Universidad Autónoma Metropolitana Unidad Azcapotzalco
División de Ciencias Básicas e Ingeniería

Proyecto Terminal de Ingeniería en Computación

Implementación de biometría vascular del dedo

Proyecto que presenta:

Ángel Gaytan Cruz

Para obtener el titulo de:

Ingeniero en Computación

Asesores de Proyecto:

Oscar Alvarado Nava
Departamento de Electrónica

Eduardo Rodríguez Martínez
Departamento de Electrónica

Mexico, D.F. **Diciembre 2013**

Resumen

En el presente reporte se presenta el desarrollo de un sistema biométrico capaz de realizar una autentificación personal confiable, a través de una biometría vascular del dedo.

Un sistema biométrico es todo aquel que realiza labores de biometría de manera automática. En otras palabras, se trata de sistemas basados en medir y analizar las características físicas y del comportamiento humano con propósito de autentificación.

El objetivo del presente sistema biométrico es identificar a las personas por medio del patrón de las venas del dedo. Para ello es necesario adquirir la imágen del dedo mediante luz infrarroja, a dicha imágen se le aplican algoritmos de deteccion de bordes, aldelgazamiento y filtrado para obtener el patrón de las venas bien definido. Despues se hace la extracción de minucias, formando el vector caracteristico para cada persona. Para la identificación de las personas se utilizo el algoritmo de distancia de Hausdorff, para determinar a quien corresponde cada patrón de venas.

Los algoritmos empleados durante el desarrollo del sistema fueron implementados con ayuda de la biblioteca OpenCV o en su caso se desarrollaron bajo el lenguaje de programación C.

Índice general

Índice de figuras

Capítulo 1

Introducción

1.1. Introducción

El concepto clásico de biometría denota la aplicación de técnicas matemáticas y estadísticas al análisis de datos en las ciencias biológicas. Dentro del contexto tecnológico, la biometría expresa la aplicación automatizada de técnicas biométricas a la certificación, autentificación e identificación de personas en sistemas de seguridad. Las técnicas biométricas se utilizan para medir características físicas o de comportamiento de las personas con el objetivo de establecer una identidad[1].

Los sistemas biométricos se han convertido en una solución ideal para necesidades de identificación y autentificación personal, ya que que son de alta precisión y usan una parte del cuerpo; actualmente se están adoptando en todo el mundo. Los investigadores han determinado que el patrón vascular de algunas partes del cuerpo humano es único para cada individuo, difícil de falsificar, sin cambios por la edad y sin verse afectado por el maltrato de la piel[2].

Actualmente la biometría se presenta en un sin número de aplicaciones, demostrando ser, posiblemente, el mejor método de identificación humana.

1.2. Justificación

En la actualidad existen varios métodos de autentificación personal, tales como números de identificación personal, contraseñas o claves de tarjetas inteligentes, que están basados en algo que el usuario tiene o conoce; sin embargo estos métodos han demostrado ser poco fiables. Es por ello que la autenticación biométrica hoy en día es objeto de continuo estudio y arduo desarrollo, con el fin de erradicar las desventajas de los métodos tradicionales.

En este proyecto terminal se realizo el diseño y la implementación de un método en particular de autentificación personal basado en los patrones de las venas del dedo, con el propósito de obtener un sistema que ofrece un alto grado de confiabilidad y precisión en la autentificación personal.

1.3. Objetivo general

Diseñar e implementar una biometría vascular del dedo capaz de realizar una autentificación personal.

1.4. Objetivos particulares

- Diseñar e implementar un módulo que permita adquirir la imágen del dedo.
- Diseñar e implementar un módulo para el pre-procesamiento de la imágen.
- Diseñar e implementar un módulo de extracción de características.
- Diseñar e implementar un módulo de verificación.

1.5. Organización del proyecto

El trabajo se encuentra organizado de la siguiente manera:

- El capítulo 1 brinda un panorama general del sistema biométrico que se desarrollo, la importancia que tiene y los objetivos que se persiguen.
- El capítulo 2 se concreta en explicar los conceptos elementales del presente trabajo, así como brindar un panorama de los sistemas biométricos que actualmente se utilizan en la industria.
- En el capítulo 3 se describe a detalle cada uno de los módulos por los cuales esta compuesto el sistema biométrico, así como los algoritmos que se utilizaron.
- En el capítulo 4 se describen las pruebas que se realizaron con el sistema biométrico implementado, así como los resultados que se obtuvieron.
- En el capítulo 5 se presentan las conclusiones del trabajo.

