imagen de nuestra huella una vez finalizada la fase de realce. Posteriormente se ha sometido a un proceso de binarización, que consiste en convertir sus píxeles en blanco o negro a través de un umbral de decisión. El resultado obtenido será la entrada para la fase de extracción de características.

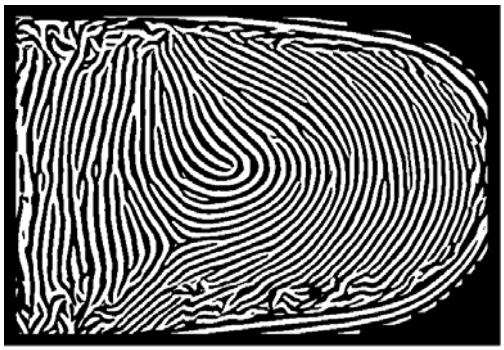


Fig. 19: Realce de la huella mediante filtros de Gabor

2.3 Extracción de características (Feature extraction)

En esta fase se van a extraer la minucias de la huella (Ver Anexo A). Normalmente se procede primero a someter a la imagen a un proceso adelgazamiento de las crestas (líneas blancas). Este proceso se denomina esqueletización.

2.3.1 Esqueletización

El proceso de esqueletización se consigue en MatLab a través de la función:

bwmorph(Imagen,'skel',Inf)

El parámetro "Inf" significa el número de veces que se aplica el algoritmo, en este caso se aplica hasta que la imagen no sufre cambios.

Una vez que la imagen esta esqueletizada, se puede proceder a la siguiente etapa de extracción de minucias propiamente dicha.

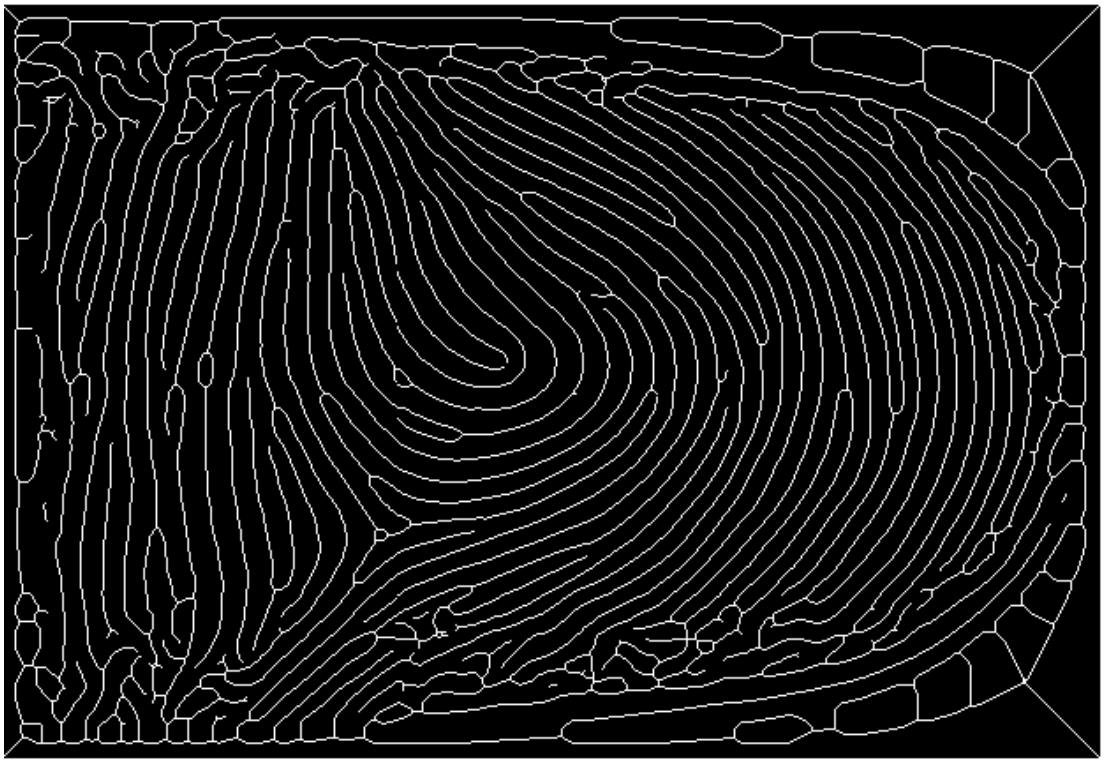


Fig. 20: Imagen de la huella esqueletizada (aumentada)

2.3.2 Extracción de Minucias

El método de comparación basado en la extracción de minucias no es el único, existen también algoritmos basados en correlación de píxeles, basados en patrones de crestas o texturas. En esta primera aproximación que hemos hecho en nuestro proyecto, hemos optado por implementar en MatLab la extracción de minucias, ya que es el método más extendido. La extracción de minucias, consiste en detectar las bifurcaciones y las terminaciones de las crestas (Ver Anexo A), esto se consigue recorriendo la imagen con una mascara de 3x3 píxeles y aplicando un algoritmo para detectar cada característica, de (2) se extraen las terminaciones o *endpoints* y de (3) se extraen las bifurcaciones.

P_3	P_2	P_1
P_4	M	P_0
P_5	P_6	P_7

$$C_N = \sum_{k=0}^{7} |P_{k+1} - P| = 2, \qquad M = endpoint \tag{2}$$

$$C_N = \sum_{k=0}^{7} |P_{k+1} - P| = 6, \qquad M = bifurcation \tag{3}$$

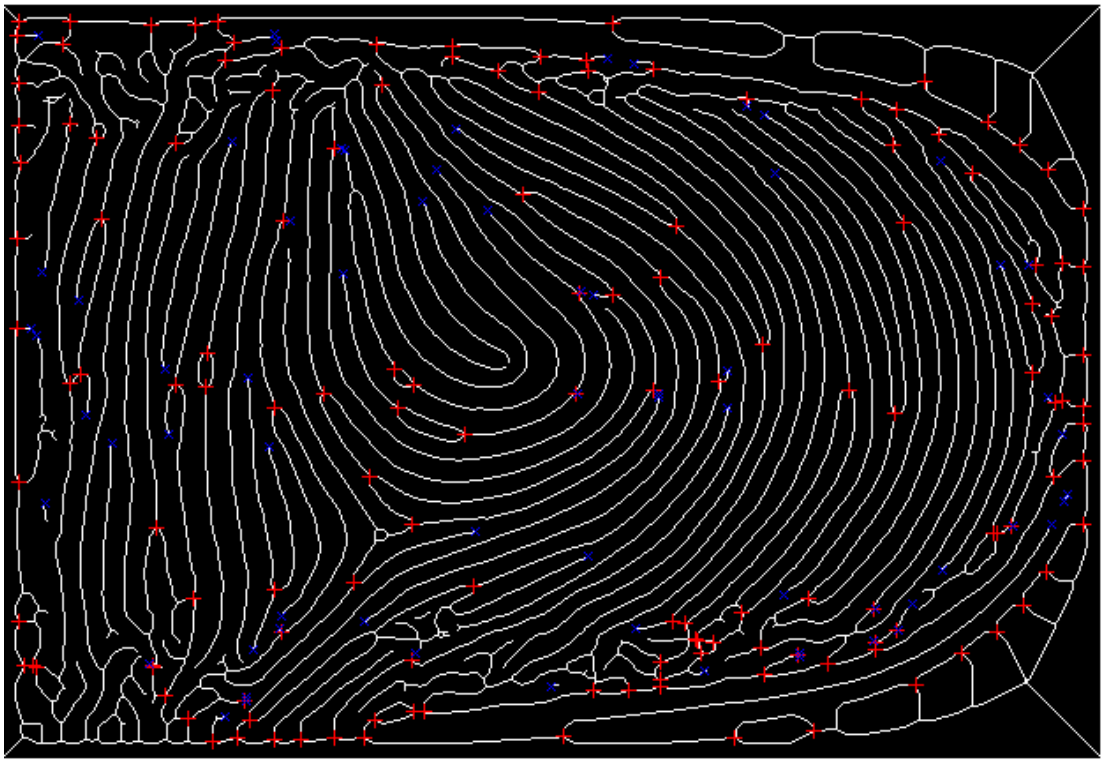


Fig. 21: Extracción de minucias; en rojo las bifurcaciones y en azul las terminaciones (aumentada)

2.4 Conclusiones

En este capitulo se ha presentado el tratamiento previo realizado a una de las imágenes de una huella capturada con la cámara de un dispositivo móvil. Este tratamiento previo realizado con MatLab, se hizo para determinar por un lado, que las imágenes capturadas eran susceptibles de ser sometidas a un tratamiento similar al realizado cuando estas proceden de un lector de huella dactilar y por otro lado determinar que procesos y algoritmos eran necesarios aplicar para llegar a obtener una imagen lista para poder realizar el proceso de reconocimiento o matching.

En este sentido MatLab es una potente herramienta que sirve de gran ayuda en estas fases previas del proyecto.

Como conclusión podemos decir, a la vista de los resultados obtenidos, que las imágenes capturadas con la cámara del teléfono pueden ser tratadas consiguiendo unos resultados satisfactorios.

En el siguiente capitulo, veremos como se implementan estos algoritmos en la aplicación que correrá en el dispositivo Android.

Veremos también las consideraciones en cuanto a la limitación de capacidad de proceso, memoria, etc. Así como también la posibilidad de

implementar todo el proceso visto en este capitulo o bien eliminar o cambiar alguna parte como solución de compromiso en cuanto a manejo de la aplicación, velocidad de proceso, fiabilidad, recursos, etc.

La parte del reconocimiento de las huellas (*Matching*), no la hemos tratado en este punto ya que es muy dependiente de la extracción de características elegida y aunque aquí se ha realizado en MatLab la extracción de minucias, es una técnica que requiere una gran cantidad de procesado y no sabemos si podremos implementarla aún en la aplicación. Quedará por tanto pendiente de ser desarrollada cuando se implementen los algoritmos en la aplicación.

3. Desarrollo de la aplicación bajo sistema operativo Android

En el presente capitulo, se explicará como se han transportado al dispositivo móvil los algoritmos implementados en MatLab en el capitulo anterior. También se explicara y justificara la solución adoptada para la fase de reconocimiento (*Matching*).

3.3 Implementación del algoritmo de autenticación

En el presente capitulo veremos como se ha implementado el algoritmo de tratamiento de imagen. Nos centraremos aquí únicamente en la parte de implementación en el dispositivo Android del tratamiento de la imagen, al ser este el punto más importante y el resto de la aplicación se vera en el Anexo B.

Veremos por un lado, como se han transportado los algoritmos vistos en el capítulo 2 desarrollados en Matlab hasta conseguir la esqueletización y extracción de minucias.

Por otro lado se abordará la solución adoptada para el algoritmo de comparación o *Matching.*

El tratamiento de la imagen se ha realizado casi por completo con la biblioteca OpenCV por lo que haremos un repaso a sus principales características.

3.3.1 La biblioteca OpenCV

OpenCV es una biblioteca de visión artificial, de uso libre que fue desarrollada originalmente por Intel en el año 1999. OpenCV es multiplataforma, habiendo versiones para Linux, Mac OS X y Windows. La biblioteca contiene más de 500 funciones en áreas como el procesado de visión, reconocimiento de objetos, calibración de cámaras, visión robótica, etc.

Con ella se puede trabajar con elementos de segmentación, detección de contornos, cálculos de etiquetado, seguimiento (*tracking*) o incluso reconocimiento facial. Una de las características que la hacen especialmente interesante es su apartado dedicado exclusivamente al reconocimiento de patrones.

3.3.2 Algoritmo

En este capítulo vamos a ver con detalle los pasos seguidos para implementar en nuestra aplicación móvil, los algoritmos equivalentes usados en MatLab:

- Preprocesado (Preprocessing)
- Realce (Enhancement)
- Extracción de características (Feature extraction)
- Posteriormente se abordará la fase de comparación o Matching.

En el dispositivo móvil, el núcleo de la aplicación es sin duda el tratamiento de la imagen. Una vez capturado el frame[3] que va a ser objeto de la comparación con otro para verificar su semejanza, se obtiene una imagen (Fig. 22) en color. Este frame es capturado con el máximo zoom disponible, la sensibilidad (ISO) en un valor de 100 y el autofocus en modo continuo enfocando en el rectángulo rojo de la figura que será el área de interés con la que nos quedaremos una vez la imagen sea recortada.

Con estos parámetros, fijados empíricamente, se consigue una baja sensibilidad a los cambios en la luz ambiente. Esto se consigue al iluminar con el Flash de la cámara. Esta luz potente y muy cercana hace que la imagen este sobre iluminada pero de una forma constante independiente de la luz ambiente. Para compensar este exceso de iluminación, se ajusta la sensibilidad (ISO) a un valor bajo como 100 y además se ajusta también la compensación de la exposición a 1/20 del valor máximo de la cámara.

Con el zoom a su máximo valor, tenemos el frame con el área de la huella dactilar prácticamente "encajado" en el área disponible.

En la figura se ve una "guía" de color azul que sirve para que el usuario posicione el dedo en el plano X-Y, la altura o distancia a la cámara queda fijada por el "encaje" del dedo hasta ocupar toda el área interior de la "guía".

El recorte de la imagen para quedarnos con el interior del rectángulo rojo se implementó ya que, se vio que debido a la curvatura del dedo, la luz no incide por igual en toda la superficie y en muchas ocasiones parte de la huella queda sin información como se aprecia en la parte superior de la figura

[3] Un frame, fotograma o cuadro es una imagen particular dentro de una sucesión de imágenes que componen una animación.

Fig. 22: Imagen original capturada con el dispositivo

Este hecho no se presenta en las imágenes de huellas dactilares capturadas con sensores dedicados, ya que en estos se apoya el dedo en una superficie transparente plana y toda la huella queda en el mismo plano.

El hecho de que parte de la huella no quede bien expuesta a la luz, en la que no se aprecian las características de la huella (crestas y valles), hace innecesaria su captura y además se aligera la cantidad de información a tratar.

Una vez que tenemos nuestra huella posicionada, iluminada y enfocada, estamos en condiciones de hacer la transformación a escala de grises que, como se vio en el capitulo 2, es el primer paso en la fase de Preprocesado. La conversión a escala de grises se realiza con la función *cvtColor* de OpenCV:

```
Imgproc.cvtColor(mRgba, mIntermediateMat, Imgproc.COLOR_RGBA2GRAY, CvType.CV_8U);
```

Con esta función se consigue transformar la imagen a escala de grises con una profundidad de bit de 255, es decir, 0 corresponde con el color negro y 255 con el color blanco.

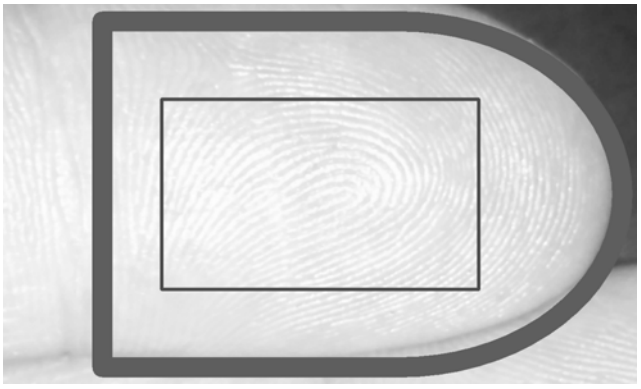


Fig. 23: Imagen convertida a escala de grises

Posteriormente se procede al recorte de la imagen para quedarnos con el área de interés antes comentada (Fig. 24). Para conseguir quedarnos con la imagen del interior del rectángulo, simplemente nos quedamos con una submatriz de la imagen original, utilizando para ello la función *submat* de OpenCV:

```
mIntermediateMat = mIntermediateMat.submat((mIntermediateMat.rows() / 2-mIntermediateMat.rows() / 5),
                   (mIntermediateMat.rows() / 2 + mIntermediateMat.rows() / 5),
                   (mIntermediateMat.cols() / 2-mIntermediateMat.cols() /5),
                   (mIntermediateMat.cols()  / 2+mIntermediateMat.cols() /5));
```


Fig. 24: Imagen recortada

Una vez tenemos la imagen recortada, procedemos a realizar la ecualización adaptativa de histograma, tal como hicimos en el capitulo 2 con MatLab, pero esta vez usando la biblioteca OpenCV, lo realizaremos mediante la función *equalizeHist (*Fig. 25*).*

```
Imgproc.equalizeHist(mIntermediateMat, mIntermediateMat);
```

Esta función realiza la ecualización de histograma de una imagen en escala de grises. El algoritmo calcula el histograma de la imagen (H), normaliza el histograma de manera que la suma de los distintos niveles de grises sea 255, posteriormente computa la integral del histograma H'

(4) y transforma la imagen original usando H' como una *lookup table*[4] (5).

H'_i = sum(by: 0 <= j < i) H(j) (4) dst(x,y) = H'(src(x,y)) (5)

El algoritmo normaliza el brillo e incrementa el contraste de la imagen.

Fig. 25: Imagen ecualizada

Tras esta etapa, procedemos a realizar una binarización de la imagen. En este caso utilizaremos La función de OpenCV *adaptativeThreshold*.

```
int maxValue = 255;
int blockSize = 41;
int meanOffset = 4;
Imgproc.adaptiveThreshold(mIntermediateMat, mIntermediateMat,
                          maxValue, Imgproc.ADAPTIVE_THRESH_GAUSSIAN_C,
                          Imgproc.THRESH_BINARY_INV, blockSize, meanOffset);
```

[4] Una lookup table (del inglés "tabla de consulta") es, en informática, una estructura de datos, normalmente un vector o un Vector asociativo, que se usa para sustituir una rutina de computación con una simple indexación de los vectores

Esta función transforma una imagen en escala de grises en otra binaria. La operación se realiza para cada pixel individualmente.

En este caso, el parámetro *THRESH_BINARY_INV*, indica que el algoritmo asigna el valor 0 al pixel que se esta computando si el valor del pixel supera un cierto umbral, que es calculado individualmente para cada pixel. El método *ADAPTIVE_THRESH_GAUSSIAN_C*, calcula el umbral de cada pixel como la suma ponderada (correlación cruzada con una gaussiana) menos el *meanOffset*, de los píxeles vecinos en una matriz de tamaño *blocksize x blocksize.*

El valor *blocksize* y *meanoffset* fueron seleccionados empíricamente.

Fig. 26: Imagen binarizada

En la siguiente fase vamos a eliminar el ruido presente en la imagen mediante la aplicación de filtros de Gabor.

La función de Gabor bidimensional es un oscilador armónico, compuesto por una onda sinusoidal plana a una frecuencia y orientación particular, dentro de una envolvente gaussiana [1].

En nuestro caso, se han implementado 8 filtrados de Gabor con distintas orientaciones y posteriormente se han sumado para obtener la respuesta total.

Para aplicar el filtrado de Gabor, primero hay que definir el núcleo o kernel del filtro, que será la matriz usada para la correlación con la imagen.

La función de OpenCV *getGaborKernel* genera los coeficientes a partir de los parámetros que le pasemos:

```
Mat Kernel = Imgproc.getGaborKernel(mask, sigma, theta1, lambd, gamma, psi, CvType.CV_64F);
```

Donde:

Mask: tamaño de la ventana del filtro.

Sigma (σ): varianza de la función, depende de la longitud de onda, siendo la relación 0.59*λ.

Lambd(λ): longitud de onda del factor coseno medido en píxeles. Este parámetro es la distancia entre crestas, que se traduce en la frecuencia de la onda sinusoidal del filtro.

Theta1(θ): orientación medida en grados con un rango de 0º a 360º. La orientación de las crestas y valles de la huella y que sirve para orientar el filtro.

ψ: desfase en grados con un rango entre -180º y 180º.

γ(gamma): aspecto, determina la elipticidad del núcleo.

Los parámetros seleccionados en nuestro caso son:

```
Size mask = new Size(3,3);
double sigma = 5.9;
double theta1 = 0;
double theta2 = Math.PI/4;
double theta3 = Math.PI/12;
double theta4 = Math.PI*0.75;
double theta5 = Math.PI;
double theta6 = Math.PI*1.25;
double theta7 = Math.PI*1.5;
double theta8 = Math.PI*1.75;
double lambd = 10;
double gamma = 1;
double psi = Math.PI;
```

Una vez obtenido cada uno de los ocho kernel con todos los parámetros comunes excepto la orientación, que es propia de cada filtro, se procede a correlacionar cada uno de los filtros con la imagen mediante la función de OpenCV *filter2D*:

```
Imgproc.filter2D(mIntermediateMat, gabor1, -1, Kernel);
Imgproc.filter2D(mIntermediateMat, gabor2, -1, Kernel);
Imgproc.filter2D(mIntermediateMat, gabor3, -1, Kernel);
Imgproc.filter2D(mIntermediateMat, gabor4, -1, Kernel);
Imgproc.filter2D(mIntermediateMat, gabor5, -1, Kernel);
Imgproc.filter2D(mIntermediateMat, gabor6, -1, Kernel);
Imgproc.filter2D(mIntermediateMat, gabor7, -1, Kernel);
Imgproc.filter2D(mIntermediateMat, gabor8, -1, Kernel);
```

Posteriormente se suman las respuestas obtenidas en cada filtrado para obtener una única imagen.

```
Core.addWeighted(gabor, 1, gabor1, 1, 0, gabor);
Core.addWeighted(gabor, 1, gabor2, 1, 0, gabor);
Core.addWeighted(gabor, 1, gabor3, 1, 0, gabor);
Core.addWeighted(gabor, 1, gabor4, 1, 0, gabor);
Core.addWeighted(gabor, 1, gabor5, 1, 0, gabor);
Core.addWeighted(gabor, 1, gabor6, 1, 0, gabor);
Core.addWeighted(gabor, 1, gabor7, 1, 0, gabor);
Core.addWeighted(gabor, 1, gabor8, 1, 0, gabor);
```

La imagen de salida (Fig. 27), está ya realzada y con el ruido eliminado.

Fig. 27: Imagen obtenida aplicando filtros de Gabor con mask = 3x3

En función del tamaño de la ventana del kernel, obtendríamos distintos resultados en nuestra imagen:

Fig. 28: mask = 2x2

Fig. 29: mask = 4x4

Fig. 30: mask = 5x5

Fig. 31: mask = 6x6

Fig. 32: mask = 7x7

Fig. 33: mask = 8x8

Con este ultimo proceso, habría acabado la fase de realce y pasaríamos a implementar la extracción de características.

Vamos a partir de dos imágenes distintas del mismo dedo de la misma persona. Partimos de la situación de la etapa anterior, es decir, las imágenes están procesadas y realzadas (Fig. 34) y (Fig. 36).

Mediante dos algoritmos que explicaremos a continuación, se obtiene por un lado la esqueletización de la imagen, es decir, dejar las líneas blancas de 1 pixel de grosor y posteriormente, se procede con otro algoritmo a extraer las minucias. En rojo se pintan las bifurcaciones y en azul las terminaciones (Fig. 35) y (Fig. 37).