Capítulo 2

Marco Teórico

2.1. Conceptos generales

2.1.1. Biometría

La biometría se define como la ciencia mediante la que se puede identificar a una persona basándose en sus características biofísicas o de comportamiento, es decir algo que el ser humano posee de manera intrínseca. El termino biometría comprende un amplio espectro de tecnologías mediante el uso de las cuales se permite verificar la identidad de una persona a partir del análisis de la medida de estas características, confiando en atributos propios de cada individuo en lugar de cosas que conocen o poseen.

La biometría presenta diversas ventajas respecto a las tecnologías desarrolladas en base a los otros dos factores, sobre todo en el hecho que en ningún momento se le puede perder u olvidar a una persona su característica biométrica, siendo solo susceptible de falsificación, y en muchos casos como se verá en la descripción de cada uno de los diferentes tipos de tecnologías biométricas, esta posibilidad de fraude al sistema ha sido estudiada y eliminada.

La capacidad para la identificación biométrica llevada a cabo por cualquier persona es algo innato, ya que siempre se ha utilizado esta técnica de manera inconsciente para reconocer a una persona por las calles o en una fotografía por ejemplo. Los distintos tipos de tecnología biométrica varían en complejidad, capacidades y modo de funcionamiento y pueden ser usadas para verificar o establecer la identidad de una persona. Sin embargo comparten igualmente elementos comunes. Todos hacen uso de dispositivos de adquisición de información adaptados a cada caso en particular, tales como cámaras o escáneres para obtener imágenes, registros o medidores, al igual que de software y hardware adaptado para extraer, codificar, almacenar y

comparar las características de las medidas.

La tecnología es por tanto capaz de aprovechar rasgos característicos de las personas, ya sean de carácter físico o de comportamiento, para llevar a cabo el reconocimiento de una persona de manera automática.

2.1.2. Sistema biométrico

Un sistema biométrico es aquel que realiza la identificación de personas basándose en patrones biométricos, es decir que el sistema mide o analiza una determinada característica física o de comportamiento de una persona, luego la procesa y la codifica, para después de esto y basado en algún algoritmo entregar una respuesta. En la actualidad la biometría tiene muchas aplicaciones y funciona con muchos parámetros.

En un sistema biométrico típico como se muetra en la Figura 2.1, la persona se registra en el sistema cuando una o más de sus características físicas y de conducta son obtenidas, procesadas por un algoritmo númerico, e introducida en una base de datos. Idealmente, cuando entra, casi todas sus características concuerdan; entonces cuando alguna otra persona intenta identificarse, no empareja completamente, por lo que el sistema no le permite el acceso.

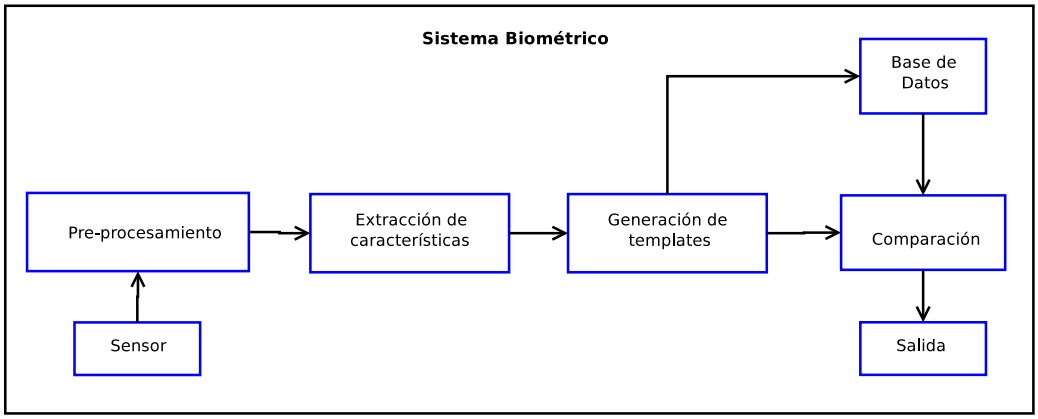


Figura 2.1: Diagrama de flujo de Sistema Biométrico.

2.2. Tipos de sistemas biométricos

Actualmente existen varios tipos de sistemas biométricos, a continuación se describen algunos de ellos.