El algoritmo de skeletización [*Guo et al., 1989*] [4], se ha implementado de la página:

http://answers.opencv.org/question/3207/what-is-a-good-thinning-algorithm-for-getting-the/

ya que en OpenCV no existe la función bwmorph (Imagen,'skel',Inf) que utilizamos en el capítulo 2.3.1.

Fig. 34: Imagen 1 segmentada y realzada

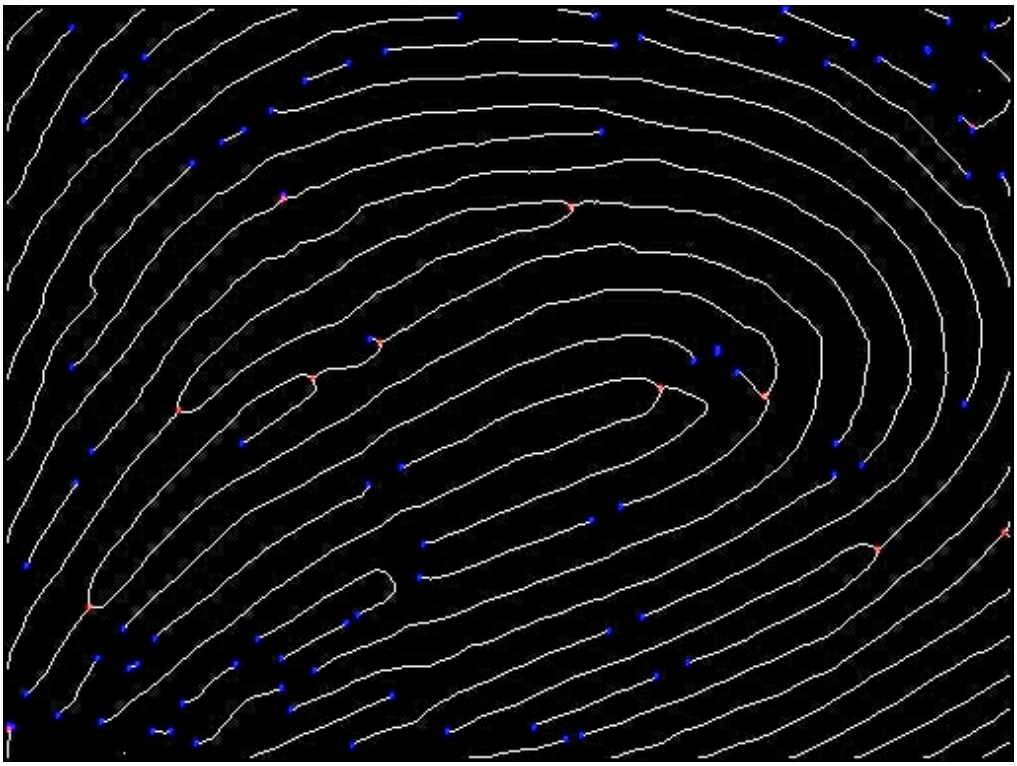


Fig. 35: Imagen 1 esqueletizada con minucias extraídas. (aumentada)

Fig. 36: Imagen 2 segmentada y realzada

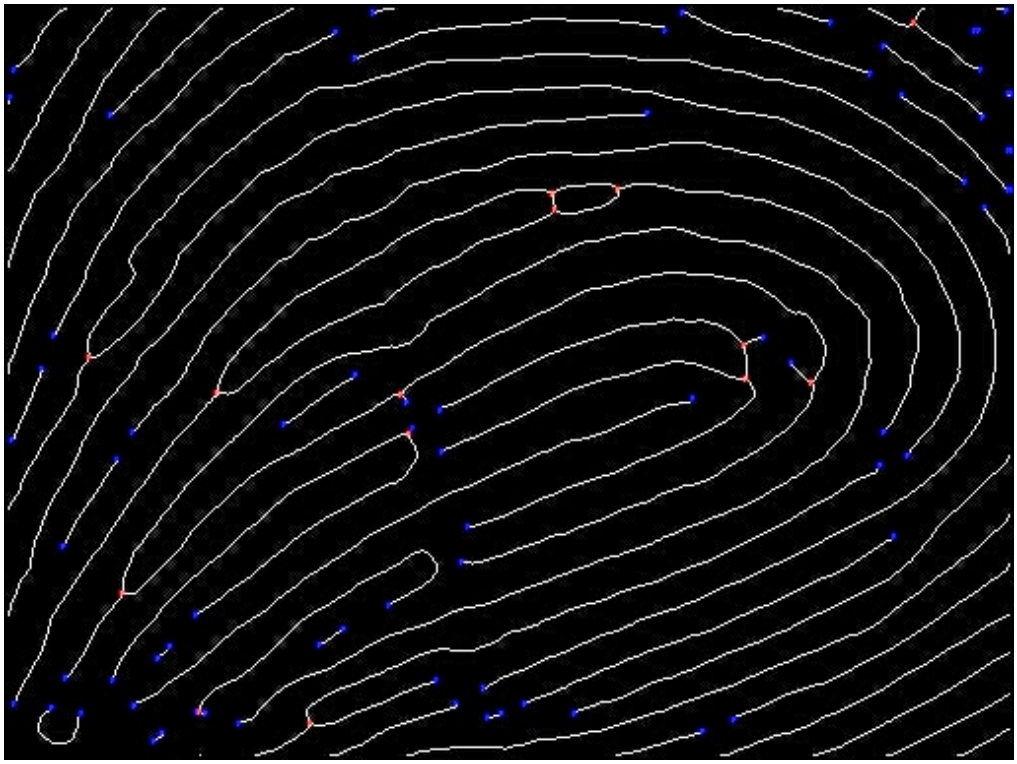


Fig. 37: Imagen 2.esqueletizada con minucias extraídas. (aumentada)

En las siguientes figuras, vemos otro ejemplo con otra huella (Fig. 38), (Fig. 39), (Fig. 40) y (Fig. 41).

Fig. 38: Imagen 3 segmentada y realzada

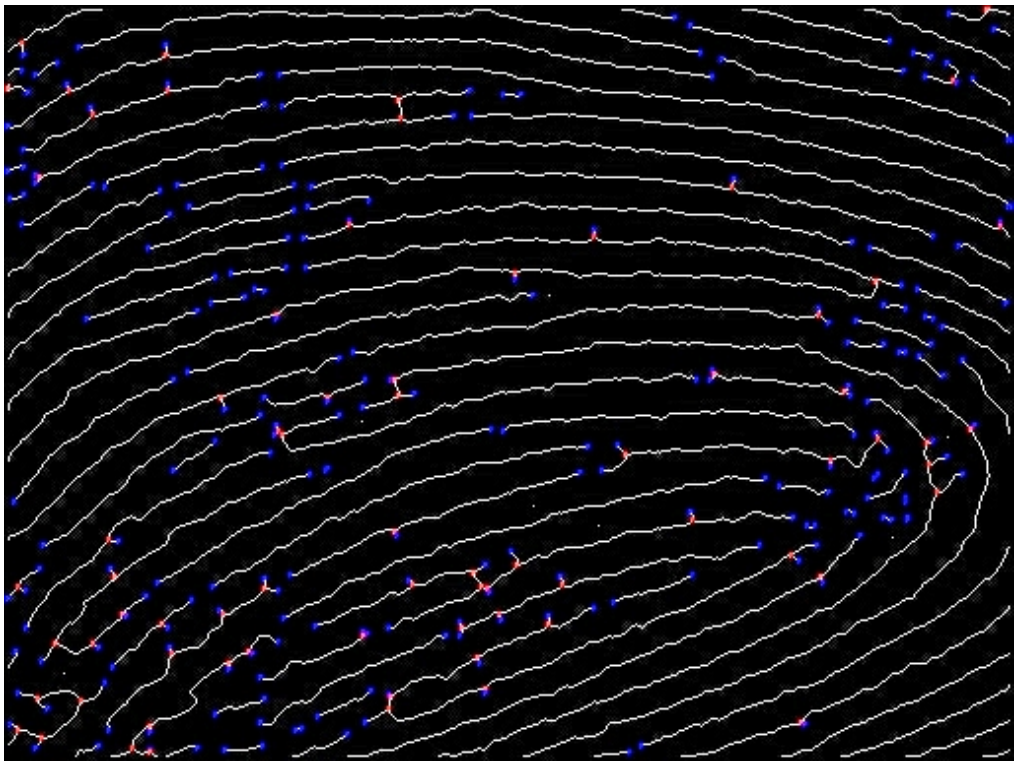


Fig. 39: Imagen 3 esqueletizada con minucias extraídas. (aumentada)

Fig. 40: Imagen 4 segmentada y realzada

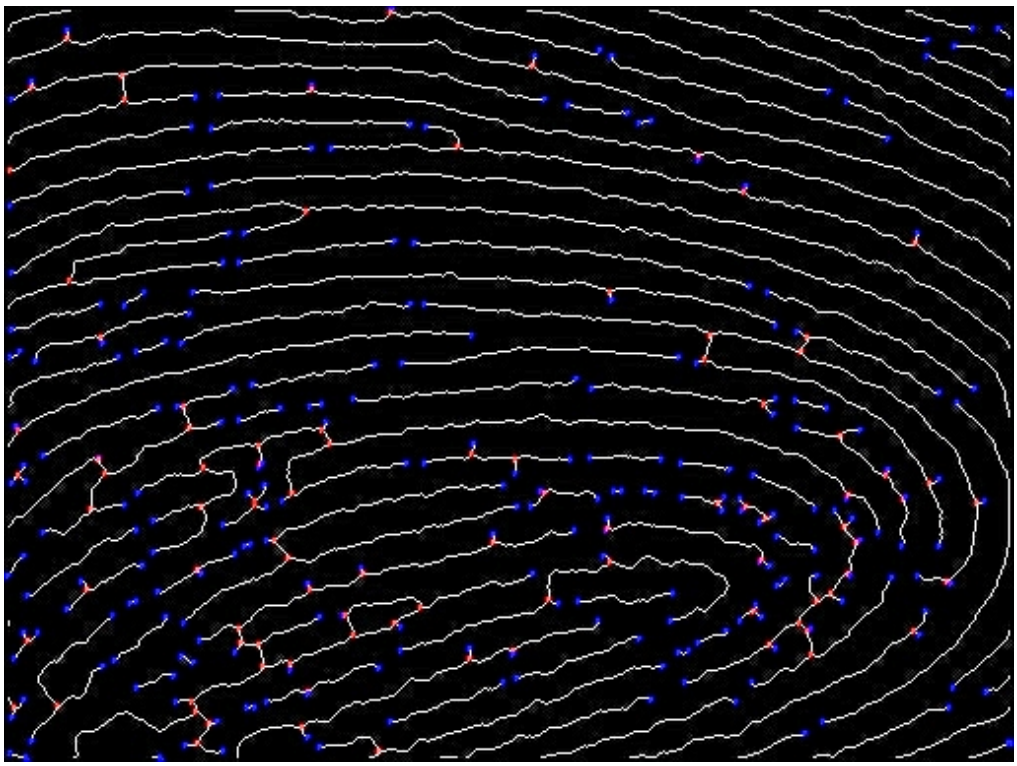


Fig. 41: Imagen 4 esqueletizada con minucias extraídas. (aumentada)

En las imágenes anteriores, se aprecia como la extracción de minucias no ha quedado tan bien definida como en el ejemplo anterior, ya que el proceso anterior no ha arrojado una esqueletización de tanta calidad.

En el siguiente ejemplo (Fig. 43), (Fig. 44) (Fig. 45) y (Fig. 46), se observa que la imagen esta muy degradada a la salida del filtro de Gabor, por lo que la esqueletización y posterior extracción de minucias realmente es de muy baja calidad. En la imagen (Fig. 42), se observa que esta huella ya en su estado original, presenta una carencia de características (Crestas y valles) en buena parte de la zona a detectar. Este hecho se agrava con la técnica utilizada de captura de imágenes mediante la cámara del dispositivo, ya que la carencia de relieve en alguna zona, aumenta el reflejo de la luz, degradándose más aún la imagen obtenida.

Fig. 42: Imagen original

Fig. 43: Imagen 5 segmentada y realzada

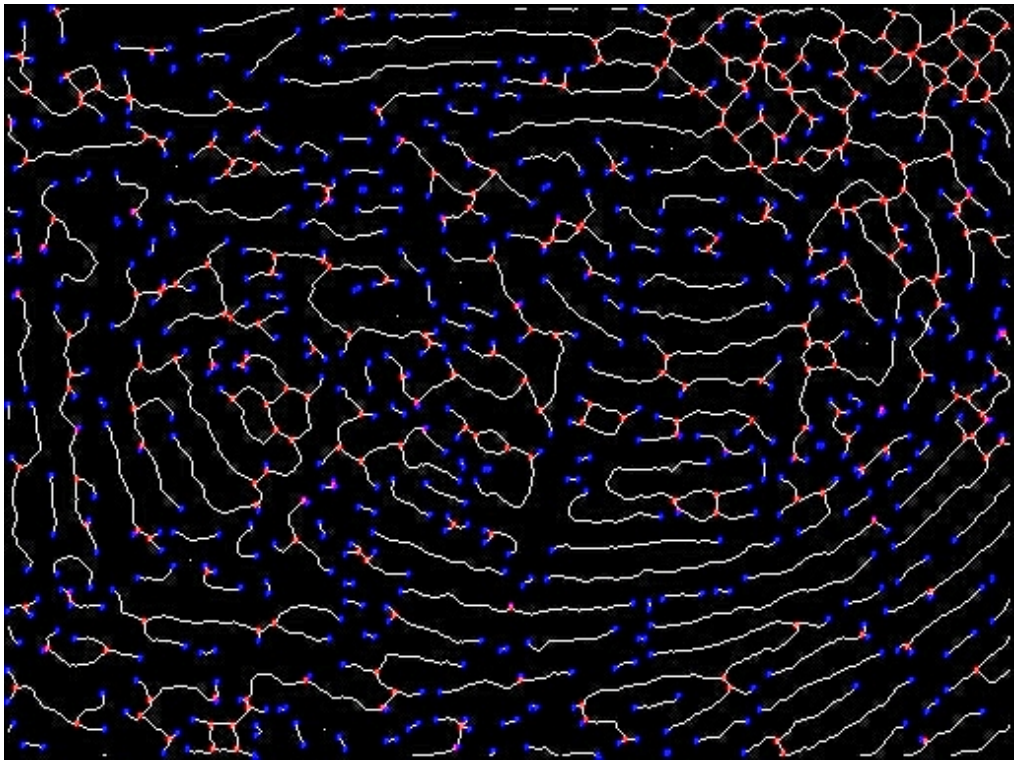


Fig. 44: Imagen 5 esqueletizada con minucias extraídas. (aumentada)

Fig. 45: Imagen 6 segmentada y realzada

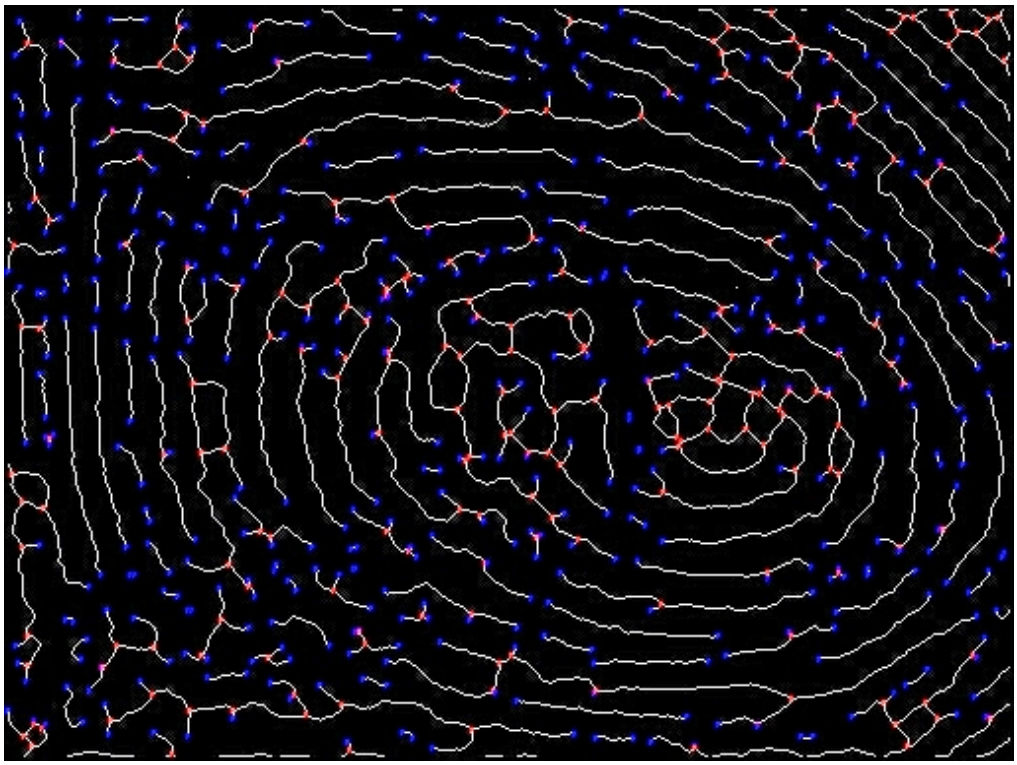


Fig. 46: Imagen 6 esqueletizada con minucias extraídas. (aumentada)

Los tres ejemplos aquí expuestos, corresponden a huellas reales que forman parte de la base de datos de huellas capturadas para la realización de este proyecto. Estas huellas han sido capturadas con los

algoritmos aquí expuestos corriendo bajo la aplicación desarrollada en el dispositivo Android. Esta base de huellas se compone de 10 huellas capturadas del mismo dedo (índice) de 14 personas, a intervalos de 500 ms.

Estos tres ejemplos muestran la mejor huella (Fig. 34) y (Fig. 36), una huella intermedia (Fig. 38) y (Fig. 40) y la peor huella (Fig. 43) y (Fig. 45) de todas las huellas capturadas.

Llegados a este punto, se puede decir que se han conseguido implementar en el dispositivo Android los algoritmos utilizados en MatLab.

No obstante, el tiempo de procesado para ejecutar los algoritmos de esqueletización + extracción de minucias consumen unos 10 s. A partir de aquí habría que implementar la comparación o *Matching.*

El *Matching* de dos huellas dactilares mediante la comparación de minucias es el método mas documentado y el más usado hasta la fecha, sin embargo, incluso en [Chris et al. 2012] [5], se dice que las minucias han sido extraídas mediante el extractor *FingerJetFX* de la compañía *DigitalPersona* [DP-2012]. Este extractor es un SDK (Software Development Kit) para Android, que comercializa esta compañía y que está pensado para poder ser integrado en la aplicación Android, capturando las huellas con un lector de la propia compañía. En el artículo se menciona el hecho, de que la extracción de huellas es de muy baja calidad, ya que este extractor esta pensado para trabajar con imágenes obtenidas de sensores capacitivos u ópticos, no con imágenes capturadas con la cámara del dispositivo.

Otro ejemplo lo tenemos en [Chris et al. 2013] [6], en el que se dice que se realiza en el dispositivo el procesado de la captura y realce de la imagen y posteriormente se envían al PC para procesarlas con el software *MorphoLite* SDK.

No se ha encontrado ningún trabajo que incluya todo el proceso desde la captura hasta el *Matching* final, que se haga 100% en el dispositivo.

En este proyecto hay una serie de premisas que se adoptaron desde el principio y que básicamente son:

Todo lo que se consiga (y lo que no), respecto al tratamiento de la imagen de la huella dactilar, se hará en el dispositivo.

Todo el software que corra en el dispositivo será desarrollado o implementado *ad hoc*. No se incluirán módulos comerciales para realizar ninguna de las fases del tratamiento de la imagen de la huella (incluida la autenticación).

La aplicación tiene que poder ser usada, incluso con sus limitaciones en cuanto a fiabilidad, dentro de unos tiempos "razonables" que un usuario esperaría del manejo de una aplicación, es decir, no nos sirve poder autenticar una huella dactilar con una fiabilidad determinada si en el proceso se tarda un tiempo inaceptable.

Después de estas reflexiones, teniendo en cuenta que además de los 10 s de la esqueletización + extracción de minucias, habría que sumar el algoritmo de *Matching* propiamente dicho y que, probablemente habría que realizar la etapa de filtrado de Gabor tal como estaba implementada en MatLab, es decir, calcular la orientación y frecuencia localmente por bloques, con lo que presumiblemente mejoraríamos el resultado en comparación con el obtenido, pero con un coste computacional (no evaluado), pero que haría que la aplicación fuera realmente difícil de manejar independientemente del resultado.

Todas estas consideraciones nos han llevado a explorar otras vías de realizar esta fase de autenticación partiendo del punto en que nos encontramos. Desde la captura del frame hasta el guardado de la imagen realzada, es decir a falta del *Matching*, la aplicación consume 500 ms, por lo tanto estamos todavía en tiempos casi "transparentes" para el usuario.

Como innovación, vamos a optar en este proyecto por abordar el problema del *Matching* mediante la extracción de características [7] [8], para lo cual utilizaremos la biblioteca OpenCV que incluye funciones para realizar esta extracción de características. La idea es obtener los puntos característicos (*keypoints*) de la imagen de la huella dactilar ya segmentada (Fig. 52) y posteriormente hacer el *Matching* de los *keypoints* de las dos imágenes a comparar (Fig. 53). No hemos encontrado ninguna referencia de *Matching* de huellas dactilares mediante el proceso de extracción de características, por lo que nos parece interesante utilizar este método, que en otros ámbitos del reconocimiento de imágenes o seguimiento (*Tracking*) consigue buenos resultados en aplicaciones en tiempo real, incluso en dispositivos móviles.

Básicamente, las regiones características de una imagen, son aquellas en los que existe una máxima variación cuando nos movemos muy poco alrededor de ellas en cualquier dirección. Encontrar esas regiones es lo que se denomina *Feature Detection*.

Una vez encontradas esas características, se describe la región alrededor de la característica, lo que ayudará a encontrarla en otra imagen. A esta descripción se denomina *Feature Description.* Posteriormente se trata de encontrar esas características en otra imagen, lo que se denomina *Feature Matching*.

Existen diferentes algoritmos en OpenCv:

- Harris corner detection: Es un detector de esquinas y bordes de 1988. La idea es encontrar las diferencias en intensidad por desplazamiento en todas las direcciones.

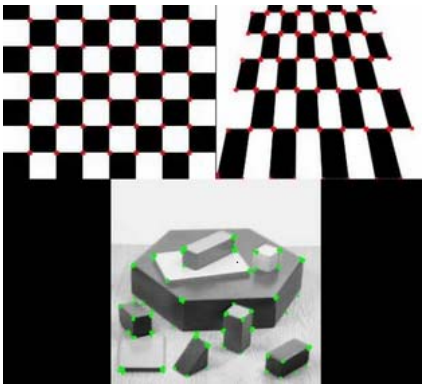


Fig. 47: Imagen obtenida del algoritmo de Harris. Fuente: http://docs.opencv.org

- Shi-Tomasi: Es una mejora del algoritmo de Harris de 1994.

-SIFT (Scale-Invariant Feature Transform): Los dos algoritmos anteriores son invariantes ante la rotación, ya que una esquina sigue siendo una esquina aunque la rotemos, pero no eran invariantes ante la escala, es decir una esquina parece una esquina si la miramos de lejos, pero si nos acercamos demasiado puede no parecer una esquina si estamos muy cerca (Fig. 48).

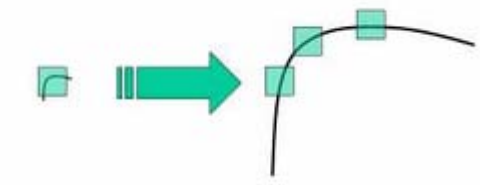


Fig. 48: Diferencia de una esquina con la escala: Fuente: http://docs.opencv.org

Este algoritmo realiza el *Matching* entre dos imágenes comparando los *Keypoints* e identificando los píxeles vecinos más próximos (Fig. 49).

Fig. 49: Keypoint obtenidos por el algoritmo SIFT. Fuente: http://docs.opencv.org

- SURF (Speeded-Up Robust Features): El algoritmo SIFT, era muy lento y en el 2006 apareció SURF, el cual es una versión acelerada de SIFT. SURF es hasta 3 veces más rápido que SIFT manteniendo la misma calidad. SURF es bueno manejando imágenes borrosas y rotadas, pero no es bueno con cambios de puntos de vista e iluminación.

Fig. 50: Resultado de SURF, se aprecia como en la imagen detecta las manchas blancas. Fuente: http://docs.opencv.org

- FAST (Features from Accelerated Segment Test): Los anteriores algoritmos no son suficientemente rápidos para aplicaciones en tiempo

real. Como solución a este problema apareció en 2006 el algoritmo FAST (Fig. 51).

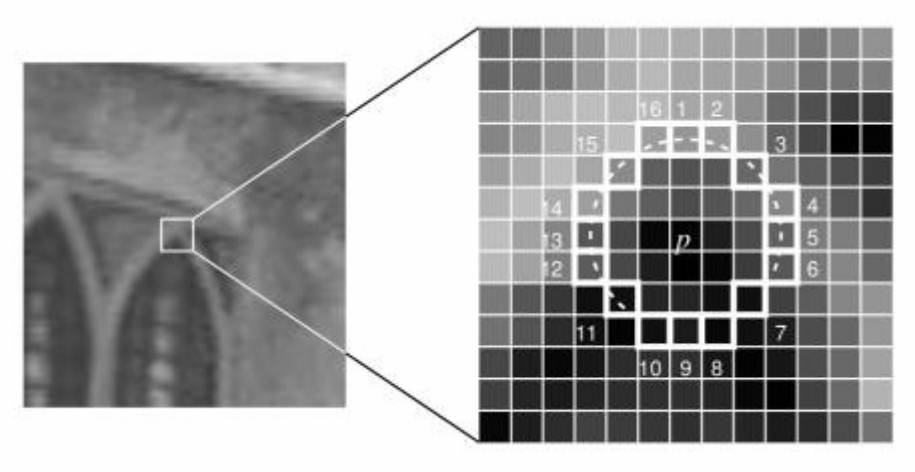


Fig. 51: Detalle funcionamiento de FAST. Fuente: http://docs.opencv.org

Básicamente, para cada pixel evaluado (P), se tiene el valor de intensidad del pixel (Ip) y se fija un valor umbral apropiado (t). Se considera un circulo de 16 píxeles alrededor del pixel bajo evaluación y se determina que el pixel P es una esquina si existen un número (n) de píxeles contiguos dentro del circulo que sean todos más brillantes que Ip+t o mas oscuros que Ip-t. FAST es varias veces más rápido que otros detectores de esquinas pero no es robusto para altos niveles de ruido.

- BRIEF: SIFT y SURF trabajan con vectores de 128 y 64 dimensiones respectivamente para los descriptores y números en coma flotante, con lo cual necesitan unos vectores para crear los descriptores de 512 y 256 bytes respectivamente. Estas dimensiones pueden no ser necesarias para realizar el matching. Estos descriptores se pueden comprimir en cadenas binarias para ser usadas en el matching computando la distancia de Hamming, que se puede implementar muy rápidamente mediante XOR. BRIEF provee una forma rápida para encontrar esas cadenas binarias. BRIEF es un descriptor de características (Feature Descriptor), no proporciona ningún método para encontrar las características, para ello hay que usar otro detector como SIFT o SURF.

BRIEF es un método rápido para calcular los descriptores y realizar el matching.

- ORB (Oriented FAST and Rotated BRIEF): Este algoritmo apareció en 2011 como alternativa a SIFT y SURF. Es más rápido, tiene mejores prestaciones y además se puede usar gratis, ya que SIFT y SURF están patentados y habría que pagar en caso de ser utilizados. ORB es una fusión del detector de keypoints de FAST y del generador de descriptores

de BRIEF, pero mejorado. En definitiva, ORB es mucho más rápido que SURF y SIFT, lo cual hace de el una buena elección en dispositivos móviles que trabajen en tiempo real como sucede en nuestro caso.

Después de este pequeño repaso a los distintos algoritmos de extracción de características y matching, decidimos implementar el algoritmo ORB mediante la biblioteca OpenCV (Fig. 53) en nuestro proceso de emparejamiento o matching de imágenes de huellas dactilares (Fig. 52).

Fig. 52: Imagen de la huella segmentada y realzada

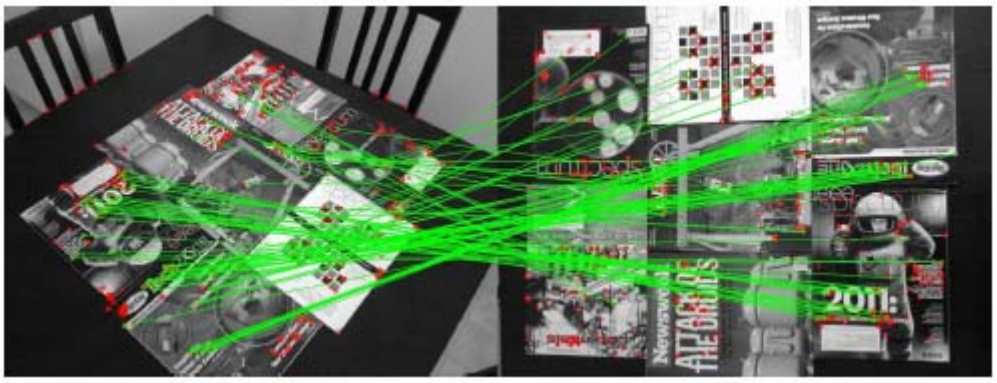


Fig. 53: Matching entre dos imágenes usando un detector de keypoints tipo ORB

El algoritmo consta de varias fases:

1. Encontrar las características de las dos huellas a comparar.

```
public static int startProcessing(Mat img1, Mat img2) {
    //Definimos el tipo detector, el descriptor y el matcher
    FeatureDetector detector = FeatureDetector.create(FeatureDetector.PYRAMID_ORB);
    DescriptorExtractor descriptor = DescriptorExtractor.create(DescriptorExtractor.ORB);
    DescriptorMatcher matcher = DescriptorMatcher.create(DescriptorMatcher.BRUTEFORCE_HAMMING);

    //first image

    Mat H = new Mat();
    Mat descriptors1 = new Mat();
    MatOfKeyPoint keypoints1 = new MatOfKeyPoint();

    // Detecta los puntos característicos de la primera imagen (keypoints)
    detector.detect(img1, keypoints1);
    // Extrae los descriptores de los keypoints de la primera imagen
    descriptor.compute(img1, keypoints1, descriptors1);

    //second image

    Mat descriptors2 = new Mat();
    MatOfKeyPoint keypoints2 = new MatOfKeyPoint();
    // Detecta los puntos característicos de la segunda imagen (keypoints)
    detector.detect(img2, keypoints2);
    // Extrae los descriptores de los keypoints de la segunda imagen
    descriptor.compute(img2, keypoints2, descriptors2);
```

2. Realizar el emparejamiento de las dos huellas. Esto es encontrar los *Matches* (líneas verdes (Fig. 53)) o emparejamiento de los *keypoints* encontrados entra una imagen y otra. En esta fase se desechan los *Matches* que tienen una distancia mayor que un cierto umbral, es decir, si colocáramos una imagen encima de la otra, las distancias que habría entre los puntos que están emparejados (*Matches)*.

```
MatOfDMatch  matches = new MatOfDMatch();
//Encuentra los mejores matches entre los puntos de una imagen y la otra
matcher.match(descriptors1,descriptors2,matches);

//output image
Mat outputImg = new Mat();
MatOfByte drawnMatches = new MatOfByte();
//this will draw all matches, works fine

// Nos quedamos solo con aquellos matches suya distancia sea < 45
int DIST_LIMIT = 45;
List<DMatch> matchList = matches.toList();
LinkedList<DMatch> matches_final_distance = new LinkedList<DMatch>();
for(int i=0; i<matchList.size(); i++){
    if(matchList.get(i).distance <= DIST_LIMIT){
        matches_final_distance.add(matches.toList().get(i));

    }
}
```

3. Calcular el ángulo sobre lo horizontal de cada uno de los *Matches* y agruparlos en función de ese ángulo.

```
//Computamos el angulo de cada uno delos matches y los agrupamos
for(int i=0; i<matches_final_distance.size(); i++)
{
   double angle = Math.round((Math.atan2(Math.abs((queryList.get(i).y)-(trainList.get(i).y)),
                  Math.abs((queryList.get(i).x)-(trainList.get(i).x))))*180/Math.PI);

   if(i == 0)
   {
      anglevalue[i] = angle;

   }

   for(int v=0; v<=i; v++)
   {

      if(anglevalue[v] == angle)
       {
          anglecount[v]++;

       }else if(anglevalue[v]  == 1000.0)
       {

                anglevalue[v] = angle;


       }
   }
  matches_final_mat.add(matches_final_distance.get(i));


}
```

4. Quedarnos con el mayor grupo de *Matches* con el mismo ángulo, es decir, con el mayor número de *Matches* que sean paralelos entre si. Con este paso se eliminan los outliers, es decir los falsos *Matches*.

```
//Nosquedamos conel grupo de angulos más numeroso
for(int i=0; i<matches_final_distance.size(); i++)
 {
   for(int r=0; r<matches_final_distance.size(); r++)
     {
       if((anglecount[i] > anglecount[r]) && (anglecount[i] > anglecountmax))
       {
          anglecountmax = anglecount[i];
          anglevaluemax = anglevalue[i];

       }
     }
 }
```

5. El resultado que arroja el algoritmo es el número de *Matches* que han cumplido todas las condiciones anteriores, lo que denominaremos "score". Posteriormente, si este valor es mayor que un cierto umbral fijado, se asume que las dos imágenes corresponden a la misma huella y el *Matching* se considera correcto, en caso contrario se considera falso.

```
    // La función retorna el valor del número de angulos paralelos encontrados
     return anglecountmax;

}
```

El resultado se puede ver en las siguientes imágenes de huellas de nuestra base de datos a las que se les ha aplicado el algoritmo completo (Segmentación+Realce+Extracción de Características (ORB)+Matching), siempre corriendo los algoritmos desde nuestro dispositivo móvil.

Fig. 54: Imagen original de "Javi"

Fig. 55: Imagen original de "Rosa"

Fig. 56: Comparación de la misma imagen de la huella Javi1. (SCORE = 995)

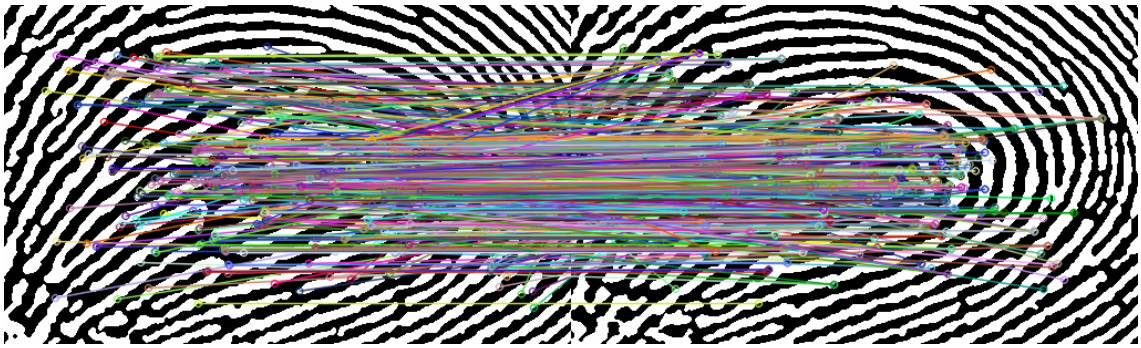


Fig. 57: Comparación de dos imágenes distintas de la misma huella (Javi1-Javi9).

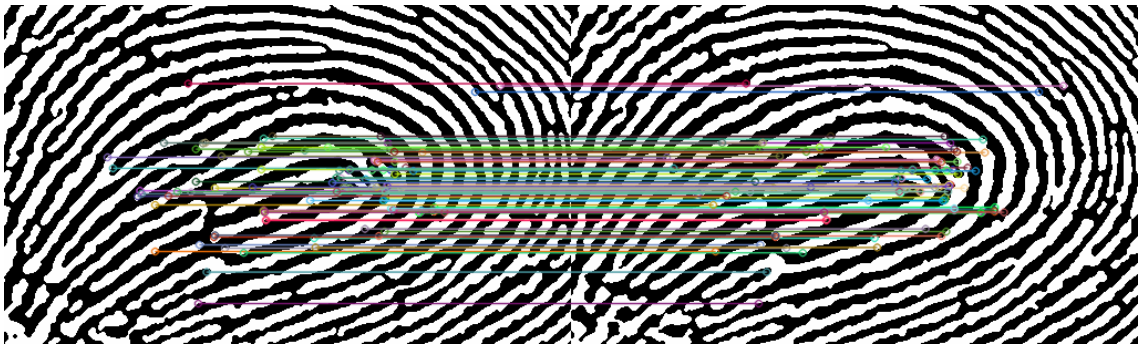


Fig. 58: Comparación de dos imágenes distintas de la misma huella (Javi1-Javi9) con los outlayers descartados. (SCORE = 97)

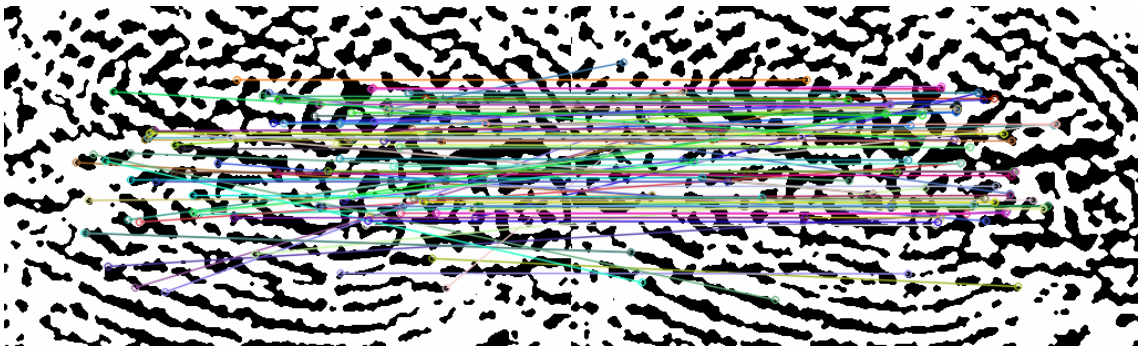


Fig. 59: Comparación de dos imágenes distintas de la misma huella (Rosa1-Rosa2).

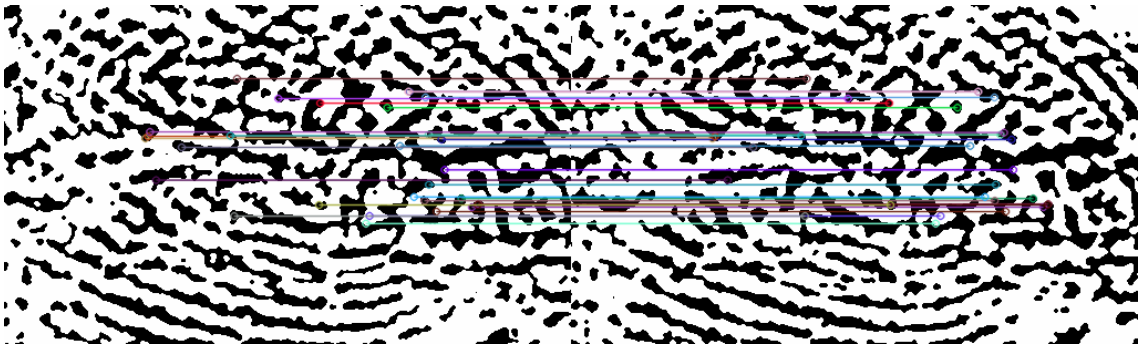


Fig. 60: Comparación de dos imágenes distintas de la misma huella (Rosa1-Rosa2) con los outlayers descartados. (SCORE = 28)

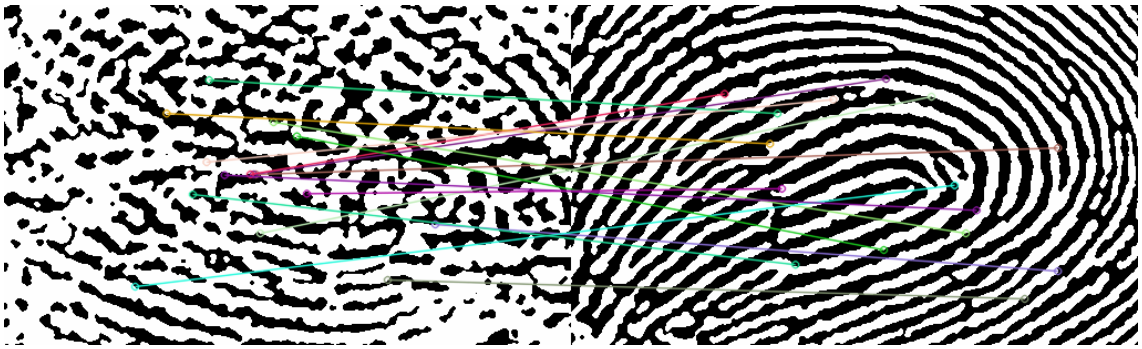


Fig. 61: Comparación de dos imágenes de distintas huellas (Rosa2- Javi1)

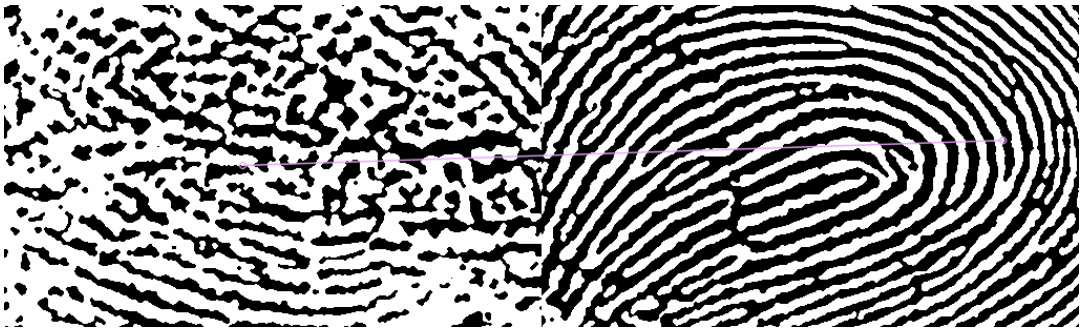


Fig. 62: Comparación de dos imágenes de distintas huellas (Rosa2- Javi1) con los outlayers descartados. (SCORE = 1)

3.4 Conclusiones

En las imágenes se observa que cuando comparamos la misma imagen (Fig. 56), el score es muy elevado (995) y además todos sus matches son paralelos entre si y con ángulo de 0º, como cabría esperar. Esto se ha verificado con el resto de huellas de la base de datos.

Por otro lado vemos, que al comparar dos imágenes distintas de la misma huella del mismo sujeto, (Fig. 57) y (Fig. 58), tenemos en la primera imagen (Fig. 57), el mismo número de Matches que en el ejemplo anterior, es decir el algoritmo ha encontrado un número similar de puntos característicos, pero ya se ve a simple vista que ya no son todos paralelos, es decir, existen muchos outlayers. En la siguiente imagen (Fig. 58), se observa como una vez limpiados los outlayers nos quedan todos los Matches paralelos y con un score de 97.

En la imágenes (Fig. 59) y (Fig. 60), tenemos la misma situación pero con las peores huellas de nuestra base de datos. En ellas vemos que se repite la misma situación pero esta vez el score obtenido es de 28. Este resultado parece lógico debido a que la huella "Rosa" es de muy baja calidad, aunque es de destacar que ha encontrado 28 emparejamientos y que si analizamos las imágenes con detenimiento, estos corresponden realmente a las mismos puntos de las imágenes, es decir, el algoritmo parece que funciona y empareja puntos de imágenes en las que a priori la técnica de las minucias habría fracasado por la pésima esqueletización conseguida.

También es de destacar el ultimo caso mostrado (Fig. 61) y (Fig. 62), en el que se ha querido demostrar que una huella de muy mala calidad (Rosa), que con sus propias imágenes da como resultado scores "bajos", cuando se compara con huellas de buena calidad (Javi) los scores son todavía mas bajos, como en el caso de la imagen (Fig. 62), en la que el score entre ambas es de 1. Esto significa que el algoritmo es capaz de extraer características "reales" de la huella de baja calidad y por ello no las confunde con otras características de una huella de buena calidad.

Como conclusión del capitulo, podemos decir que la implementación del algoritmo de extracción de características parece que puede dar buenos resultados y que su funcionamiento responde a lo que esperábamos.

En el siguiente capitulo, extraeremos resultados de toda la base de datos, cruzando todas las imágenes y obtendremos las tasas EER FAR y FRR del conjunto (proceso asimilado a la identificación). Además haremos alguna consideración sobre el resultado obtenido en el *Matching* Local implementado en la aplicación (proceso asimilado a la autenticación).

4. Resultados

4.1 Método de obtención de datos

Tal como se explicó en el capítulo 1.3.2 Objetivos, vamos a calcular la FAR (*False Aceptante Rate)*, FRR *(False Rejection Rate)* y EER (Equal Error Rate). Para extraer estas tasas se ha realizado el matching de las huellas de toda la base de datos del proyecto. Siguiendo con las premisas antes mencionadas, el procesado de todas las huellas se ha realizado íntegramente en el dispositivo móvil.

Para ello se ha implementado en la aplicación una actividad (Ver anexo B) que es capaz de realizar el matching de cada una de las huellas con todas las demás y generar una hoja Excel con los valores obtenidos (scores) de todas estas comparaciones.

La base de datos del proyecto se compone de 10 huellas del dedo índice capturadas de un total de 14 personas (Fig. 63) y (Fig. 64), lo que hacen un total de 140 huellas y 140x140 = 19600 comparaciones.