2.2.1. Reconocimiento de huella dactilar

La comparación de la huella dactilar es una de las técnicas más antiguas y ampliamente utilizadas y aceptadas a nivel globlal. Los sitemas actuales de comparación de la huella dactilar tienen su base en los desarrollos realizados por Galton y Purkinje.[4, 5]

La mayoría de los sistemas de reconocimiento de huellas dactilares (Figura 2.2), se hallan englobados dentro de los sistemas AFIS[1] conformando un elemento indispensable dentro de las investigaciones policiales.

Las técnicas de reconocimiento de huellas se dividen en dos categorías:

- Las técnicas locales basadas en las minucias y
- Las globales basadas en la correlación

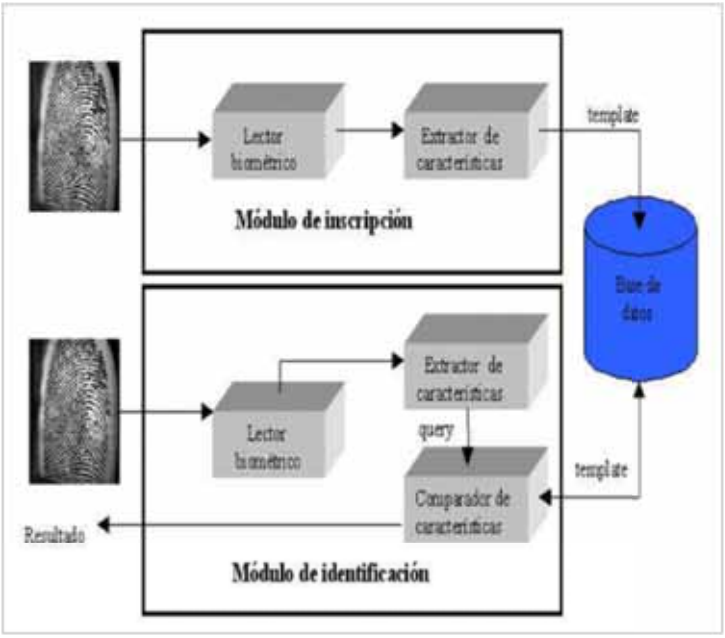


Figura 2.2: Sistema biométrico de huella dactilar.

[1] Acrónimo en inglés *Automated Fingerprint Identification Systems*

El principal inconveniente de la primera aproximación radica en la difícil tarea de extracción de las minucias en imágenes de baja calidad, mientras que la segunda, se ve claramente afectada por traslaciones y rotaciones.

2.2.2. Reconocimiento de iris

La identificación basada en el reconocimiento de iris es más moderna que la basada en patrones retinales; desde hace unos años el iris humano se viene utilizando para la autenticación de usuarios.

Se captura una imagen del iris en blanco y negro, en un entorno correctamente iluminado, usando una cámara de alta resolución.

Generalmente esto se hace mirando a través de la lente de una cámara fija, la persona simplemente se coloca frente a la cámara y el sistema automáticamente localiza los ojos, los enfoca y captura la imagen del iris, ésta imagen se somete a deformaciones pupilares (el tamaño de la pupila varía enormemente en función de factores externos, como la luz) y de ella se extraen patrones, que a su vez son sometidos a transformaciones matemáticas hasta obtener una cantidad de datos suficiente para los propósitos de autenticación.

El iris humano (el anillo que rodea la pupila, que a simple vista diferencia el color de ojos de cada persona) es igual que la vasculatura retinal, una estructura única por individuo que forma un sistema muy complejo (de hasta 266 grados de libertad) e inalterable durante toda la vida de la persona.

El uso por parte de un atacante de órganos replicados o simulados para conseguir una falsa aceptación es casi imposible con análisis infrarrojo, capaz de detectar con una alta probabilidad si el iris es natural o no.

2.2.3. Reconocimiento facial

Los sistemas de reconocimiento facial (Figura 2.3) extraen los rasgos faciales de los usuarios para su identificación. La fuente para realizar la identificación puede ser tanto imágenes fotográficas como de vídeo. La identificación se puede hacer en 2D, 3D o una combinación de ambas. El objetivo de un sistema de reconocimiento facial es, generalmente, el siguiente: dada una imagen de una cara «desconocida», o imagen de test, encontrar una imagen de la misma cara en un conjunto de imágenes «conocidas», o imágenes de entrenamiento. La gran dificultad añadida es la de conseguir que este proceso se pueda realizar en tiempo real.