La comparación de cada par de huellas tarda unos 500 ms, lo que hace un total de 9800 s ≈ 2 horas y 45 minutos, que es el tiempo que tardó el dispositivo en generar la Excel mencionada con todos los valores.

A partir de aquí, acaba el vínculo con el procesamiento de la imagen, por lo que podemos extraer la Excel (Tabla 1) a un PC, trabajar con ella, y obtener las tasas mencionadas.

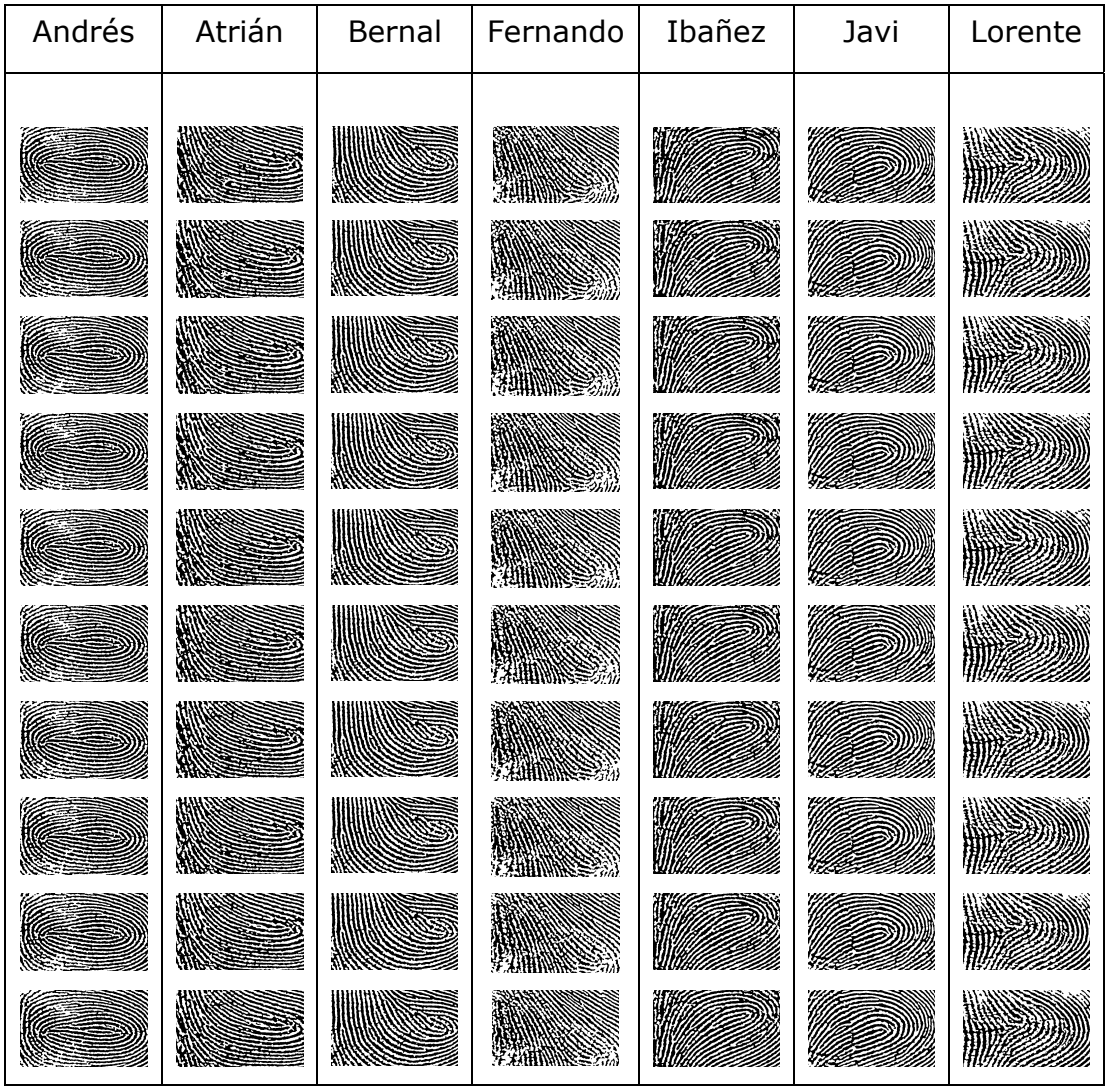


Fig. 63: Huellas de la Base de Datos del proyecto (I)

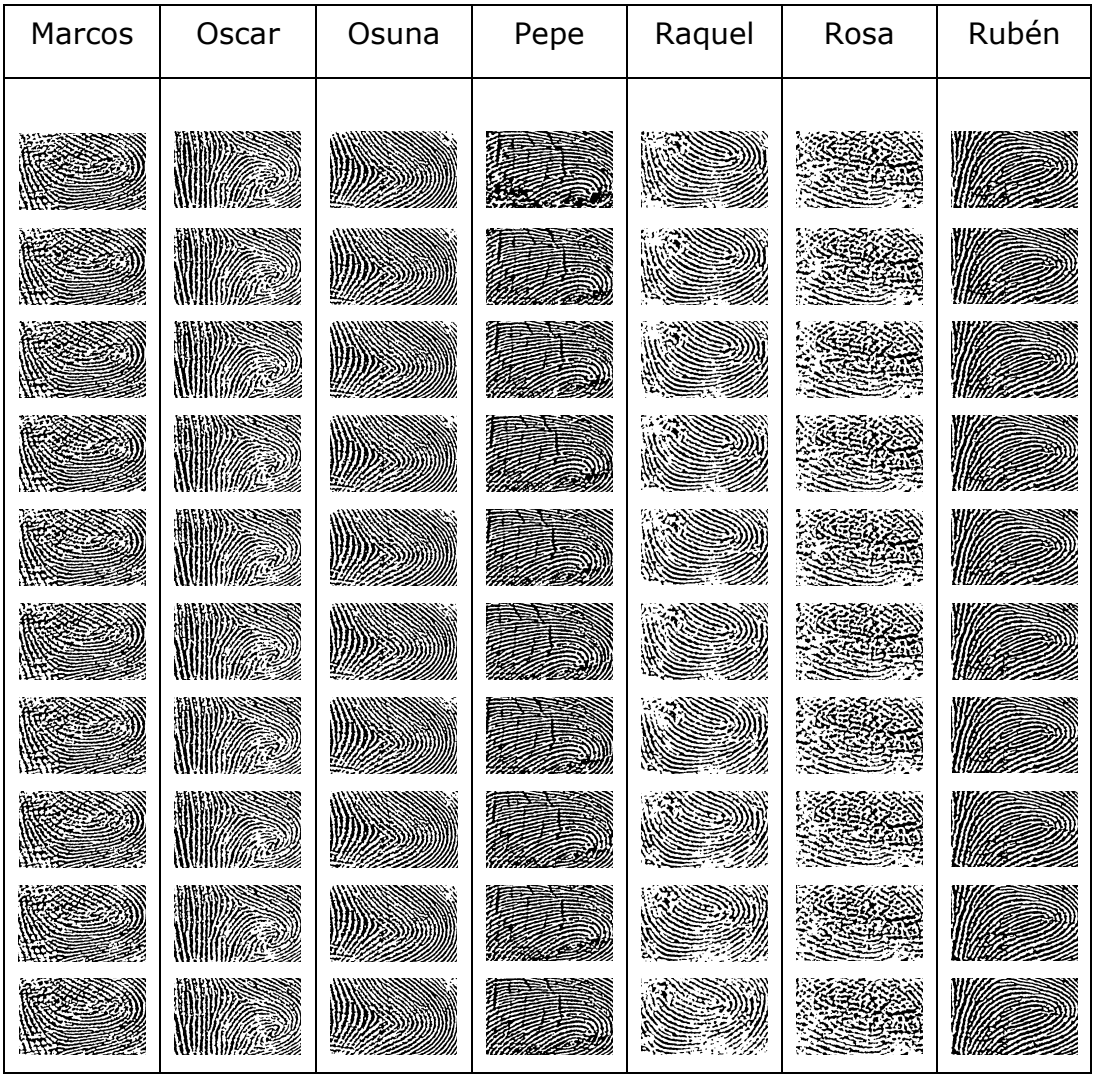


Fig. 64: Huellas de la Base de Datos del proyecto (II)

		andres1.png	andres2.png	andres3.png	andres4.png	andres5.png	andres6.png	andres7.png	andres8.png	andres9.png	andres10.png	atrian1.png	atrian2.png	atrian3.png	atrian4.png
Andrés	andres1.png	985	32	17	30	16	13	11	12	16	19	8	7	5	6
	andres2.png	34	966	33	28	12	16	11	11	13	17	5	7	5	5
	andres3.png	18	32	974	120	41	13	17	15	12	14	7	7	6	6
	andres4.png	33	28	118	956	25	20	19	14	9	18	6	4	5	5
	andres5.png	15	14	40	24	971	54	21	20	16	17	5	5	4	5
	andres6.png	13	16	13	18	54	975	35	26	24	16	6	5	6	6
	andres7.png	10	11	14	19	22	33	976	21	25	14	6	4	5	7
	andres8.png	13	9	17	14	21	26	24	967	41	13	7	7	7	5
	andres9.png	16	13	12	11	17	25	27	39	968	60	6	4	4	5
	andres10.png	18	14	14	19	18	15	15	11	60	950	8	4	6	6
Atrián	atrian1.png	6	5	8	5	6	9	5	6	6	8	984	43	20	31
	atrian2.png	9	7	7	5	6	6	6	6	3	6	40	981	105	82
	atrian3.png	5	5	6	5	5	6	4	5	5	6	22	105	978	71
	atrian4.png	7	5	6	6	5	6	6	6	6	6	31	83	74	992
	atrian5.png	5	5	4	5	5	4	5	6	4	4	25	57	83	75
	atrian6.png	5	6	6	8	4	5	4	6	4	6	19	15	15	30
	atrian7.png	5	7	4	5	5	6	4	5	5	5	13	20	24	25
	atrian8.png	6	7	6	4	7	6	6	4	6	6	20	14	19	21
	atrian9.png	5	5	8	8	7	4	5	4	5	9	14	14	7	18
	atrian10.png	4	6	4	5	6	5	6	6	4	6	18	29	29	27
Bernal	bernal1.png	11	8	7	9	6	7	9	9	7	7	7	8	9	10
	bernal2.png	8	8	8	7	8	7	6	7	6	7	9	7	12	7
	bernal3.png	8	7	6	7	6	7	7	5	6	8	8	8	7	9
	bernal4.png	7	6	9	8	9	8	6	9	8	9	9	6	7	9
	bernal5.png	9	7	8	9	5	7	6	6	9	8	11	8	9	8
	bernal6.png	6	10	8	6	11	9	6	7	8	6	10	7	9	11
	bernal7.png	8	9	7	7	5	7	8	7	6	8	7	8	8	8
	bernal8.png	6	6	6	11	7	7	6	9	11	8	9	7	9	8
	bernal9.png	7	6	6	8	7	8	9	8	6	9	9	7	8	7
	bernal10.png	6	6	6	8	6	7	8	6	7	7	7	7	9	11
Fernando	fernando1.png	8	5	6	7	6	7	5	5	6	8	5	5	5	5
	fernando2.png	4	6	5	6	5	5	6	6	7	6	7	5	5	5
	fernando3.png	5	5	5	7	5	4	4	6	5	6	6	4	8	5
	fernando4.png	5	5	6	6	5	5	9	4	7	4	6	4	4	5
	fernando5.png	5	5	5	6	4	6	4	5	5	5	4	6	5	5
	fernando6.png	4	5	5	6	4	5	5	5	5	3	5	3	3	4
	fernando7.png	4	6	6	5	5	6	5	5	5	5	5	5	6	5
	fernando8.png	4	5	5	5	5	3	4	5	5	4	4	3	4	4
	fernando9.png	6	5	5	4	4	4	4	4	5	4	4	4	3	5
	fernando10.png	4	4	4	5	4	3	4	6	4	3	3	4	4	3
Ibáñez	ibañez1.png	7	10	8	8	7	7	9	8	9	9	8	8	8	10
	ibañez2.png	10	9	10	8	10	7	8	14	10	7	9	5	10	8
	ibañez3.png	9	8	7	9	8	10	9	11	7	6	12	8	8	8
	ibañez4.png	9	10	9	10	11	12	9	8	12	8	8	8	9	9
	ibañez5.png	9	10	10	10	9	10	7	10	9	6	11	8	9	8
	ibañez6.png	10	9	10	10	12	10	8	9	10	8	9	8	9	8
	ibañez7.png	8	8	11	8	8	6	8	10	8	7	9	6	8	7
	ibañez8.png	8	8	11	8	12	11	8	11	11	11	10	7	8	9
	ibañez9.png	10	12	8	8	10	10	8	12	10	6	9	9	8	8
	ibañez10.png	11	10	10	10	11	10	8	10	8	9	6	7	9	7

Tabla 1: Fragmento de la Excel obtenida con 10 huellas por persona

4.2 Generación de resultados

Una vez obtenida la Excel, se hizo una primera evaluación en la que se obtuvo el porcentaje de aciertos, es decir, para cada una de las huellas comparada con todas las demás, que porcentaje de veces se elegiría una huella de la misma persona.

Este proceso se llevó a cabo primero con toda la base de huellas, en la que hay, como ya dijimos, 10 huellas de cada persona.

	Andrés	Atrián	Bernal	Fernando	Ibañez	Javi	Lorente	Marcos	Oscar	Osuna	Pepe	Raquel	Rosa	Rubén
Andrés	10	0	0	0	0	0	0	0	0	0	0	0	0	0
Atrián	0	10	0	0	0	0	0	0	0	0	0	0	0	0
Bernal	0	0	10	0	0	0	0	0	0	0	0	1	0	0
Fernando	0	0	0	10	0	0	0	0	0	0	0	0	0	0
Ibañez	0	0	0	0	10	0	0	0	0	0	0	0	0	0
Javi	0	0	0	0	0	10	0	0	0	0	0	0	0	0
Lorente	0	0	0	0	0	0	10	0	0	0	0	0	0	0
Marcos	0	0	0	0	0	0	0	10	0	0	0	0	0	0
Oscar	0	0	0	0	0	0	0	0	10	0	0	0	0	0
Osuna	0	0	0	0	0	0	0	0	0	10	0	0	0	0
Pepe	0	0	0	0	0	0	0	0	0	0	10	0	0	0
Raquel	0	0	0	0	0	0	0	0	0	0	0	9	0	0
Rosa	0	0	0	0	0	0	0	0	0	0	0	0	10	0
Rubén	0	0	0	0	0	0	0	0	0	0	0	0	0	10
	10	10	10	10	10	10	10	10	10	10	10	10	10	10
%	100	100	100	100	100	100	100	100	100	100	100	90	100	100
Huellas tomadas a "altura constante", con posicionado asistido por la app en el plano X-Y por realimentación visual del usuario, a interalos de 0.5s														
Tiempo de detección ~70s =>10 huellas x 14 personas = 140 huellas x 0.5s cada comparación														

Tabla 2: Comparación con 10 huellas por persona

En la Tabla 2, se observa como solamente una de las huellas de Raquel fue confundida con una de Bernal.

Esta primera comparación se hizo sin fijar un umbral, es decir, eligiendo la huella que arrojaba un mayor "score" al ser comparada con la huella "a identificar".

Este estudio se repitió reduciendo el número de huellas de cada persona, para ver si empeoraba el porcentaje. Para ello se programaron unas macros en la hoja Excel con las que se generaron unas subtablas de distinto número de huellas por persona. En concreto se generaron tablas con 7, 5, 3 ,2 y 1 huella de cada persona respectivamente.

Cada subtabla de cada tipo se puede generar tantas veces como se quiera y se genera aleatoriamente, es decir, la macro programada es

capaz de formar una subtabla seleccionando aleatoriamente un subconjunto de huellas del conjunto principal de las 10 de cada persona para generar una nueva tabla y calcular además los nuevos porcentajes.

Este proceso de generación aleatoria de subtablas se repitió numerosas veces, verificando que los porcentajes obtenidos con la tabla inicial de 10 huellas por persona se mantenían aunque se disminuyera el número de huellas. Con esto concluimos que no afecta significativamente el número de huellas.

4.3 Interpretación de los resultados

Partiendo de las subtablas generadas, se obtuvieron las tasas FAR, FRR y EER para cada una de ellas.

En la Fig. 65, se ve la grafica obtenida para el conjunto completo de huellas (140 huellas). En ella se puede ver que la FAR o tasa de falsa aceptación, es del 100% para valores de umbral (scores) de 0. Este resultado es coherente ya que, si ponemos la condición de que el número de coincidencias entre huellas sea de 0, todas serán aceptadas. Según vamos aumentando el umbral, la FAR cae hasta que alrededor del umbral 15, llega a valer 0. Esto quiere decir que si ponemos la condición de que las huellas tengan 15 coincidencias o más entre ellas, no habrá ningún falso emparejamiento.

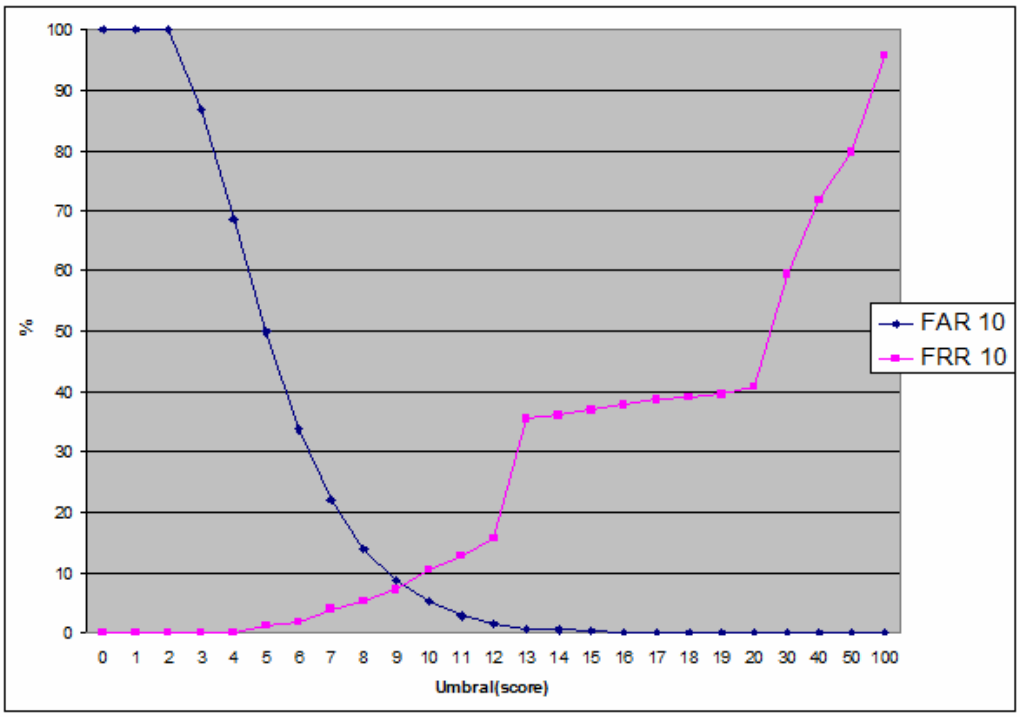


Fig. 65: FAR y FRR con 10 huellas por persona

Por otro lado, se observa que la tasa FRR es del 100% para valores de umbral mayores de 100, lo que significa que si ponemos la condición de que las huellas tengan más de 100 coincidencias, un 100% serán rechazadas siendo autenticas. Esta tasa FRR va decayendo según va disminuyendo el valor del umbral hasta que se hace 0 alrededor del umbral 5 o menor, que significa que con umbrales menores que 5, todas las huellas autenticas serán reconocidas como tales.

La tasa EER o tasa de error igual, o lo que es lo mismo, el valor de umbral en el que FAR = FRR, se obtiene entorno al valor de umbral 9. En ese punto el valor de FRA = FRR ≈ 9 %.

En la Fig. 66, se puede ver la gráfica generada con la subtabla de 7 huellas (Fig. 67). En la gráfica se han representado los valores FAR y FRR para cuatro subtablas generadas aleatoriamente del conjunto total.

Se puede apreciar que las curvas obtenidas no difieren mucho entre sí y tampoco de la grafica obtenida para las 10 huellas. El EER se sitúa de nuevo entorno al umbral 9 y la EER ≈ 9%

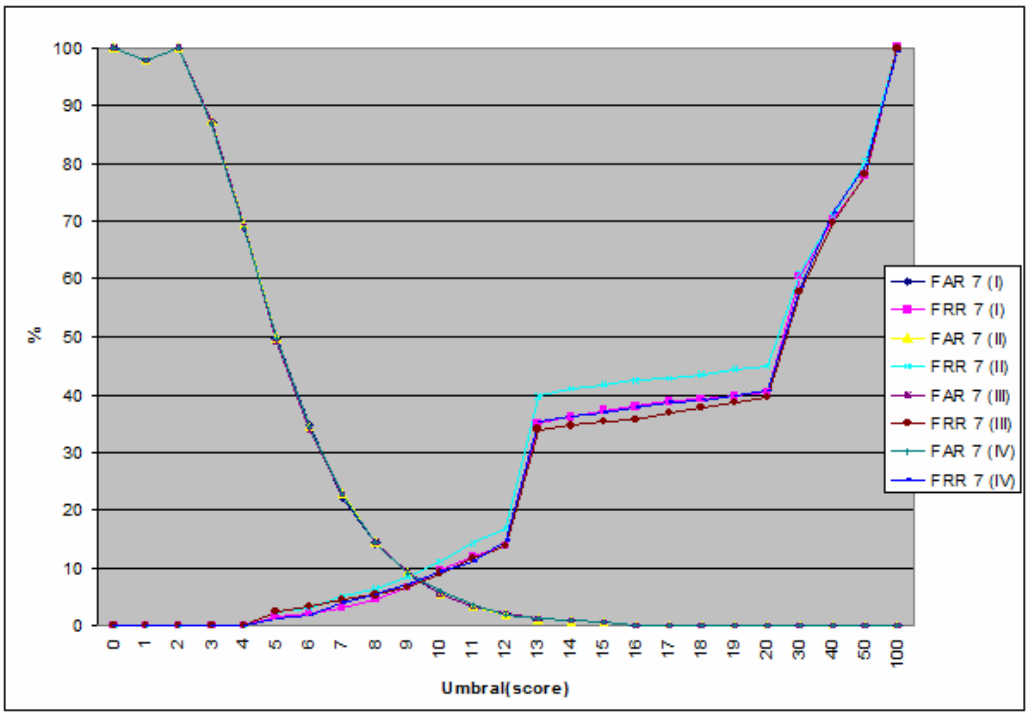


Fig. 66: FAR y FRR con 7 huellas por persona

		andres7.png	andres9.png	andres5.png	andres10.png	andres4.png	andres1.png	andres8.png	atrian9.png	atrian3.png	atrian5.png	atrian1.png	atrian7.png	atrian2.png	atrian6.png
Andrés	andres7.png	800	25	22	14	19	10	21	5	5	5	6	5	4	7
	andres9.png	27	800	17	60	11	16	39	6	4	5	6	5	4	5
	andres5.png	21	16	800	17	24	15	20	8	4	5	5	5	5	5
	andres10.png	15	60	18	800	19	18	11	8	6	4	8	6	4	7
	andres4.png	19	9	25	18	800	33	14	7	5	5	6	5	4	7
	andres1.png	11	16	16	19	30	800	12	5	5	4	8	6	7	6
	andres8.png	24	41	21	13	14	13	800	5	7	6	7	6	7	6
Atrián	atrian9.png	5	5	7	9	8	5	4	800	7	22	14	44	14	23
	atrian3.png	4	5	5	6	5	5	5	11	800	80	22	25	105	17
	atrian5.png	5	4	5	4	5	5	6	19	83	800	25	30	57	36
	atrian1.png	5	6	6	8	5	6	6	13	20	27	800	12	43	20
	atrian7.png	4	5	5	5	5	5	5	41	24	31	13	800	20	36
	atrian2.png	6	3	6	6	5	9	6	15	105	55	40	21	800	16
	atrian6.png	4	4	4	6	8	5	6	22	15	37	19	34	15	800
Bernal	bernal3.png	7	6	6	8	7	8	5	9	7	8	8	8	8	12
	bernal8.png	6	11	7	8	11	6	9	8	9	7	9	8	7	10
	bernal10.png	8	7	6	7	8	6	6	7	9	8	7	8	7	10
	bernal5.png	6	9	5	8	9	9	6	8	9	9	11	8	8	6
	bernal4.png	6	8	9	9	8	7	9	6	7	8	9	7	6	11
	bernal7.png	8	6	5	8	7	8	7	7	8	7	7	7	8	8
	bernal6.png	6	8	11	6	6	6	7	8	9	8	10	10	7	7

Fig. 67: Fragmento de la Excel con 7 huellas por persona

En la Fig. 68, se puede ver la gráfica generada con la subtabla de 5 huellas (Fig. 69). En la gráfica se han representado los valores FAR y FRR para cuatro subtablas generadas aleatoriamente del conjunto total.