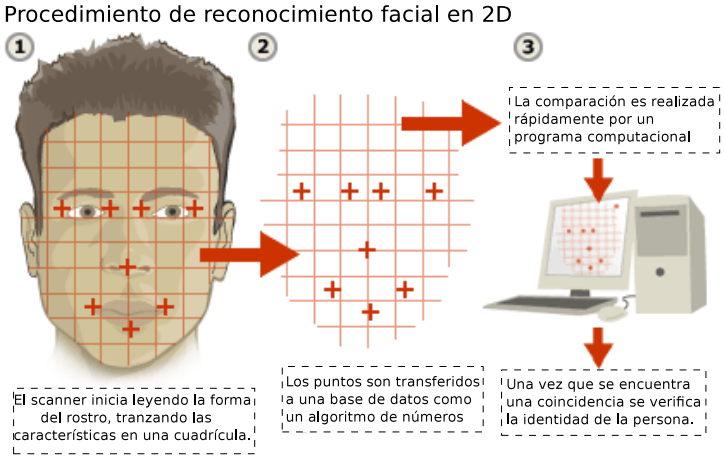


Figura 2.3: Sistema de reconocimiento facial.

2.2.4. Reconocimiento de voz

La voz es otra característica que las personas utilizan comúnmente para identificar a los demás. Es posible detectar patrones en el espectro de la frecuencia de voz de una persona que son casi tan distintivos como las huellas dactilares. Tan solo basta recordar las veces en que se reconoce a alguien conocido por teléfono para comprender la riqueza de esta característica como método de reconocimiento.

Los sistemas de verificación mediante la voz (Figura 2.4) "escuchan" mucho más allá del modo de hablar y el tono de voz. Mediante el análisis de los sonidos que se emiten, los tonos bajos y agudos, vibración de la laringe, tonos nasales y de la garganta, también crean modelos de la anatomía de la tráquea, cuerdas vocales y cavidades. Muchos de estos sistemas operan independientemente del idioma o el acento de la persona.

Figura 2.4: Sistema de reconocimiento de voz.

2.2.5. Comparación de los sistemas biométricos

A continuación se puede observar una comparación de diferentes características propias de los sistemas biométricos.

	Voz	Escritura Firma	Huellas Dactilares	Ojo Retina	Ojo Iris	Geometría de Mano
Fiabilidad	alta	alta	alta	muy alta	muy alta	alta
Facilidad de Uso	alta	alta	alta	baja	media	alta
Prevención de Ataques	media	media	alta	muy alta	muy alta	alta
Aceptación	alta	muy alta	media	media	media	alta
Estabilidad	media	media	alta	alta	alta	media
Identificación	no	si	si	si	si	
Autenticación	si	si	si	si	si	si
Interferencias	ruidos, resfriados	firmas fáciles	suciedad, heridas	irritación	gafas	artritis, reumatismo

Figura 2.5: Comparación de sistemas biométricos.

Capítulo 3

Desarrollo del sistema biométrico

3.1. Descripción técnica

Como se muestra en la Figura 3.1, el sistema de autentificación biométrico consta de cuatro módulos, los cuales se describen a continuación.

- **Módulo de adquisición de imágen.** Es el encargado de obtener la imágen del dedo, que nos brinda el primer acercamiento al patrón de las venas, mediante luz infrarroja y una camara web.

- **Módulo de pre-procesamiento.** En este módulo se aplican diferentes técnicas de procesamiento a la imágen, con el objetivo de eliminar características que no son de utilidad y obtener una imágen que nos muestre el patrón de las venas del dedo bien definido.

- **Módulo de extracción de características.** La función que tiene este módulo es extraer los puntos en donde las venas se intersectan, con el fin de generar la plantilla que identifica a cada individuo y que es almacenada de forma inicial en la base de datos. De la misma forma cuando se lleva a cabó el proceso de autentificación, este módulo le brinda al módulo de verificación el patrón que se compara con las plantillas existentes en la base de datos.

- **Módulo de verificación.** Este módulo realiza una comparación entre las plantillas que se almacenaron en la base de datos y el patrón que genera el módulo de extracción de características, y dado un grado de parentesco determina la identidad de la persona.

  Cabe destacar que el objetivo de la base de datos es almacenar las plantillas que identifican a cada usuario, por lo tanto se maneja como un directorio dentro del mismo sistema de archivos, sin necesidad de utilizar un manejador de bases de datos específico.

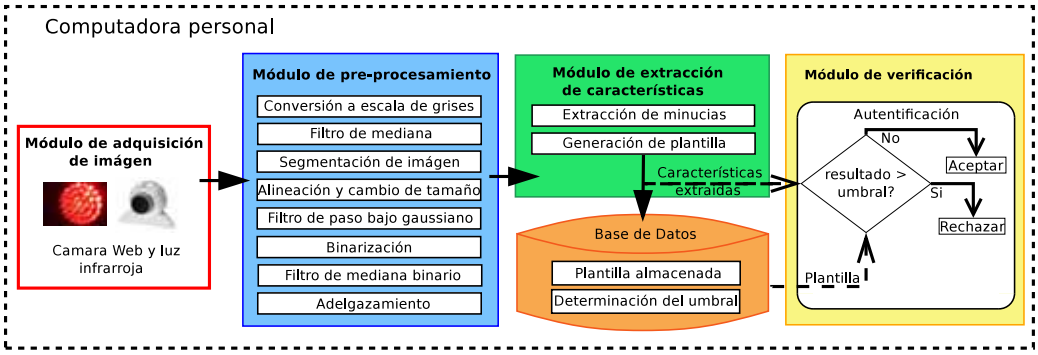


Figura 3.1: Sistema de autentificación biométrico.

3.2. Especificación técnica

3.2.1. Adquisición de la imágen

El módulo de adquisición de imágen se implementa con una cámara web tradicional y luz infrarroja, ya que este tipo de luz debido a su longitud de onda, que va desde los 700nm a los 1000nm, puede pasar a través del tejido humano y al mismo tiempo es absorbida completamente por la hemoglobina existente en la sangre. La imágen que se obtiene (Figura 3.2) es de 400x200 pixeles en formato JPEG.

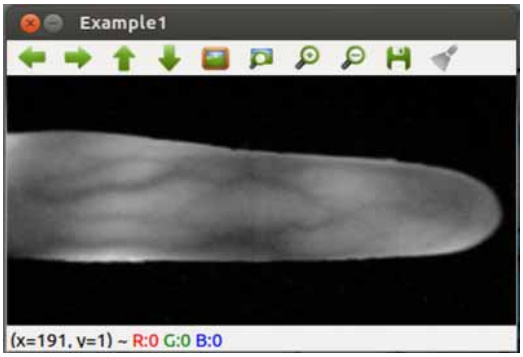


Figura 3.2: Primer acercamiento al patrón de las venas.

3.2.2. Pre-procesamiento

Conversión a escala de grises

La imágen original que se obtiene esta en modo RGB e indica que cada pixel esta representado por un nivel de “rojo”, un nivel de “verde” y un nivel de “azul”. A dicha imágen se le aplico una transformación para pasarla a modo escala de grises

(Figura 3.3), cada pixel esta representado por un valor de brillo que va desde el 0 (negro) hasta el 255 (blanco), con el objetivo de reducir el tamaño de la imágen original.

Para lograr dicha transformación se hizo uso de la función cvCvtColor() de la biblioteca OpenCV, la cual toma los valores en modo RGB de cada pixel y aplica la siguiente formula:

$$Y = 0.299*R + 0.587*G + 0.114*B$$

Obteniendo el valor correspondiente en modo escala de grises. La sintaxis de la función cvCvtColor es la siguiente:

void cvCvtColor(CvArr* src, CvArr* dst, int code)

- **src** - Imágen de entrada.
- **dst** - Imágen de salida.
- **code** - Define la operación de conversión a llevar a cabo.

OpenCV cuenta con códigos de conversión predefinidos, en este caso se utilizo del código CV_RGB2GRAY. Los códigos se pueden consultar en el manual de la función[13].

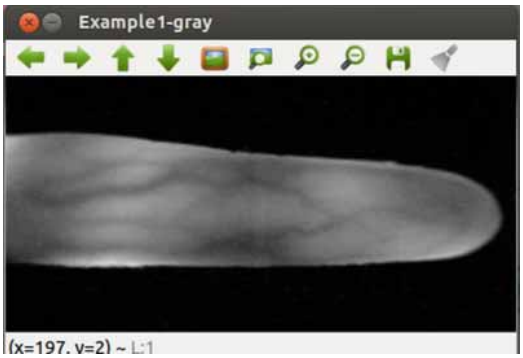


Figura 3.3: Imágen en escala de grises.

Filtro de mediana

A la imágen en escala de grises se le aplica un filtro de mediana para suavizar o eliminar el ruido existente en ella. La función principal del filtro de mediana es forzar a los puntos con valores de intesidad muy distintos a sus vecinos a tener valores más próximos a sus vecinos, de modo que se eliminan los picos de intensidad que aparecen en áreas aisladas. La Figura 3.4 muestra la imágen despues de aplicar el filtro de mediana.