Se puede apreciar de nuevo, que las curvas obtenidas no difieren mucho entre sí y tampoco de la grafica obtenida para las 10 huellas. El EER se sitúa de nuevo entorno al umbral 9 y la EER ≈ 9%, aunque se aprecia que en la curva FRR 5 (II) (curva amarilla), el EER se ha movido hacia el 10%.

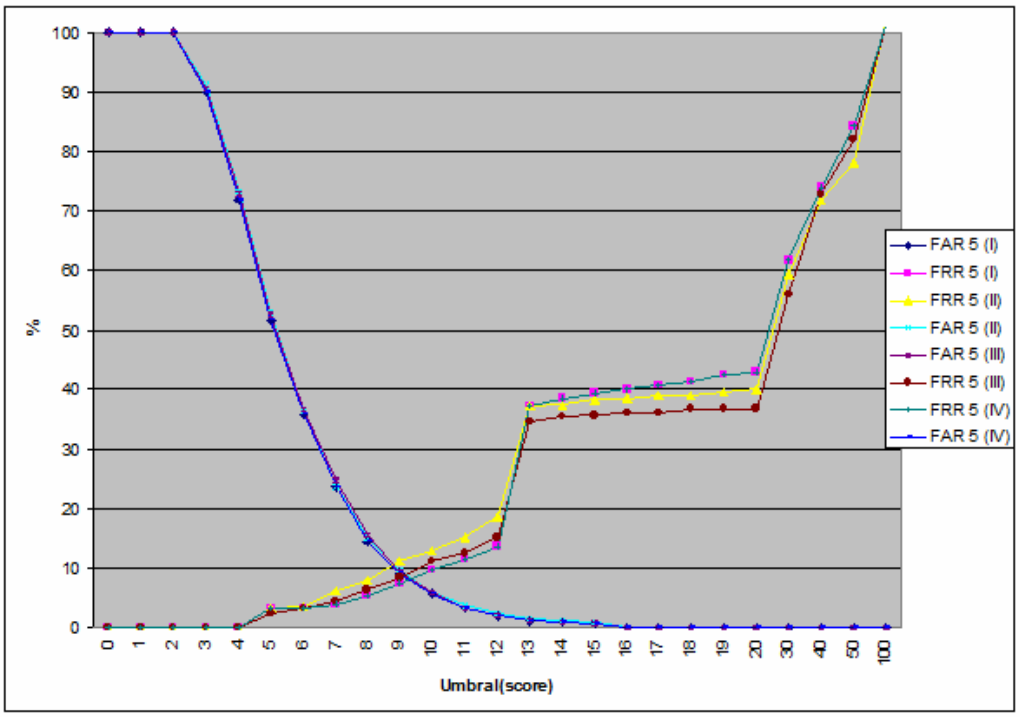


Fig. 68: FAR y FRR con 5 huellas por persona

		andres10.png	andres1.png	andres6.png	andres5.png	andres3.png	atrian8.png	atrian3.png	atrian7.png	atrian2.png	atrian10.png
Andrés	andres10.png	800	18	15	18	14	5	6	6	4	5
	andres1.png	19	800	13	16	17	5	5	6	7	6
	andres6.png	16	13	800	54	13	5	6	6	5	5
	andres5.png	17	15	54	800	40	6	4	5	5	7
	andres3.png	14	18	13	41	800	6	6	5	7	4
Atrián	atrian8.png	6	6	6	7	6	800	19	53	14	13
	atrian3.png	6	5	6	5	6	20	800	25	105	31
	atrian7.png	5	5	6	5	4	56	24	800	20	45
	atrian2.png	6	9	6	6	7	14	105	21	800	31
	atrian10.png	6	4	5	6	4	14	29	45	29	800
Bernal	bernal2.png	7	8	7	8	8	10	12	7	7	9
	bernal9.png	9	7	8	7	6	13	8	9	7	8
	bernal4.png	9	7	8	9	9	14	7	7	6	7
	bernal10.png	7	6	7	6	6	9	9	8	7	7
	bernal1.png	7	11	7	6	7	8	9	6	8	7

Fig. 69: Fragmento de la Excel con 5 huellas por persona

En la Fig. 70, se puede ver la gráfica generada con la subtabla de 3 huellas (Fig. 71). En la gráfica se han representado los valores FAR y FRR para cuatro subtablas generadas aleatoriamente del conjunto total.

Se puede apreciar de nuevo, que las curvas obtenidas no difieren mucho entre sí y tampoco de la grafica obtenida para las 10 huellas. El EER se sitúa entre el umbral 9 y 10. La EER ≈ 9%.

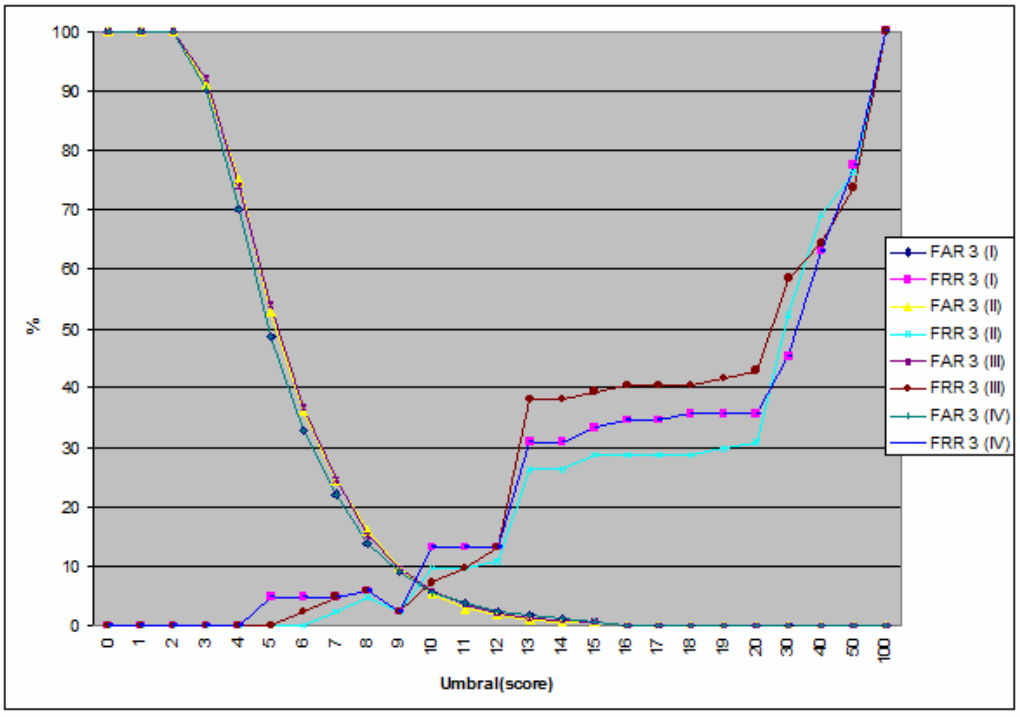


Fig. 70: FAR y FRR con 3 huellas por persona

		andres8.png	andres4.png	andres3.png	atrian4.png	atrian10.png	atrian2.png
Andrés	andres8.png	800	14	17	5	6	7
	andres4.png	14	800	118	5	7	4
	andres3.png	15	120	800	6	4	7
Atrián	atrian4.png	6	6	6	800	24	83
	atrian10.png	6	5	4	27	800	29
	atrian2.png	6	5	7	82	31	800
Bernal	bernal5.png	6	9	8	8	5	8
	bernal4.png	9	8	9	9	7	6
	bernal6.png	7	6	8	11	7	7
Fernando	fernando10.png	6	5	4	3	3	4
	fernando9.png	4	4	5	5	5	4
	fernando8.png	5	5	5	4	4	3

Fig. 71: Fragmento de la Excel con 3 huellas por persona

En la Fig. 72, se puede ver la gráfica generada con la subtabla de 2 huellas (Fig. 73). En la gráfica se han representado los valores FAR y FRR para cuatro subtablas generadas aleatoriamente del conjunto total.

De nuevo, las curvas obtenidas no difieren mucho entre sí. El EER se sitúa entre el umbral 9 y 10. La EER ≈ 8%.

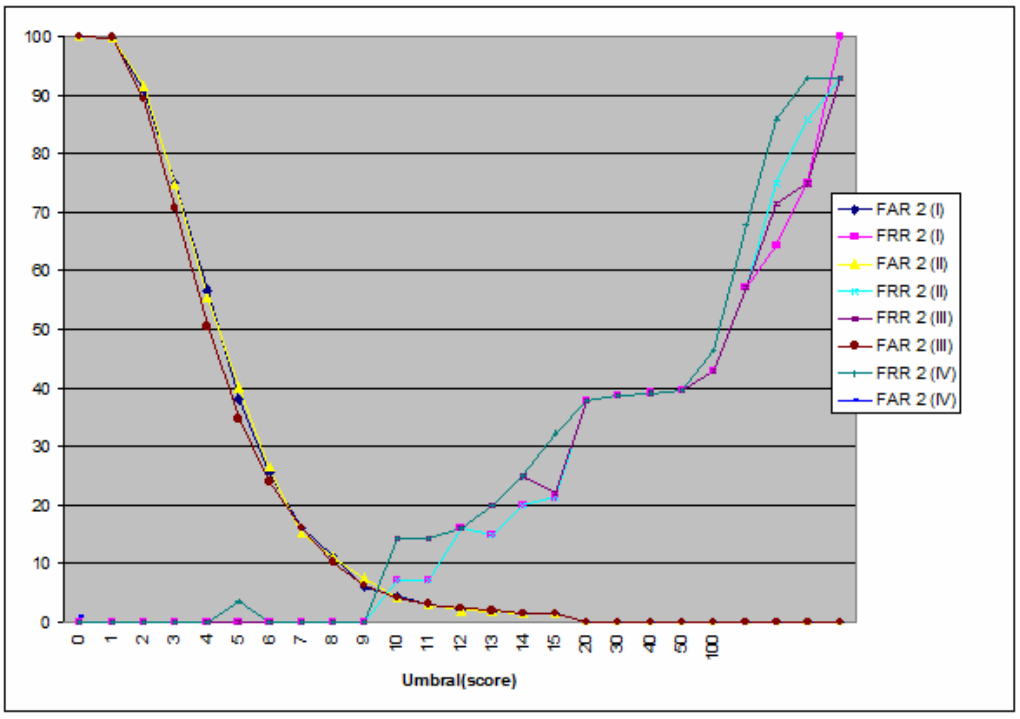


Fig. 72: FAR y FRR con 2 huellas por persona

		andres9.png	andres2.png	atrian8.png	atrian4.png	bernal7.png	bernal5.png
Andrés	andres9.png	800	13	8	5	5	7
	andres2.png	13	800	7	5	6	5
Atrián	atrian8.png	6	7	800	21	5	5
	atrian4.png	6	5	22	800	10	7
Bernal	bernal7.png	6	9	8	8	800	46
	bernal5.png	9	7	7	8	45	800
Fernando	fernando3.png	5	5	5	5	7	6
	fernando9.png	5	5	4	5	5	5
Ibañez	ibañez6.png	10	9	8	8	11	12
	ibañez10.png	8	10	9	7	10	11
Javi	javi7.png	9	6	9	9	15	11
	javi3.png	7	6	11	8	9	9
Lorente	lorente5.png	3	4	6	3	6	6
	lorente2.png	3	3	5	5	5	7
Marcos	marcos10.png	3	3	2	4	3	4
	marcos4.png	2	3	2	4	4	3

Fig. 73: Fragmento de la Excel con 2 huellas por persona

En la Fig. 74, se puede ver las gráficas generadas con la subtablas de 1 huellas (Fig. 75). En la gráfica se han representado los valores FAR y FRR para cuatro subtablas generadas aleatoriamente del conjunto total.

Aquí, las curvas difieren entre sí ya que al coger solo 1 huella por persona, puede haber diferencias de score por el posicionado del dedo. El EER se sitúa entre el umbral 8 y 10. La EER entre el 9% y el 10%.

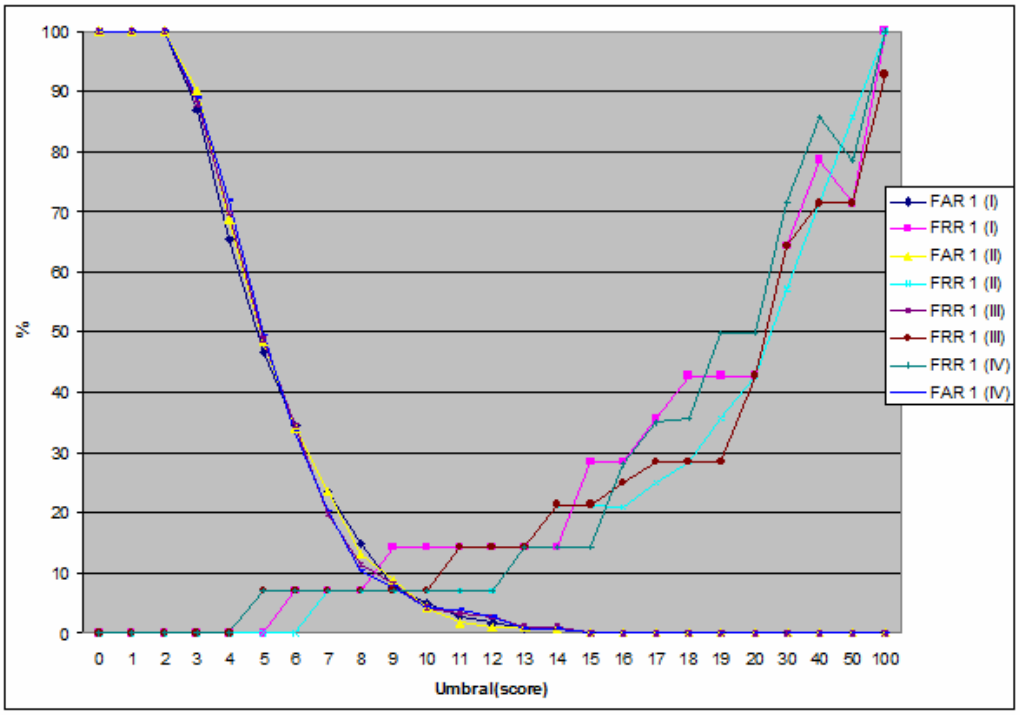


Fig. 74: FAR y FRR con 1 huella por persona

		andres4.png	atrian6.png	bernal4.png	fernando2.png	ibañez2.pr
Andrés	andres8.png	14	6	10	5	10
Atrián	atrian3.png	8	17	8	5	7
Bernal	bernal1.png	8	10	59	8	9
Fernando	fernando8.png	6	4	6	16	7
Ibañez	ibañez7.png	8	9	11	6	28
Javi	javi1.png	7	7	11	7	14
Lorente	lorente2.png	4	3	6	4	4
Marcos	marcos10.png	2	3	4	2	5
Oscar	oscar4.png	3	5	5	3	6
Osuna	osuna4.png	6	4	6	4	4
Pepe	Pepe7.png	3	4	3	4	5
Raquel	Raquel9.png	4	5	8	6	5
Rosa	rosa1.png	2	2	2	2	2
Rubén	ruben9.png	5	7	8	6	8

Fig. 75: Fragmento de la Excel con 1 huella por persona

Esta tasa EER se suele utilizar para definir la calidad de un sistema de reconocimiento de huellas dactilares en particular y de uno biométrico en general [9] [10].

Cuanto menor sea el valor de EER, mejor será el sistema de reconocimiento, siendo el valor obtenido para este proyecto de

EER ~ 10 %

Tal como ya se comentó, el punto EER no tiene porque ser el punto de trabajo del sistema. En este caso, el comparar cada una de las huellas con todas las demás, se puede asimilar a los procesos de verificación de huellas dactilares, en los que se introduce una huella en la base de datos sin saber a quien pertenece y el sistema devuelve los posibles usuarios a los que podría pertenecer en función del umbral fijado. En aplicaciones de verificación forenses, interesa tener una alta FAR, es decir, preferimos aceptar falsos usuarios a condición de que no se nos ”pase” uno autentico.

En otras aplicaciones como la autenticación, el usuario se identifica en el sistema a la hora de registrarse y cuando se quiere autenticar, se tiene que volver a identificar. Tras este proceso, el sistema extrae la huella perteneciente al usuario y la compara con la que el usuario está introduciendo en ese momento. En este caso, interesa FAR baja, es decir, no queremos que alguien pueda suplantar la identidad de otra persona en el sistema, aunque para ello tengamos altas tasas de FRR, es decir, que el usuario tenga que autenticarse varias veces porque el sistema no lo reconozca.

4.4 Conclusiones

En este capitulo se ha explicado el método aplicado en la verificación.

Se ha presentado la forma en la que se han obtenido las tasas FAR, FRR y EER del sistema y se han interpretado y justificado los mismos.

Comparamos nuestro sistema con dos sistemas de reconocimiento de huella dactilar:

- El sistema NFIS2, reconocido como una referencia en verificación de huella dactilar. El sistema NFIS2 fue creado por el *National Institue of Standards and Technology americano* (NIST), obteniendose tasas **EER = 1.47%.** Tasas muy bajas como cabría esperar en sistemas de este tipo.
- Sistema embebido basado en smart-card [13] que realiza la comparación dentro del propio chip (Match-on-Card), obteniéndose tasas **EER = 9.78%.** Una tasa mucho más alta que la anterior, como cabría esperar también de este tipo de sistemas, dada su limitación de procesado y memoria.

Nuestro sistema, se presenta cercano a los sistemas Match-on-Card en cuanto al valor de EER obtenido.

Como conclusión a este capitulo, podemos decir que el método de autenticación mediante la extracción de características de las imágenes obtenidas a través de la cámara del dispositivo móvil, ha dado unos resultados cercanos a sistemas del tipo Match-on-Card, por lo que se pueden considerar aceptables y podría ser objeto de futuros estudios.

5. Conclusiones

En este proyecto se ha abordado el problema de la autenticación de huellas dactilares mediante un dispositivo móvil, a través de las imágenes capturadas por la cámara del mismo.

En primer lugar se ha realizado un estudio del algoritmo a implementar en MatLab, utilizando como huellas las imágenes capturadas por el dispositivo, llegando hasta la fase de extracción de minucias.

En segundo lugar se ha implementado este algoritmo en la aplicación desarrollada hasta la fase de segmentación y realce de la imagen.

En tercer lugar se ha propuesto un método de autenticación alternativo a la extracción de minucias, justificado por motivos de tiempo de procesado. El método propuesto ha sido el de extracción de características de la imagen de la huella.

En cuarto lugar se ha desarrollado la aplicación con este método y se han extraído, con la propia aplicación los resultados del proceso de autenticación de la base de datos del proyecto capturada con el dispositivo.

En quinto lugar se han extraído las tasas de error FAR, FRR y EER de la base de huellas del proyecto y se han interpretado sus resultados.

En sexto lugar, se ha conseguido desarrollar una aplicación en Java para dispositivos móviles con sistema operativo Android. Esta aplicación (Ver anexo B), tiene la opción de generar una comparación de todas las huellas de la base de datos (proceso asimilado a la verificación) y también la de comparar una huella en tiempo real con la almacenada de forma local por el usuario (proceso asimilado a la autenticación). Se ha conseguido desarrollar una aplicación estable, rápida y con unos resultados aceptables sin incorporar software comercial en ninguna de las fases, lo cual era una de las premisas de partida.

Podemos decir que en general se han cumplido los objetivos del proyecto, ya que además se han adquirido conocimientos de Java, Android y de autenticación de huellas dactilares.

6. Futuros Desarrollos

Como futuros desarrollos para mejorar la aplicación actual, cabría mencionar varios aspectos:

1.- Implementar métodos de detección automática del dedo del usuario. De esta manera la aplicación sería capaz de realizar el proceso de autenticación y registro automáticamente.

2.- Mejorar la sensibilidad a la escala y rotación. En el actual desarrollo, la imagen de la huella se encaja dentro de la guía dibujada en la pantalla, de esta manera queda posicionada en el plano X-Y (rotación) y mediante el encaje ocupando el máximo área de la guía, también en el eje Z (escala). Sería una mejora importante implementar por ejemplo el "Matching" para distintos ángulos y obtener el máximo score para cada huella en función del ángulo.

3.- Implementar métodos de "rechazo" de falsas huellas, es decir, intentar detectar que lo que se pone delante de la cámara es un dedo de una persona y no una fotografía, por ejemplo.

4.- Implementar un refuerzo al actual método de detección mediante extracción de características, por ejemplo, esqueletizando y calculando las minucias de una parte muy pequeña y "estratégica" de la imagen de la huella que ha sido seleccionada como huella "ganadora" del proceso de autenticación. Esta pequeña parte de la imagen se podría seleccionar automáticamente de una zona en la que se tuviera buena calidad de crestas y valles. Al ser una zona pequeña el tiempo de procesado podría ser aceptable.