Para realizar este filtro se utilizo la función cvSmooth() de OpenCV, a continuación se describe su sintaxis.

void cvSmooth(CvArr* src, CvArr* dst, int smoothtype, int size1, int size2)

- **src** - Imágen fuente.
- **dst** - Imágen destino.
- **smoothtype** - Tipo de filtro.
  - CV_BLUR_NO_SCALE
  - CV_BLUR
  - CV_GAUSSIAN
  - CV_MEDIAN
  - CV_BILATERAL
- **size1** - Dimensión de anchura de la submatriz.
- **size2** - Dimensión de altura de la submatriz. Ignorado por los métodos CV_MEDIAN y CV_BILATERAL. En caso de no ser ignorado y ser igual a 0 se iguala a size1.

En nuestro caso en particular, el valor del parametro "tipo de filtro" que utilizamos fue CV_MEDIAN. La función sustituye el pixel del centro de la submatriz por el del valor de la mediana de todos los pixeles de la submatriz. Para ello se utiliza una submatriz size1 x size2, donde size1 debe ser igual a size2 y además debe ser impar para tener claro cuál es el pixel central. Es importante elegir bien el tamaño de la submatriz, ya que un valor para size1 pequeño implica que no se elimine bien el ruido y un valor alto implica la distorsión de la imagen. El valor que se utilizo para size1 y size2 fue igual a 3.

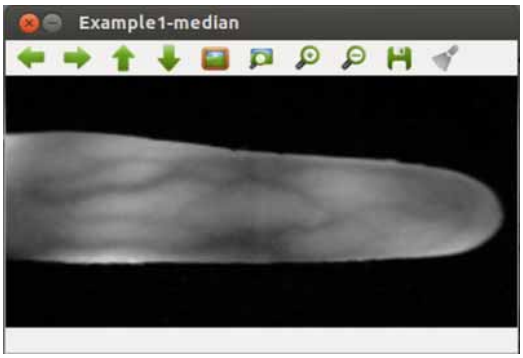


Figura 3.4: Filtro de mediana a escala de grises.

Segmentación de imágen

Para separar la región del dedo del fondo de la imágen, se realizo un proceso de segmentación el cual involucra 3 pasos; detección de bordes, suavizado de bordes y rellenar la región del dedo.

Para la detección de bordes se utilizo la función cvCanny() de la biblioteca OpenCV, dicha función lleva el nombre del australiano John F. Canny quien desarrollo el algoritmo de detección de bordes en 1986.

El algoritmo de Canny hace uso de la primera derivada, ya que toma el valor de cero en todas las regiones donde no varía la intensidad y tiene un valor constante en toda la transición de intensidad. Por tanto un cambio de intensidad se manifiesta como un cambio brusco en la primera derivada, característica que es usada para detectar un borde.

El algoritmo de Canny consiste en tres grandes pasos:

- Obtención del gradiente: en este paso se calcula la magnitud y orientación del vector gradiente en cada pixel.
- Supresión no máxima: en este paso se logra el adelgazamiento del ancho de los bordes, obtenidos con el gradiente, hasta lograr bordes de un pixel de ancho.
- Histéresis de umbral: en este paso se aplica una función de histéresis basada en dos umbrales; con este proceso se pretende reducir la posibilidad de aparición de contornos falsos.

A continuación se muestra la sintaxis de la función.

void cvCanny(CvArr* image, CvArr* edges, double threshold1, double threshold2, int aperture_size=3)

- **image** - Imágen de entrada.
- **edges** - Imágen de salida.
- **threshold1** - Primer valor de umbral para el procedimiento de histéresis.
- **threshold2** - Segundo valor de umbral para el procedimiento de histéresis
- **aperture_size** - Tamaño de la submatriz.

Dada la tonalidad de grises que se presentan en la imágen del dedo, se eligieron los valores de 50 y 200 para threshold1 y threshold2 respectivamente, con el propósito de descartar los posibles máximos locales creados por ruido y al mismo tiempo sin eliminar bordes débiles.