5.- En el proceso de autenticación, o "Local Matching" en nuestra aplicación, se podría mejorar el uso de la misma si después de registrar la huella de un nuevo usuario, se capturara varias veces la huella del mismo usuario para compararla con la huella almacenada. De este modo se podría asociar un umbral de detección a cada huella, e incluso rechazar la huella capturada y solicitar al usuario que la introdujera de nuevo.

7. Bibliografía

[1] Hong, L., Wan, Y., and Jain, A. K. 'Fingerprint image enhancement: Algorithm and performance evaluation'. IEEE Transactions on Pattern Analysis and Machine Intelligence 20, 8 (1998), pp 777-789.

[2] D. Maltoni. "A Tutorial on Fingerprint Recognition, Advanced Studies in Biometrics. Summer School on Biometrics", Alghero, Italy, June 2003. M. Tistarelli, J. Bigun, E. Grosso (Eds.), LNCS 3161 Springer, 2005.

[3] M. Haghighat, S. Zonouz, M. Abdel-Mottaleb, "Identification Using Encrypted Biometrics," Computer Analysis of Images and Patterns, Springer Berlin Heidelberg, pp. 440-448, 2013.

[4] Z. Guo and R. W. Hall, "Parallel Thinning with Two-Subiteration Algorithm", Communications of the ACM, vol. 32, no. 3, March 1989

[5] Fingerphoto Recognition with Smartphone Cameras Chris Stein, Claudia Nickel, Christoph Busch Hochschule Darmstadt - CASED Mornewegstraße 32 D-64295 Darmstadt 2012.

[6] Video-based Fingerphoto Recognition with Anti-spoofing Techniques with Smartphone Cameras Chris Stein[1], Vincent Bouatou[2], Christoph Busch[1] 2013.

[7] A Few Things One Should Know About Feature Extraction, Description and Matching Karel Lenc, Jirı Matas, and Dmytro Mishkin CMP CTU in Prague, Czech Republic 2014.

[8] ORB: an efficient alternative to SIFT or SURF Ethan Rublee, Vincent Rabaud, Kurt Konolige, Gary Bradski WillowGarage, MenloPark, California.

[9] A Novel Method of Score Level Fusion Using Multiple Impressions for Fingerprint Verification. Chunxiao Ren, Yilong Yin, Jun Ma, and Gongping Yang School of Computer Science and Technology Shandong University Jinan, China. Proceedings of the 2009 IEEE International Conference on Systems, Man, and CyberneticsSan Antonio, TX, USA - October 2009.

[10] Performance Evaluation of Fingerprint Verification Systems Raffaele Cappelli, DarioMaio, Member, IEEE, DavideMaltoni, Member, IEEE, JamesL.Wayman, andAnil K. Jain, Fellow, IEEE. IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE, VOL.28, NO.1, JANUARY 2006.

[11] Bruce Eckel PIENSA EN JAVA Segunda edición PEARSON EDUCACIÓN, S.A. Madrid, 2002 ISBN: 84-205-3 192-8

[12] TESIS DOCTORAL: RECONOCIMIENTO AUTOMÁTICO MEDIANTE PATRONES BIOMÉTRICOS DE HUELLA DACTILAR Autor: DANILO SIMÓN ZORITA Director: JAVIER ORTEGA GARCÍA Departamento de Señales, Sistemas y Radiocomunicaciones Escuela Técnica Superior de Ingenieros de Telecomunicación UNIVERSIDAD POLITÉCNICA DE MADRID Año 2003

[13] VULNERABILIDADES EN SISTEMAS DE RECONOCIMIENTO BASADOS EN HUELLA DACTILAR: ATAQUES HILL-CLIMBING -RESUMEN DEL PROYECTO FIN DE CARRERA- XXVII Convocatoria premios "Ingenieros de Telecomunicación" Marcos Martínez Díaz Febrero de 2007.

[14] An Introduction to Biometric Recognition. Anil K. Jain, Arun Ross, Salil Prabhakar. 2004. IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY, VOL.14, NO.1 , JANUARY 2004.

[15] DESARROLLO DE UN SISTEMA DE RECONOCIMIENTO DE HUELLA DACTILAR PARA APLICACIONES MATCH-ON-CARD. AUTOR: Gustavo Francisco Sanz. Área de Tratamiento de Voz y Señales. Dpto. de Ingeniería Informática Escuela Politécnica Superior Universidad Autónoma de Madrid Julio de 2009

[16] Dactiloscopia, estudio sobre la huella dactilar, disponible en: http://es.wikipedia.org/wiki/Dactiloscopia

Universidad Zaragoza
1542

Trabajo Fin de Máster

Autenticación por huella dactilar para dispositivos Android mediante captura de imágenes

ANEXO A
HUELLAS DACTILARES

Autor

Javier Granado Fornás

Director

José Elías Herrero Jaraba

Escuela de Ingeniería y Arquitectura

2014

Anexo A. Huellas Dactilares

Índice de Figuras

A.1 Historia

Existen pruebas que indican que ya se utilizaba la identificación de individuos mediante sus huellas dactilares desde el año 6000 a.C. Se han encontrado restos de cerámica con impresiones de huellas dactilares que apoyan esta creencia.

Hay algunos documentos chinos de esa época que presentan sellos estampados con la huella del pulgar del firmante. También se han encontrado ladrillos usados en las casas de la antigua ciudad de Jericó en los que se había marcado el pulgar de los trabajadores. Sin embargo no existe evidencia de que en aquella época se utilizaran las impresiones de las huellas dactilares como método de identificación.

La primera referencia sobre la estructura de las crestas, valles y poros de las huellas dactilares data de 1684. Fue realizado por Nehemiah Grew. Desde entonces muchos investigadores han trabajado en este campo. En 1823, Purkinje propuso un sistema de clasificación de huellas en nueve clases en función de la configuración de la estructura de las crestas. Hacia 1800 se llegó a dos conclusiones importantes que, aun hoy en día, siguen teniendo vigencia: No existen dos huellas de individuos diferentes con un patrón de crestas coincidente, y la invariabilidad de estos patrones durante la vida del individuo.

En 1900, se admitieron las siguientes premisas como ciertas en el campo de la identificación biométrica:

- La estructura de crestas y valles de cada individuo es única y lo representa unívocamente.

- La estructura de crestas y valles de un individuo, varia dentro de unos límites tan reducidos, que hace posible su identificación sistemática.

- Los detalles de crestas y valles, así como las minucias, son particulares de cada individuo e invariables con el tiempo.

En1864, Nehemiah Grew, publicó el primer tratado científico en el que se explicaba la estructura de crestas valles, lo que llevó a su utilización en la identificación criminal, primero en Argentina, en 1896, en Scotland Yard, en 1901 y en otros países a principios de 1900.

Desde principios de 1900 la identificación de personas a partir de huellas dactilares fue formalmente aceptada y comenzó a ser una practica rutinaria en aplicaciones forenses, instaurándose en todo el mundo, agencias policiales de identificación de huellas dactilares y creándose a su vez, bases de datos criminales.

En 1924, se creó la primera división de identificación de huellas dactilares del FBI.

El reconocimiento automático de huellas dactilares comenzó a principios de los años 60. Desde entonces, los sistemas automáticos de identificación de huellas dactilares se utilizan en las policías de todo el mundo.

Hacia mediados de los 80, con el desarrollo de los ordenadores personales, se comenzaron a utilizar los sistemas automáticos de identificación en aplicaciones no criminales, como los controles de acceso.

A finales de los 90, los dispositivos de estado sólido, con su bajo coste y el desarrollo de algoritmos fiables de reconocimiento de patrones, contribuyeron a la rápida expansión de los sistemas de reconocimiento basado en huellas dactilares.

A.2 Modos de funcionamiento de un sistema biométrico

La necesidad de crear una base de datos en la que queden registrados los patrones biométricos de todos los individuos que son usuarios del sistema determina el modo de funcionamiento llamado modo de inscripción o registro (*Enroll Users* en nuestra aplicación) (Figura 1: Modo Registro (*Enroll*. En esta fase, por tanto, se genera el patrón o modelo biométrico que representa la identidad de cada usuario. En la fase de reconocimiento (*Matching* en nuestra aplicación), dependiendo de cómo interactúe el sistema con el individuo y la base de datos, se distinguen otros dos modos de trabajo: (1) el modo de identificación o clasificación (*Remote Matching*) (Figura 2), y (2) el modo de verificación o autenticación (*Local Matching*) (Figura 3).

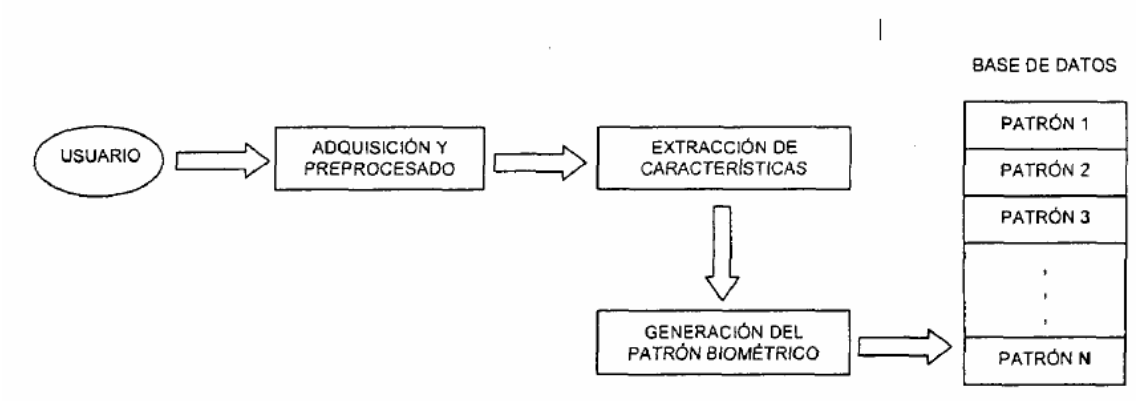


Figura 1: Modo Registro (*Enroll*). Fuente: [12]

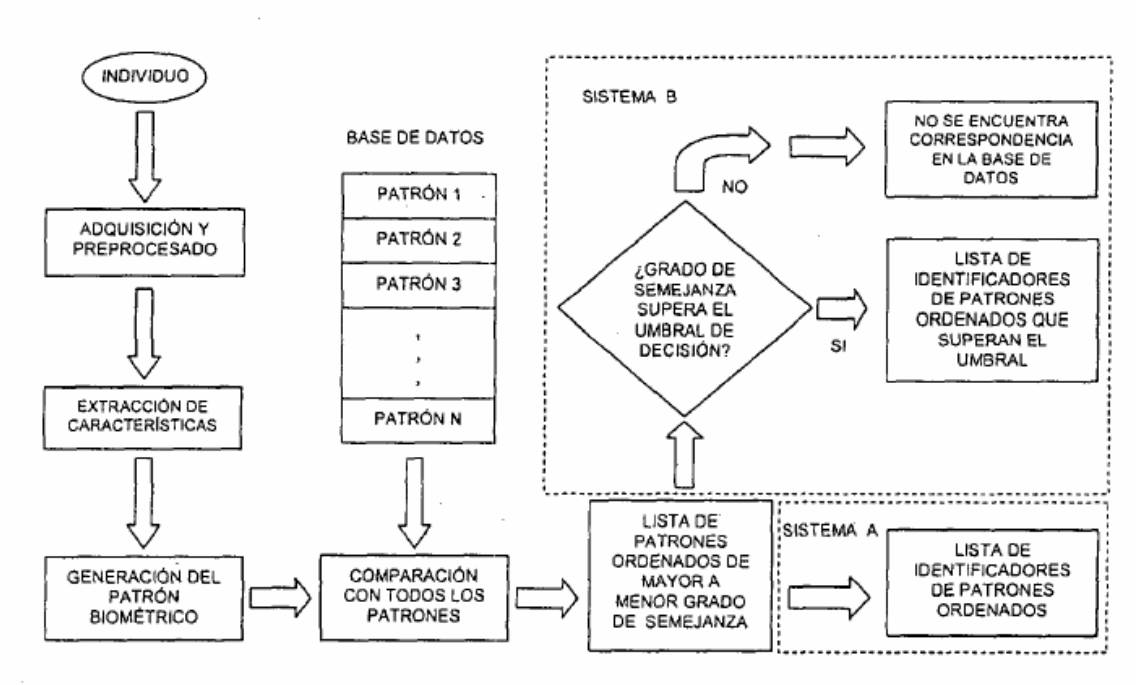


Figura 2: Modo Identificación (Remote *Matching*). Fuente: [12]

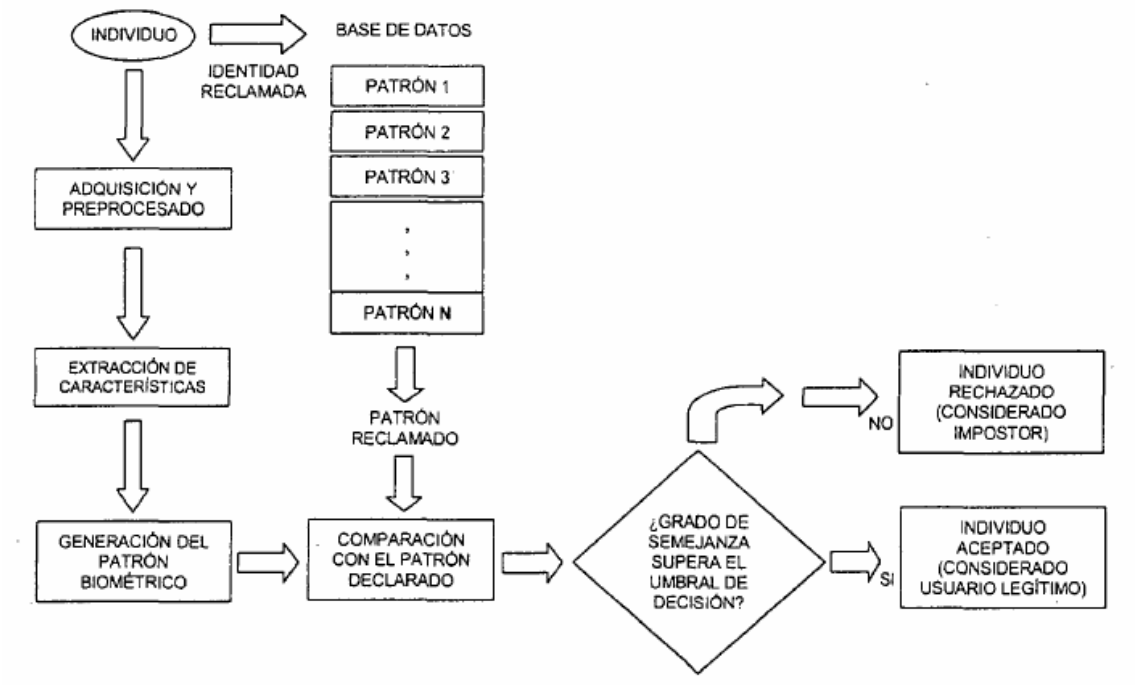


Figura 3: Modo Verificación (*Local Matching*). Fuente: [12]

A.3 Captura de huellas dactilares

La forma en la que se capturan las huellas dactilares es muy dependiente del tipo de aplicación biométrica en la que se van a procesar.

El método de adquisición mediante impresión en tinta, sigue siendo valida en el ámbito judicial y forense. En entornos de control de acceso y seguridad, sin embrago, requieren del uso de técnicas de adquisición en tempo real. En estos casos, las huellas son capturadas con dispositivos electrónicos (sensores de huellas). Dependiendo del principio físico de funcionamiento del sensor, se clasifican en ópticos, de estado sólido y ultrasónicos.

A.4 Características de las imágenes

La imagen de una huella dactilar reproduce el patrón de crestas y valles de la epidermis, y normalmente es adquirida cuando se coloca la yema del dedo sobre la superficie sensible del escáner.

El grosor de las crestas oscila entre 100 y 300 μm, y el periodo espacial de repetición de crestas suele ser de aproximadamente 500 μm. Generalmente, las heridas producidas en la piel, como por ejemplo, las quemaduras superficiales, las abrasiones o los cortes, no afectan a la estructura de crestas, ya que los daños producidos son restaurados con el crecimiento de la nueva piel. En estas condiciones, puede afirmarse que los patrones representados por las imágenes adquiridas contienen toda la información biométrica necesaria, y estable en el tiempo, para poder reconocer de forma automática a un individuo. Las características que presentan las imágenes digitales proporcionadas por un escáner de huella dactilar son:

- ***Resolución***: dependiendo del tipo del dispositivo de captura empleado, la resolución de las imágenes obtenidas varía, desde 250 dpi hasta 625 dpi., siendo un estándar la resolución de 500 dpi. El valor de 250 dpi es la resolución mínima que permite efectuar con precisión la extracción de puntos característicos {minucias) de la estructura de crestas de una huella dactilar.

- ***Área de captura***: es el tamaño de la superficie sensible del escáner. Cuanto mayor es el área de captura del sensor, mayor es la zona del dedo que puede adquirirse y, por tanto, mayor es el número de características identificativas que pueden obtenerse. La impresión completa de la yema del dedo implica, generalmente, un área de captura de 1 a 1.25 pulgadas (6.5 a 10.1 cm^2). La tendencia a reducir el tamaño de los dispositivos de captura en algunas aplicaciones conlleva también

la necesaria reducción del tamaño de los sensores. El tamaño más pequeño que permite la extracción de un número de características suficientemente identificativo es de 0.5 pulgadas (1.6 cm^2). En estos casos, la aparición de un elevado grado de variabilidad /intra-clase puede empeorar el funcionamiento del sistema de reconocimiento, debido a un aumento del número de falsos rechazos.

- ***Número de píxeles***: conocidas, la resolución R en dpi, y el área de captura del sensor AxB (alto x ancho) en pulgadas cuadradas, el número de píxeles viene dado por: R·AxR·B píxeles.

- ***Rango dinámico***: es el número de valores posibles de luminancia de la imagen y está relacionado con el número de bits utilizados para codificar la luminancia de cada píxel. El color no aporta información al reconocimiento por lo que no se adquiere. El estándar actual es el de representar las imágenes capturadas en escala de grises de 8 bits/píxel, lo que supone un rango dinámico de O a 255.

A.5 Tipos de Sensores

En la adquisición de huellas en tiempo real, estas se obtienen colocando el dedo sobre la superficie sensible del sensor electrónico. Atendiendo al principio físico de funcionamiento del sensor se clasifican en:

- ***Ópticos***: Basados en la reflexión de la luz sobre la yema del dedo, basados en fibra óptica, electro-ópticos y los sensores sin contacto (dentro de estos últimos podríamos clasificar nuestro método de captura de imagen en el proyecto).

La técnica de captura más utilizada es la FTIR. Consiste en que cuando el dedo se apoya sobre la superficie de cristal del sensor, un diodo LED proyecta un haz de luz por debajo del cristal. La luz atraviesa el cristal e incide sobre las crestas de la huella, dispersándose y reflejándose de manera aleatoria en múltiples direcciones.

La luz que incide en el interior de los valles se refleja en una determinada dirección. Esta luz se enfoca mediante lentes hacia un dispositivo CCD o CMOS, capturándose así la imagen de la huella dactilar.

- ***Ultrasónicos***: Estos sensores presentan una imagen muy clara de la huella, mediante la proyección sobre la superficie de la huella de pulsos ultrasónicos. El eco que se refleja permite determinar la profundidad del relieve. Presentan el inconveniente del tamaño y su elevado coste.

- ***Sensores basados en fibra óptica***: En este caso, los sensores disponen de fibras ópticas que hacen incidir perpendicularmente, un haz

de luz por debajo del cristal sobre el que se apoya el dedo. La luz reflejada incide sobre un sensor CCD o CMOS.

- ***Sensores electro-ópticos***: Estos sensores están formados por dos capas. La primera formada por un polímero que emite luz proporcional al voltaje aplicado en una de sus caras. El dedo se coloca en la cara opuesta dando lugar a diferencias de potencial entre crestas y valles, originando diferencias en la emisión de luz. La segunda capa está formada por fotodiodos en toda su superficie, que capturan la luz procedente de la primera capa, proporcionando la imagen digital de la huella.

- ***Sensores sin contacto***: Aquí se incluyen las técnicas de captura con cámara, en las que no se produce contacto físico entre el dedo y el sensor. No introducen deformación elástica pero tienen la dificultad de obtener imágenes bien enfocadas. Nuestro proyecto estaría incluido es este grupo, ya que realizamos de la huella con la cámara del dispositivo móvil.

- ***Sensores de estado sólido***: Presentan la ventaja de no necesitar ningún componente óptico. Se pueden distinguir cuatro tipos:

Capacitivos. Los cuales se forman por la distribución de microcondensadores (unos 10.000) en una superficie plana, sobre la que se deposita un dieléctrico. Todas las placas conductoras de una de las caras del dieléctrico están eléctricamente unidas. Las del otro lado se forman cuando el dedo se coloca sobre la superficie. La medida del voltaje de estos condensadores determina la imagen de la huella dactilar, ya que dicho valor depende de la distancia entre placas. Los condensadores formados por las crestas presentan mayor voltaje, al estar más próximas a la superficie que los valles. Tienen el inconveniente de que deben limpiarse a menudo, ya que la grasa y la suciedad empeoran mucho la calidad de la imagen.

Térmicos. Construidos con materiales termoeléctricos capaces de crear corrientes a partir de diferencias de temperatura. El sensor se mantiene caldeado eléctricamente a una temperatura ligeramente superior a la del dedo. El contacto con el dedo origina diferencias de temperatura entre crestas y valles.

De campo eléctrico. Formados por un anillo emisor de señal sinusoidal de baja potencia, bajo el que se dispone una matriz de micro antenas receptoras. La amplitud de señal recibida por cada antena se modifica al producirse el contacto con el dedo.

Piezoeléctricos: En estos, la superficie es sensible a la presión ejercida durante el contacto dedo-sensor. Tienen el inconveniente de no ser muy sensibles a las pequeñas diferencias de relieve entre las crestas y los valles y además, la imagen que proporcionan es binaria, lo que supone una perdida de información.

A.6 Clasificación de huellas dactilares

Las huellas dactilares se clasifican fundamentalmente según la distribución de sus crestas y valles en la zona central de la huellas.

La zona de interés, normalmente se circunscribe al área formada por las crestas y valles existentes entre dos crestas llamadas de referencia. Estas crestas definen dos puntos singulares denominados Delta y Núcleo (figura 1).