El suavizado de bordes se obtuvo utilizando la función cvDilate() de la biblioteca OpenCV, con el objetivo de unir bordes que estuvieran interrumpidos. A continuación se muestra la sintaxis de la función.

void cvDilate(CvArr* src, CvArr* dst, IplConvKernel* element=NULL, int iterations=1)

- **src** - Imágen de entrada.
- **dst** - Imágen de salida.
- **element** - Elemento estructurante utilizado para la dilatación.
- **iterations** - Número de iteraciones que se aplica la dilatación.

Una vez que se obtuvieron los bordes bien definidios se procedio a rellenar la región del dedo con pixeles blancos, este procedimiento se realizo haciendo uso de la función cvFloodFill(), a continuación se muestra su sintaxis.

void cvFloodFill(CvArr* image, CvPoint seed_point, CvScalar new_val, CvScalar lo_diff=cvScalarAll(0), CvScalar up_diff=cvScalarAll(0), CvConnectedComp* comp=NULL, int flags=4, CvArr* mask=NULL)

- **image** - Imágen de entrada/salida.
- **seed_point** - Punto de partida.
- **new_val** - Nuevo valor de los pixeles.
- **lo_diff** - Diferencia inferior máxima de brillo/color entre el pixel observado actualmente y uno de sus vecinos pertenecientes al componente.
- **up_diff** - Diferencia superior máxima de brillo/color entre el pixel observado actualmente y uno de sus vecinos pertenecientes al componente.
- **flags** - Los bits inferiores contienen un valor de conectividad, 4 (por default) ó 8. La conectividad determina que vecinos de un pixel son considerados.

La Figura 3.5 muestra la imágen resultante del proceso de segmentación.

Alineación y cambio de tamaño

Despues de extraer la región del dedo se aplica una alineación, para minimizar la discrepancia en el proceso de verificación. Para realizar este paso se aplico una traslación vertical, con el fin de centrar la región del dedo dentro de toda la imágen. Al mismo tiempo se realizo una traslación horizontal, con el objetivo de alinear la región hacia el lado izquierdo de la imágen.

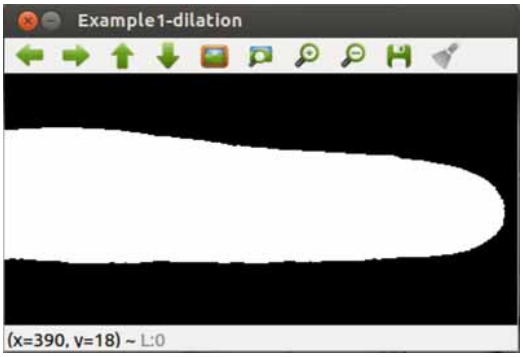


Figura 3.5: Imágen de la región del dedo.

Este procedimiento se implemeto utilizando la función cvWarpAffine() de la biblioteca OpenCV. A continuación se muestra su sintaxis.

void cvWarpAffine(CvArr* src, CvArr* dst, const CvMat* map_matrix, int flags=CV_INTER_LINEAR+CV_WARP_FILL_OUTLIERS, CvScalar fillval=cvScalarAll(0))

- **src** - Imágen fuente a la que se le aplica la transformación.
- **dst** - Imágen destino, donde se obtiene la transformación realizada.
- **map_matrix** - Matriz de transformación, de dimensiones 2X3.
- **flags** - tipos de interpolación.
  - CV_INTER_NN : Interpolación tipo vecino más próximo.
  - CV_INTER_LINEAR : Interpolación bilineal (es la que se utiliza por defecto).
  - CV_WARP_FILL_OUTLIERS : Rellena todos los pixeles de la imágen destino. Si algunos de ellos corresponden con partes que quedan fuera de la imagen origen, son rellenados a un valor fijo.
- **fillval** - Es el valor de relleno que se usa para definir los trozos que quedan fuera de la imágen.

Los tipos de interpolación que se utilizaron para llevar a cabo la traslación de la imágen fueron CV_INTER_LINEAR que se utiliza por defecto y CV_WARP_FILL_OUTLIERS para poder rellenar los pixeles que quedarán fuera de la imágen con el valor que se asigna al parametro fillval, 0 (color negro) en nuestro caso.

Además se reajusto el tamaño de la imágen, a 320 X 160 pixeles, para eliminar pixeles innecesarios permitiendo que el procesamiento sea rápido, para el cambio de tamaño se utilizo la función cvResize() de la biblioteca OpenCV. Su sintaxis es la

siguiente:

void cvResize(const CvArr* src, CvArr* dst, int interpolation=CV_INTER_LINEAR)

- **src** - Imágen de entrada.
- **dst** - Imágen de salida.
- **interpolation** - Método de interpolación.