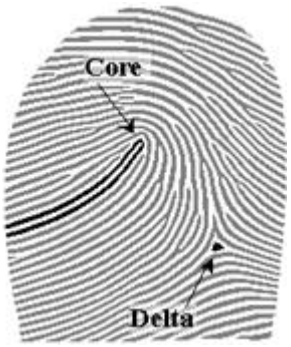


Figura 4. Fuente: redyseguridad.fi-p.unam.mx

En función de cómo se desarrollen las crestas y valles entre estos puntos singulares podemos distinguir varios tipos de huellas (figura 2):

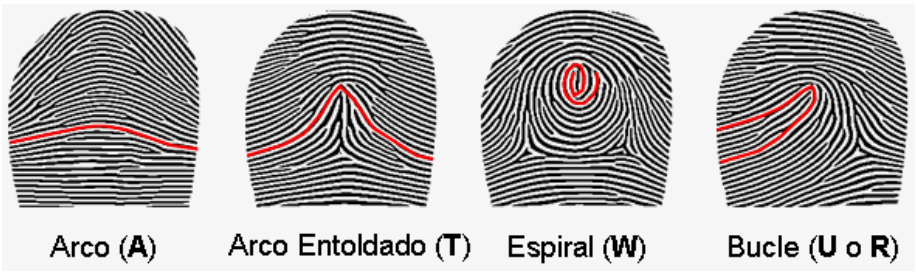


Figura 5: Clases de huellas. Fuente: www.monografias.com

Aún hoy en día sigue siendo un problema complejo la elección de criterios universales de clasificación.

A.7 Minucias

Uno de los patrones biométricos de huella dactilar más utilizados, por su elevada fiabilidad, es el patrón de minucias [*Ratha et al., 1995*]

Recibe el nombre de minucia cualquier punto de la imagen que indica que una determinada cresta presenta un final/comienzo o una bifurcación. Una minucia estará determinada, por tanto, por sus coordenadas espaciales dentro de la imagen (Figura 6).

Generalmente, los patrones biométricos están constituidos por las coordenadas espaciales de cada minucia, los ángulos de orientación de las crestas asociadas a las mismas (una minucia pertenece siempre a una cresta), y las coordenadas espaciales de un número finito de puntos muestreados de dichas crestas.

En la mayoría de las ocasiones, la calidad de las imágenes proporcionadas por los sensores en la etapa de adquisición de las huellas dactilares no es suficiente para garantizar la correcta extracción de sus características biométricas. Lo que se traducen en:

- La aparición de discontinuidades a lo largo de las líneas que definen las crestas.
- La aparición discontinuidades transversales a dos o más líneas de crestas.
- La falta de paralelismo entre las crestas. La presencia de ruido puede provocar la falsa unión de crestas entre sí (conexión de crestas). Como consecuencia de todas las imperfecciones derivadas de la baja calidad de las imágenes, las minucias detectadas por los algoritmos de extracción de características pueden no coincidir con las minucias reales, ya que se produce:
  - La pérdida de determinadas minucias presentes en la estructura de crestas (minucias borradas o eliminadas)
  - La inserción de minucias falsas, no presentes en la estructura de crestas (minucias espurias)
  - Error en la determinación de las coordenadas de las minucias y sus orientaciones en la imagen.

El objetivo de esta etapa de procesado es el de proporcionar una imagen de la huella dactilar, cuyo grado de calidad permita la extracción precisa y fiable de las características biométricas.

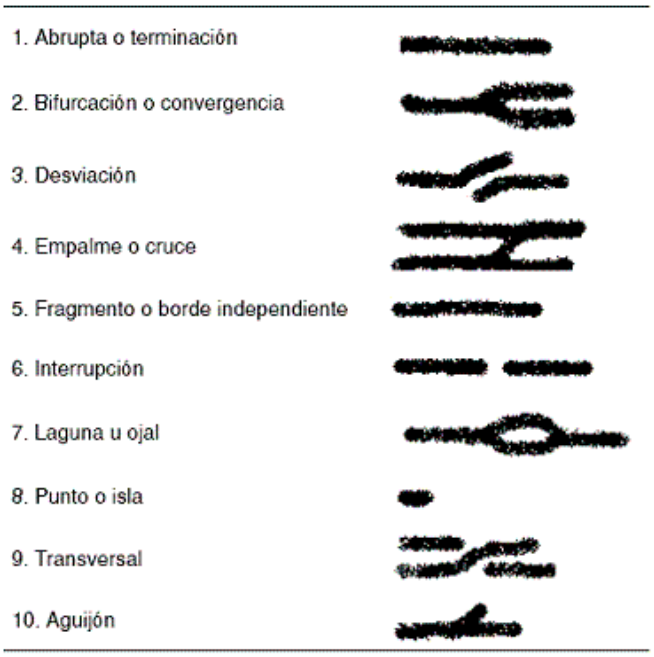


Figura 6: Clases de minucias. Fuente: www.dolthink.com

Una minucia estará determinada, por tanto, por sus coordenadas espaciales dentro de la imagen. Generalmente, los patrones biométricos están constituidos por las coordenadas espaciales de cada minucia, los ángulos de orientación de las crestas asociadas a las mismas (una minucia pertenece siempre a una cresta), y las coordenadas espaciales de un número finito de puntos muestreados de dichas crestas.

Universidad Zaragoza
1542

Trabajo Fin de Máster

Autenticación por huella dactilar para dispositivos Android mediante captura de imágenes

ANEXO B
ESTRUCTURA DE LA APLICACIÓN

Autor

Javier Granado Fornás

Director

José Elías Herrero Jaraba

Escuela de Ingeniería y Arquitectura

2014

Anexo B. Estructura de la aplicación

Índice de Figuras

B.1 El Sistema Operativo Android

Android es el sistema operativo para dispositivos móviles desarrollado por Google. Android entró en el mercado en el año 2008 y está basado en el kernel[5] de Linux[6] 2.6, el cual controla servicios del sistema operativo como la seguridad, la gestión de memoria o los procesos.

Android es una plataforma de código abierto por lo que su distribución es libre, pudiéndose modificar su código fuente.

Android fue desarrollado inicialmente por Google y aunque posteriormente se unió a la Open Handset Alliance (formada por T-Mobile, Intel, Samsung, HTC o Nvidia), Google ha publicado la mayor parte del código fuente.

Para el desarrollo del software en esta plataforma se dispone de un SDK[7] y un plugin[8] que se integran dentro del entorno de desarrollo Eclipse.

En este entorno se incluye un emulador para facilitar el desarrollo de la aplicación. Con Android, el desarrollador tiene acceso a todas las funciones que ofrece el dispositivo móvil de una forma sencilla.

Las versiones de Android reciben, en inglés, el nombre de diferentes postres (Figura 7). En cada versión el postre elegido empieza por una letra distinta, conforme a un orden alfabético:

[5] En informática, un núcleo o kernel (de la raíz germánica Kern, núcleo, hueso) es un software que constituye una parte fundamental del sistema operativo,

[6] Linux es un núcleo o kernel de sistema operativo libre y uno de los principales ejemplos de software de código abierto.

[7] Un kit de desarrollo de software o SDK (siglas en inglés de software development kit)

[8] Un complemento o plugin es una aplicación que se relaciona con otra para aportarle una función nueva y generalmente muy específica.

Figura 7: Evolución del sistema operativo Android. Fuente: luisjoseb.blogspot.com.es

La arquitectura del sistema operativo Android (Figura 8), esta diseñada por capas.

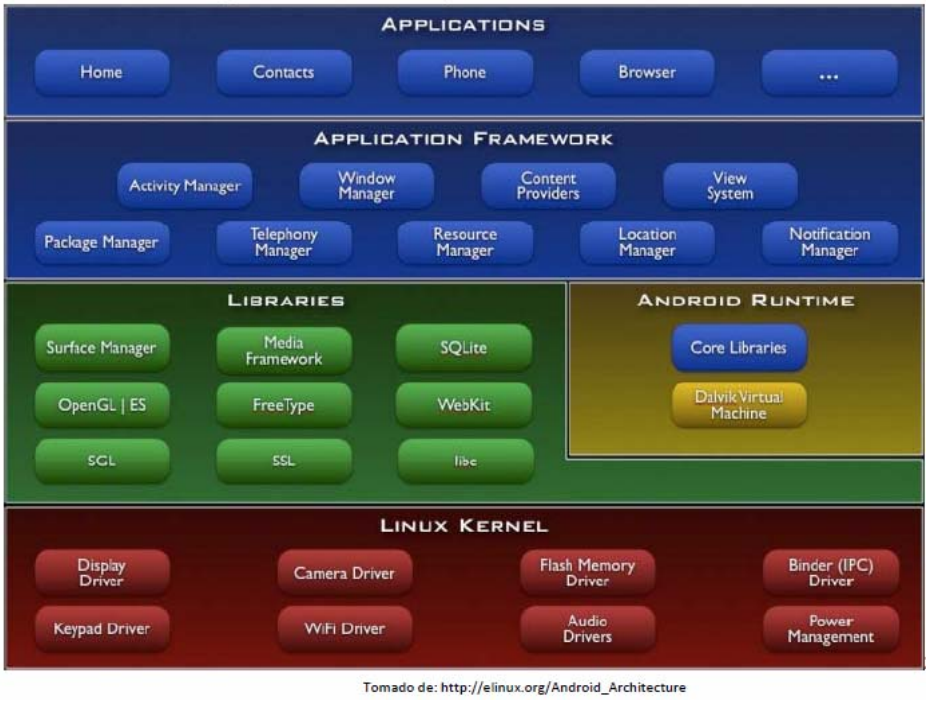


Figura 8: Arquitetura del sistema operativo Android. Fuente: http://elinux.org/Android_Architecture

Las ***Applications*** (aplicaciones), están desarrolladas en Java[9] y pueden estar formadas por algunos de los siguientes bloques: *Activities, Intents, Broadcast, Services y Content Providers.* En esta capa es donde está incluida la aplicación *FingerMatching* desarrollada en este proyecto.

[9] Java es un lenguaje de programación originalmente desarrollado por Sun Microsystems, adquirida por Oracle, para aplicaciones software independientes de la plataforma.

El ***Application Framework*** es el marco de aplicaciones. La arquitectura está diseñada para que la reutilización de componentes sea sencilla, ya que cualquier aplicación puede dar un perfil público a sus datos o capacidades para que otra aplicación haga uso de esas cualidades.

El sistema incluye un conjunto de ***Libraries*** (Bibliotecas) en C y C++ que proporcionan la mayor parte de las funcionalidades presentes y que son utilizadas por varios de los componentes del sistema Android. Estas capacidades son expuestas a los desarrolladores a través del framework descrito anteriormente para que puedan ser utilizadas.

Al mismo nivel que las bibliotecas de Android, se sitúa el ***Android runtime*** (entorno de ejecución), compuesto por bibliotecas que proporcionan la mayor parte de las funcionalidades de Java además de incluir la máquina virtual. Cada aplicación, al ejecutarse crea su propio proceso con su propia instancia de la máquina virtual Dalvik, que ha sido escrita de manera que un mismo dispositivo pueda lanzar múltiples instancias de dicha máquina virtual de forma eficiente.

La capa más cercana al hardware corresponde al núcleo de Android que está basado en el ***kernel de Linux 2.6***. Android utiliza este kernel como una abstracción del hardware disponible en cada terminal conteniendo los drivers necesarios para utilizar cualquier parte de este hardware a través de las llamadas correspondientes. Android también depende de este kernel para gestionar los servicios base del sistema como pueden ser la seguridad, gestión de memoria, de procesos, la pila de red y el modelo de driver.

Una aplicación Android se ejecuta en su propio proceso Linux. Este proceso es creado cuando parte del código necesita ser ejecutado, y continuará vivo hasta que ya no sea requerido y el sistema reclame su memoria para asignársela a otra aplicación. Una característica importante y poco usual de Android es que el tiempo de vida de un proceso no es controlado directamente por la aplicación, sino que es el sistema quien determina el tiempo de vida basado en el conocimiento que tiene de las partes de la aplicación, la importancia de la misma y cuánta memoria hay disponible. Por lo tanto, el usuario no tiene conocimiento sobre estas áreas ya que según la naturaleza de Android son tareas del sistema operativo.

B.2 Características Hardware

El dispositivo utilizado para el desarrollo de la aplicación es un SmartPhone Nexus 5 de LG (Figura 9), con las siguientes características técnicas:

- Pantalla de 4,95 pulgadas capacitiva y 1920x1080 (445 ppp) Full HD.
- Cámara frontal de 1.3 MPx y trasera de 8 MPx con Flash.
- Procesador: Qualcomm Snapdragon™ 800 de 4 núcleos a 2.26 GHz.
- Procesador Gráfico: Adreno™ 330 a 450 MHz.
- Almacenamiento interno: 16 GB (SDCARD) y 2 GB de RAM.
- Sensores: GPS, Giroscopio, Acelerómetro, Brújula.
- Sistema operativo: Android 4.4 Kit Kat.

Figura 9: SmartPhone Nexus 5 utilizado en el desarrollo del proyecto. Fuente: http://www.google.es/nexus/5/

B.3 Flujo de la aplicación

B.4 Activity Principal

Figura 10

Esta es la pantalla de la primera actividad con la que arranca la aplicación (Figura 10). En ella se puede ver 5 botones:

Local Matching: Nos lleva a la actividad donde se realiza el proceso de autenticación local, es decir, la comparación de la huella que presentamos ante la cámara contra la que fue guardada en la memoria del teléfono mediante el proceso de Enroll Users.

Remote Matching: Nos lleva a otra actividad donde se puede realizar o bien la comparación de huellas (2 a 2) almacenadas en DropBox, o bien la comparación de todas las huellas para generar una Excel con los resultados (scores) de las mismas.

Enroll Users: Nos lleva a la actividad donde se registra un nuevo usuario.

About Fingerprint: Nos lleva a una pantalla donde se nos muestra una breve explicación de la aplicación.

Exit: Con este botón salimos de la aplicación

B.5 Activity “Enroll Users”

Como hemos comentado, en esta actividad se registran nuevos usuarios en el sistema. En ella (Figura 11) podemos ver en la parte superior un cuadro donde introduciremos el nombre del usuario a registrar. Posteriormente y presionando el botón *Take User Photo* se lanza un Intent (llamada) a la cámara nativa de Android. Esto significa que la aplicación hace una llamada a la aplicación de la cámara que viene integrada en el sistema operativo. Al capturar la foto (usualmente la cara de la persona a registrar), la cámara devuelve el control a nuestra aplicación pasándole la imagen capturada, la cual visualizamos en la pantalla de nuestra aplicación.

Después, podemos capturar una huella de esa persona, para lo cual presionaremos el botón *Take User Fingerprint*, entrando en otra actividad (Figura 12), en la que capturaremos la imagen de la huella que una vez procesada (Figura 13), será guardada en el teléfono.

Figura 11

Figura 12

Figura 13

Esta imagen de la huella dactilar se guarda en formato *.png que es un formato de compresión sin perdidas. La imagen ocupa unos 16 kb.

La imagen de la huella se guarda con el nombre introducido anteriormente, de manera que cada usuario registrado produce dos ficheros, *Nombre_usuario*.jpg que contiene la imagen capturada de la cara de la persona a registrar y *Nombre_usuario*.png que contiene la imagen de la huella de la persona a registrar.

Estas imágenes se guardan en un directorio de la memoria del teléfono (SdCard/storage/emulated/0/Pictures/PFM/USERS).

El directorio *PFM/USERS* es el directorio de la aplicación. La primera vez que se ejecuta, la aplicación va a buscar este directorio y si no lo encuentra, lo crea (Figura 14).

Figura 14

Hay que destacar que la captura de la huella no se hace a través de una llamada a la cámara nativa (Intent), como en el caso de la captura de la foto del usuario. Esto es así debido a que en este caso necesitamos tener una interfaz de usuario personalizada. Como se ve en la imagen (Figura 12) se ha desarrollado una actividad para capturar imágenes en la que hay dos botones (Get y Back) para capturar la imagen y para regresar a la actividad anterior respectivamente, además de pintar una "guía" (de color azul) que permite al usuario "encajar" el dedo en la posición correcta. El rectángulo rojo se pinta a titulo informativo ya que es el área que se recorta y con la que al final nos quedamos. Esta

cámara personalizada se ha desarrollado con la API[10] de Android *Android.hardware.*

Respecto al botón *Take User 10 Fingerprint*, tiene la misma función que el botón *Take User Fingerprint*, solo que como su nombre indica, captura 10 imágenes de la misma huella a intervalos de 500 ms (Figura 15). La captura de 10 huellas se utilizó para crear la base de huellas del proyecto. Estas huellas, posteriormente se almacenaron en una carpeta de DropBox creada a tal efecto. Se almacenaron un total de 140 huellas pertenecientes a 14 personas de las cuales se capturaron 10 imágenes de la huella de su dedo índice.

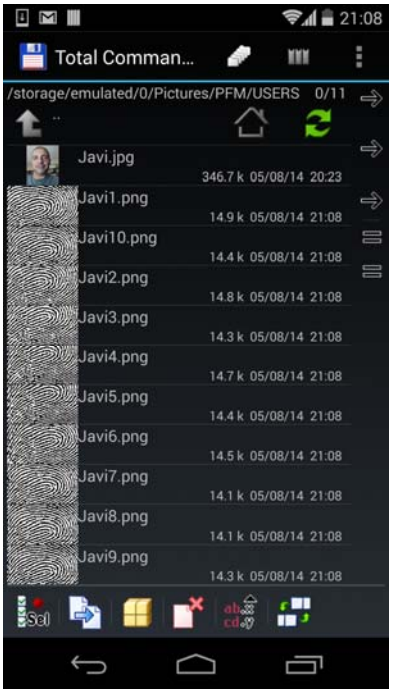


Figura 15

[10] API (Application Programming Interface) es el conjunto de funciones y procedimientos (o métodos, en la programación orientada a objetos) que ofrece una biblioteca para ser utilizado por otro software como una capa de abstracción

B.6 Activity "Local Matching"

Cuando se presiona el botón *Local Matching* en la aplicación principal (Figura 10), saltamos a otra actividad en la que se nos muestra una pantalla similar a la de la actividad *Enroll Users* (Figura 12), pero en este caso el botón "*Go!*" de la derecha (Figura 16) lanza el proceso de comparación o *Matching* entre la imagen capturada del dedo colocado bajo la cámara y todas las huellas que contenga el directorio *PFM/USERS*. En primer lugar se captura un frame de la imagen de la huella que se ha colocado bajo la cámara (que es la huella a autenticar), una vez obtenida una imagen recortada, segmentada y realzada (Figura 13), se procede a ejecutar el algoritmo de *Matching* explicado en el capitulo 3 entre esa huella y cada una de las que haya en el directorio. Al acabar el proceso con todas las huellas, la aplicación comprueba que huella ha obtenido la mayor puntuación o numero de matches (score) y si ese valor es igual o mayor que el umbral fijado, se supone que esa huella corresponde con la que se esta mostrando a la cámara y la aplicación devuelve (a través de otra actividad) (Figura 17) un mensaje "*Matching Success!!!*" y muestra la imagen de la persona asociada a la huella ganadora.

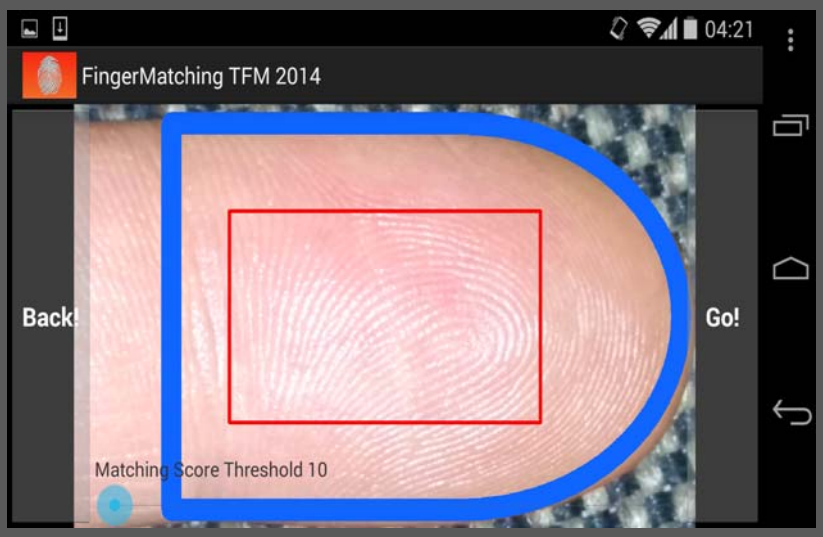


Figura 16

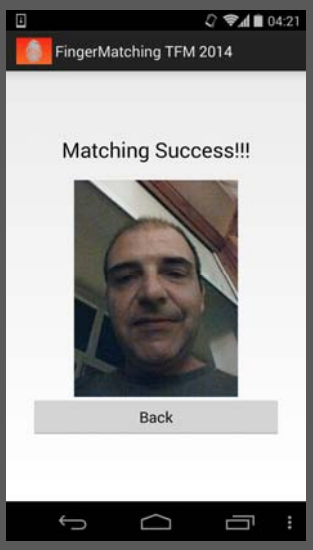


Figura 17

Este umbral esta fijado por defecto en 10, pero puede ser ajustado entre 0 y 30 en la propia aplicación con la "barra deslizante" (*Matching Score Thershold*) que aparece en la pantalla de la actividad. En principio

el número de huellas registradas no está limitado, aunque cada comparación cuesta alrededor de 500 ms y lógicamente al aumentar el número de huellas registradas, aumenta en esa misma proporción el tiempo que tarda la aplicación en completar el proceso.

Si ninguna huella de las registradas cumple la condición de tener un score igual o mayor que el umbral fijado, la aplicación devuelve un mensaje "*No Matching Found!*" en la misma pantalla (Figura 16) y permite volver a intentar la autenticación, por ejemplo, reposicionando el dedo y volviendo a presionar el botón "Go!".

Como ya se explicó, este proceso de "*Local Matching*" esta asimilado a la autenticación, en la que cuando el usuario se identifica, la aplicación recupera la huella almacenada de ese usuario y la compara con la huella capturada en tiempo real. Este proceso sería equivalente a registrar solo un usuario aunque como decimos, en la aplicación no se ha puesto límite al número de usuarios registrado. Esto permite ver como se comporta la aplicación ante el registro de usuarios con distinta calidad de huella y que presentan distintos "scores", lo que presenta problemas a la hora de fijar un umbral que funcione bien para todos. Como ya se vio en el capitulo 5, para nuestra base de huellas el EER obtenido esta entorno al valor 10 (número de coincidencias o score entre huellas), que es el umbral fijado por defecto en nuestra actividad "*Local Matching*".

En una aplicación de autenticación "pura", el usuario tendría que identificarse primero, por ejemplo, introduciendo un PIN asociado con su huella en el proceso de registro (*Enroll Users*), con lo cual la huella capturada en tiempo real se compararía con la huella almacenada de ese usuario y solo con esa.

B.7 Activity "Remote Matching"

La actividad "*Remote Matching*" nos envía a una pantalla en la que podemos elegir entre "*Matching from DropBox*" o "*Create Scores to Excel File*".

Las dos actividades se apoyan en el hecho de que disponemos de una base de huellas previamente capturada con la aplicación como ya se ha comentado, que está alojada en DropBox. En la aplicación se ha implementado el acceso a DropBox mediante la integración del *Android Chooser SDK*, que es una librería que se integra en el proyecto y que permite el acceso de la aplicación a DropBox.

B.7.1 Activity "Matching from DropBox"

Seleccionando la opción "*Matching from DropBox*" (Figura 18), pasamos a otra pantalla en la que podemos seleccionar la primera y la segunda huella de DropBox a comparar y un botón para lanzar el "*Matching*" entre ambas (Figura 19). Al presionar el botón "*Select first FingerPrint*" o "*Select Second FingerPrint*" la aplicación accede vía Internet a nuestra carpeta creada en DropBox y que previamente hemos enlazado en nuestro proyecto mediante clave encriptada para evitar que otra aplicación acceda a esa carpeta. Esto nos permite "navegar" por la carpeta (Figura 20) y (Figura 21) y poder elegir 2 huellas de entre las 140 que componen la base de huellas de nuestro proyecto.

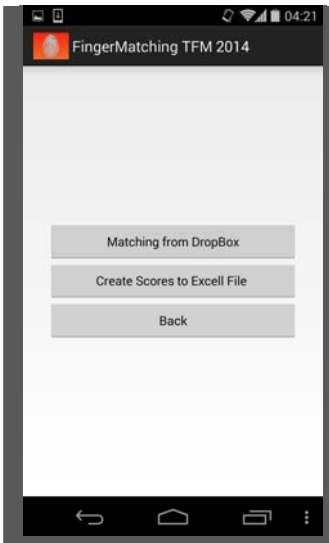


Figura 18

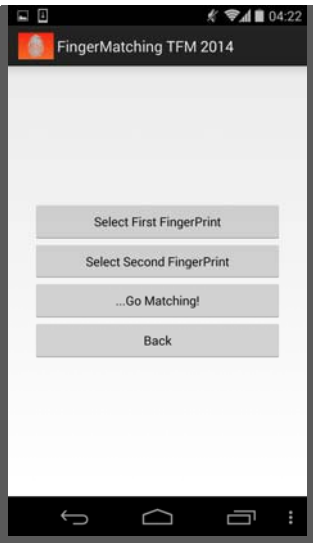


Figura 19

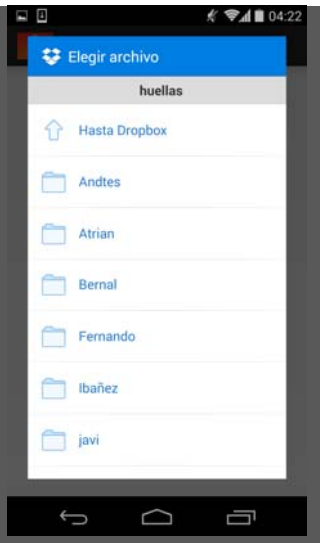


Figura 20

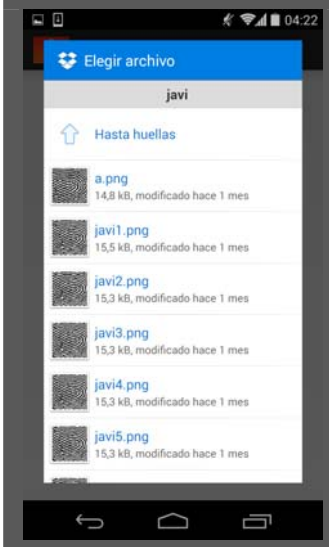


Figura 21

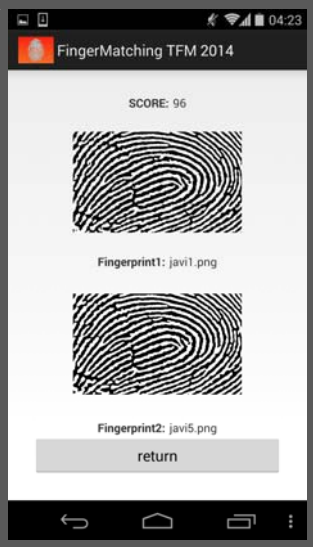


Figura 22

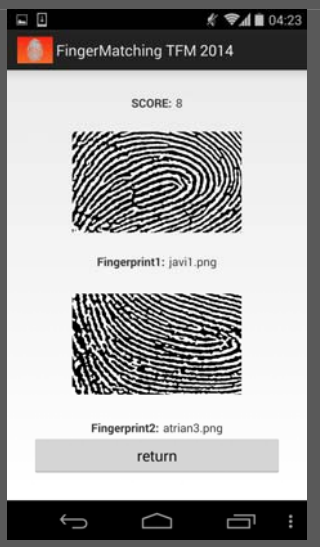


Figura 23

Esta actividad permite, de una forma visual, comprobar como se comporta el algoritmo de autenticación implementado de una manera sencilla y rápida, ya que en la base de huellas los usuarios están organizados en carpetas por su nombre y cada huella lleva el nombre del usuario seguido de un número correlativo. La actividad devuelve el resultado en una pantalla con las imágenes de las huellas comparadas, sus nombres y el "score" obtenido (Figura 22) y (Figura 23). La base de datos puede ir creciendo hasta el límite de la capacidad de almacenamiento, que actualmente está en 2 Gb para la cuenta gratuita, es decir, con un tamaño de huella de unos 15 Kb, podríamos almacenar unas 130.000 huellas.

B.7.2 Activity "Create Scores to Excel file"

Seleccionando la opción "*Create Scores to Excel File*" (Figura 18), pasamos a otra pantalla que nos permite generar una Excel con los resultados (scores) de comparar todas las huellas de la base de huellas dos a dos (Figura 24) y (Figura 25).

Esta actividad se implementó para poder extraer todos los valores que nos servirían para obtener las tasas FAR, FRR y EER explicados en el capítulo 4.

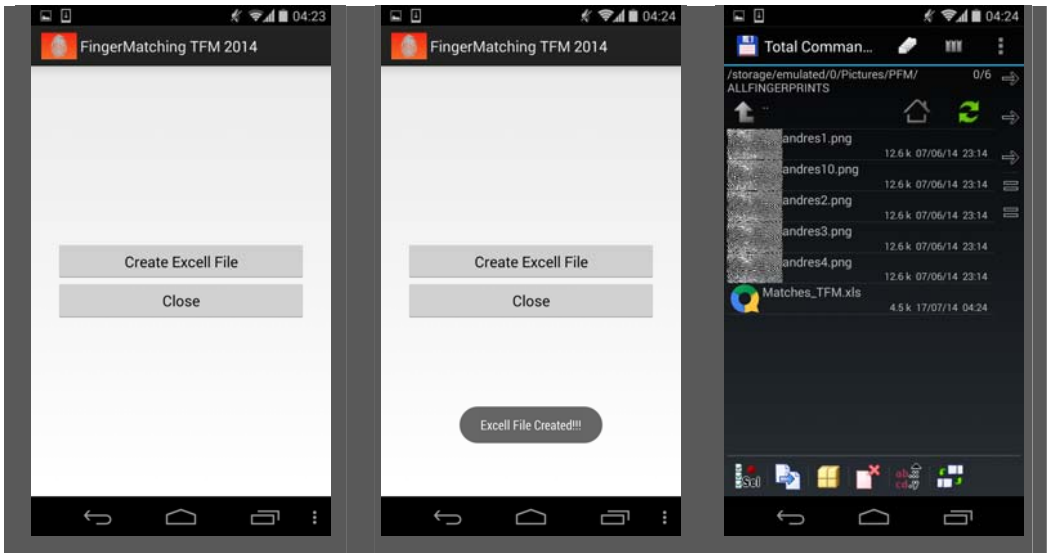
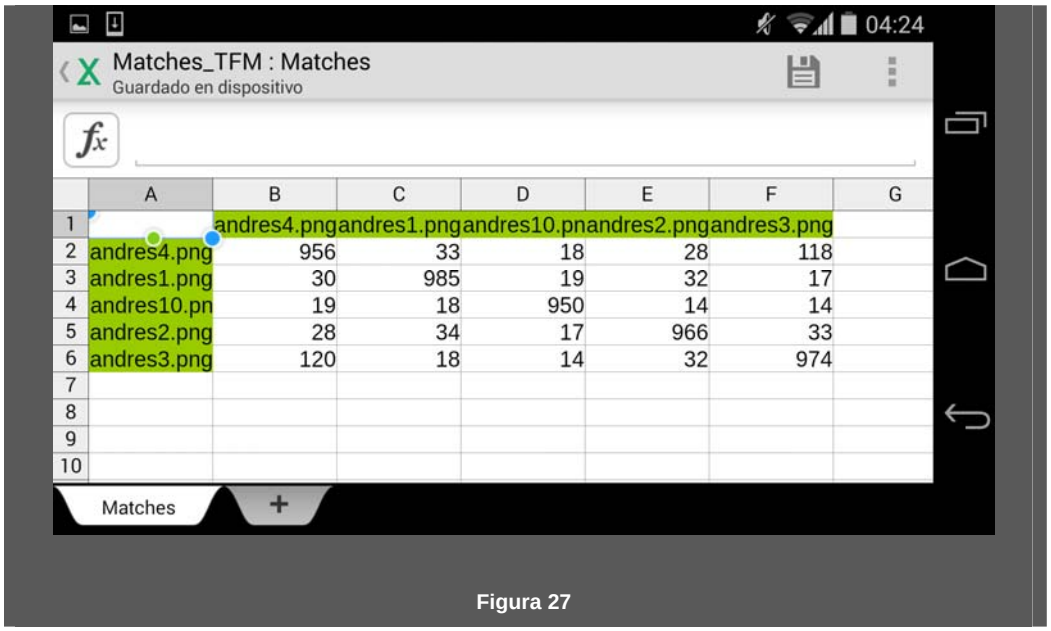


Figura 24 Figura 25 Figura 26



	A	B	C	D	E	F	G
1		andres4.png	andres1.png	andres10.pn	andres2.png	andres3.png	
2	andres4.png	956	33	18	28	118	
3	andres1.png	30	985	19	32	17	
4	andres10.pn	19	18	950	14	14	
5	andres2.png	28	34	17	966	33	
6	andres3.png	120	18	14	32	974	
7							
8							
9							
10							

Figura 27

Para generar esta Excel es necesario descargar al teléfono todas las huellas almacenadas de la base de huellas. Las huellas se descargan a un directorio creado a tal fin (…/PFM/ALLFINGERPRINTS) (Figura 26), en el que la aplicación va a buscar las huellas y una vez generada la Excel se almacena en ese directorio (Figura 27).

Para generar la Excel fue necesario integrar en nuestro proyecto Android la librería *Apache POI*, una librería de Java puro que se utiliza para interactuar con hojas de cálculo de distintos formatos. Consiste en API's para manipular formatos de ficheros basados en el formato Microsoft OLE2.

B.8 Entorno de desarrollo

El entorno de programación utilizado para el desarrollo de la aplicación fue eclipse (versión Kepler) (Figura 28). El SDK (Software Development Kit) de Eclipse incluye las herramientas de desarrollo de Java, ofreciendo un IDE con un compilador de Java interno y un modelo completo de los archivos fuente de Java.

El Desarrollo de Programas para Android se hace habitualmente con el lenguaje de programación Java y el conjunto de herramientas de desarrollo (SDK).

El SDK de Android, incluye un conjunto de herramientas de desarrollo. Comprende un depurador de código, biblioteca, un simulador

de teléfono basado en QEMU, documentación, ejemplos de código y tutoriales. Las plataformas de desarrollo soportadas incluyen Linux (cualquier distribución moderna), Mac OS X 10.4.9 o posterior, y Windows XP o posterior. La plataforma integral de desarrollo IDE (Integrated Development Environment) soportada oficialmente es Eclipse junto con el complemento ADT (Android Development Tools plugin).

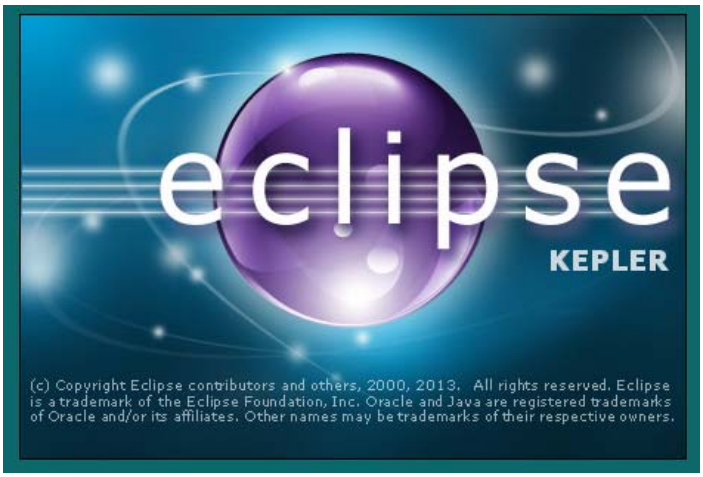


Figura 28

Para poder trabajar con esta plataforma en el entorno de desarrollo Eclipse se ha necesitado la instalación del SDK de Android. El SDK proporciona las bibliotecas de cada API, así como herramientas de desarrollo necesarias para construir, testear y depurar aplicaciones para Android. Esto se realiza mediante un gestor del SDK (SDK Manager) que se integra en el programa, desde el cual se puede elegir la versión del API que se desea descargar para trabajar con ella (Figura 29).

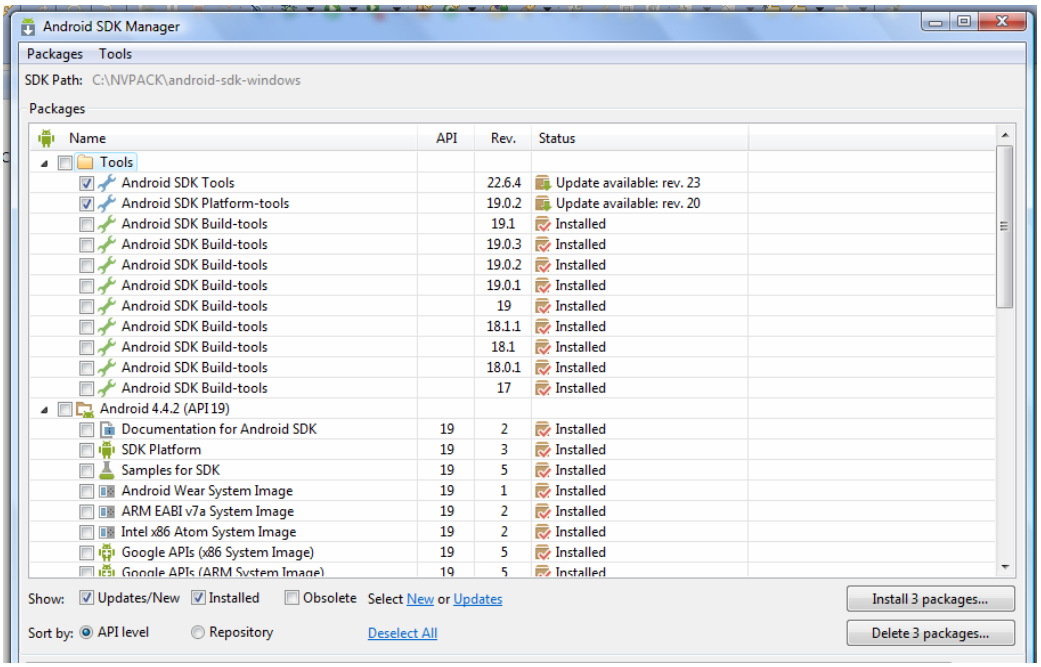


Figura 29

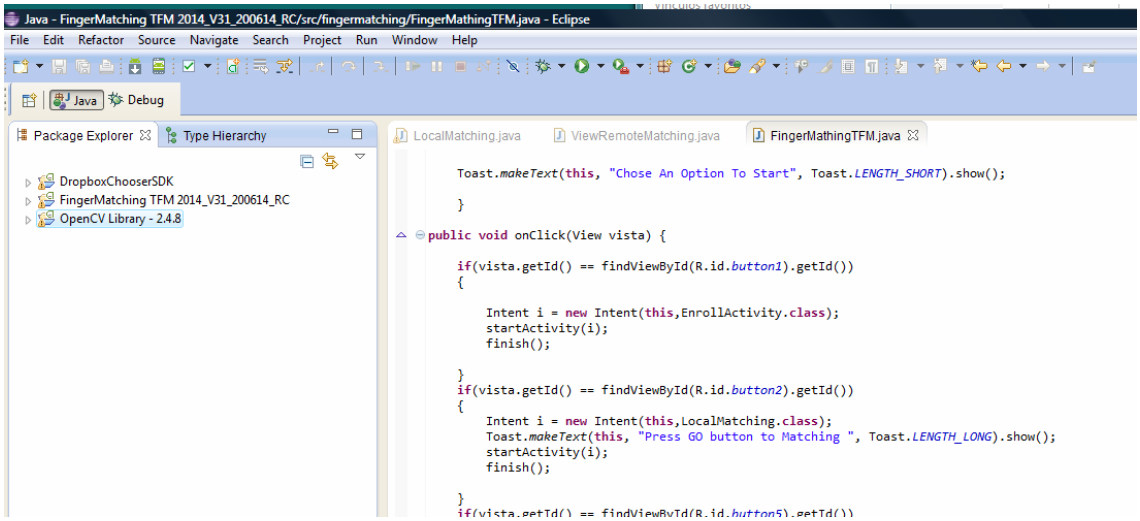


Figura 30

En la Figura 30 se puede ver el entorno eclipse en el que a la izquierda tenemos nuestro proyecto ***FingerMatching TFM 2014_V31_200614_RC*** junto con el ***DropboxChooserSDK*** para el manejo de DropBox como ya explicamos y por supuesto la biblioteca ***OpenCV Library-2.4.8*** con la que hemos implementado todo el tratamiento de imagen de la aplicación.

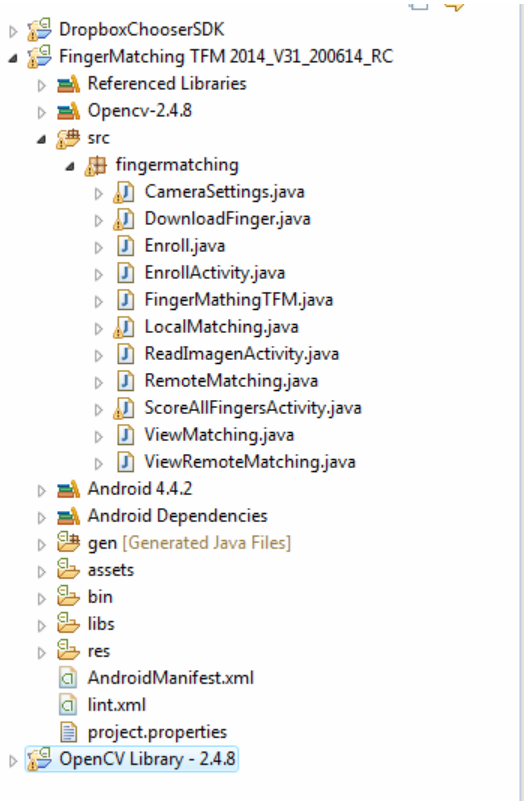


Figura 31

En la Figura 31 vemos nuestro proyecto expandido. La carpeta más importante es ***src***, abreviatura de fuente en inglés (source), es donde se incluyen las actividades o clases dentro del paquete de la aplicación (fingermatching). Son archivos java, en los cuales se implementa toda la codificación lógica de la aplicación.

El archivo más elemental de una aplicación Android es una actividad. las actividades llevan asociada siempre una pantalla para interaccionar con el usuario por lo que hay que crear una ventana para a continuación introducir una interfaz. Esta interfaz es la llamada vista (View en Inglés). Desde la aplicación más simple a la más compleja todas se estructuran de la misma manera. En nuestro caso se pueden ver en la figura 22, que nuestro proyecto tiene 11 actividades:

- CameraSettings.java: En ella se inicializan los parámetros relativos a la cámara, zoom, sensibilidad, autofocus, uso del flash, etc.

- DownloadFinger.java: Gestiona la conexión cifrada con DropBox y la descarga de las huellas seleccionadas hacia el teléfono.

- EnrollActivity.java: Pantalla principal del proceso de *Enroll*, gestiona la llamada (Intent) a la cámara nativa para la captura de la foto del

usuario, permite la introducción del nombre y lanza un *Intent* a la actividad *Enroll.java*.

- Enroll.java: Gestiona la captura de la imagen de la huella a través del manejo de la cámara mediante la API androi.hardware, procesa la imagen y guarda en la SdCard del teléfono la imagen del usuario junto con su huella.

- FingerMatching.java: es la actividad que se lanza al arrancar la aplicación. Básicamente, está compuesta pon Intents que llaman al resto de las actividades en función del botón que se presione.

- LocalMatching.java: Se encarga de capturar una imagen de la huella con la cámara y compararla con el resto de las que están guardadas. Da como resultado el "score" entre las dos y si determina que alguna de las huellas almacenadas corresponde con la huella presentada a la cámara lanza la actividad *ViewMatching.java.*

- ReadmagenActivity.java: Es la actividad encargada de presentar el "*About FingerMatching*", un texto donde se explica brevemente el funcionamiento de la aplicación que incluye un *scroll (deslizador)* para mover el texto arriba y abajo.

- RemoteMatching: Pantalla principal de la opción "*Remote Matching*" que a su vez nos permite elegir lanzar "*DownLoadFinger.java*" o "*ScoreAllFingersActivity.java*".

- ScoreAllFingersActivity.java: Compara todas las huellas descargadas en el teléfono y genera la Excel con todos los "scores".

- ViewMatching.java: Presenta en la pantalla la foto del usuario autenticado junto con el mensaje "*Matching Success!!!".*

- ViewRemoteMatching: Realiza la comparación de las dos huellas descargadas de DropBox y presenta en pantalla las imágenes de ambas huellas, con su nombre y el "score" obtenido.

Además en el proyecto se puede distinguir también otros elementos:

- **Gen**: son archivos de configuración y de gestión de datos. Estos archivos se van generando automáticamente según se va desarrollando la aplicación en las clases del src. NO se debe modificar manualmente, el entorno de desarrollo lo hace automáticamente.

- **Android X.X.X**: Versión de Android utilizada. En el caso de la aplicación creada ha sido Android 2.4.4. Aquí se almacenan todas las funciones de las bibliotecas de esa API.

- **Android Dependencies**: Donde se almacenan las bibliotecas implementadas como por ejemplo la opencv.jar y la dropboxchoosersdk.jar de este proyecto.

- **Assets**: Donde se implementan nuevos cambios de la imagen predeterminada por el sistema, como cambios de tipo de letra mediante un .ttf.

- **Bin**: Donde se almacenan los archivos al construir la aplicación.

- **Res**: Es el abreviado de recursos en inglés (resources), y como su propio nombre indica almacena todos los recursos de la aplicación como sonidos, imágenes, etc.… En ella es donde se encuentran los diseños (layout en inglés) que almacenan las vistas (views en inglés) de las imágenes, archivos .XML que se diseñan para darle la apariencia deseada a la actividad, etc.

- **Android Manifest**: Manifiesto de Android en español. Es el documento más importante del programa. Se encuentra en la raíz del proyecto y actúa como descriptor de implementación de la aplicación.