La Figura 3.6 muestra el resultado del procedimiento de alineación y cambio de tamaño.

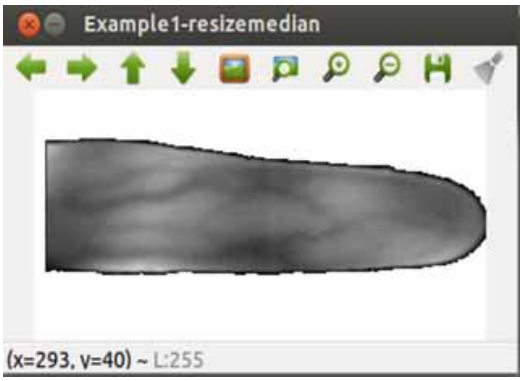


Figura 3.6: Imágen alineada y con cambio de tamaño.

Filtro gaussiano

Se aplica un filtro gaussiano para eliminar el ruido de alta frecuencia. El valor de cada punto es el resultado de promediar con distintos pesos los valores vecinos a ambos lados de dicho punto.

Para realizar este filtro se utilizo la función cvSmooth() de OpenCV, a continuación se describe su sintaxis.

void cvSmooth(CvArr* src, CvArr* dst, int smoothtype, int size1, int size2)

- **src** - Imágen fuente.
- **dst** - Imágen destino.
- **smoothtype** - Tipo de filtro.
  - CV_BLUR_NO_SCALE
  - CV_BLUR

  - CV_GAUSSIAN
  - CV_MEDIAN
  - CV_BILATERAL

- **size1** - Dimensión de anchura de la submatriz.
- **size2** - Dimensión de altura de la submatriz. Ignorado por los métodos CV_MEDIAN y CV_BILATERAL. En caso de no ser ignorado y ser igual a 0 se iguala a size1.

El valor del parametro "tipo de filtro" que utilizamos fue CV_GAUSSIAN, el valor que se asigno a size1 y size2 fue 7, este tamaño de submatriz permitio eliminar el ruido eficientemente. La Figura 3.7 muestra la imágen despues de aplicar el filtro gaussiano.

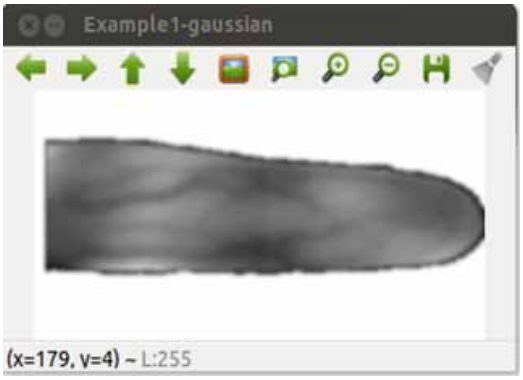


Figura 3.7: Aplicación de Filtro gaussiano.

Binarización

La binarización es una técnica para convertir la imágen de modo escala de grises a una representación de dos niveles que son el pixel negro con valor '0' y el pixel blanco con valor '255'. Esta técnica nos permite extraer el patrón de las venas del dedo de la imágen.

Para realizar esta operación se utilizo la función cvAdaptiveThreshold() de la biblioteca de OpenCV, la cual analiza cada pixel con respecto a su barrido local. A continuación se muestra su sintaxis.

```
void cvAdaptiveThreshold(CvArr* src, CvArr* dst, double max_value,
int adaptive_method=CV_ADAPTIVE_THRESH_MEAN_C,
int threshold_type=CV_THRESH_BINARY, int block_size=3, double param1=5 )
```

- **src** - Imágen fuente en escala de grises.
- **dst** - Imágen destino.
- **max_value** - Valor máximo a utilizar para los dos tipos de binarización más frecuentes.
- **adaptive_method** - Algoritmo de umbral adaptativo a utilizar.
  - ADAPTIVE_THRESH_MEAN_C
  - ADAPTIVE_THRESH_GAUSSIAN_C
- **threshold_type** - Tipo de binarización a utilizar.
  - THRESH_BINARY : dst(x,y) = max_value si src(x,y)>threshold, 0 en caso contrario.
  - THRESH_BINARY_INV : dst(x,y) = 0 si src(x,y)>threshold, max_value en caso contrario.
- **block_size** - Tamaño del conjunto de pixeles que se utiliza para calcular un valor de umbral.

La Figura 3.8 muestra el resultado de la imágen binarizada.

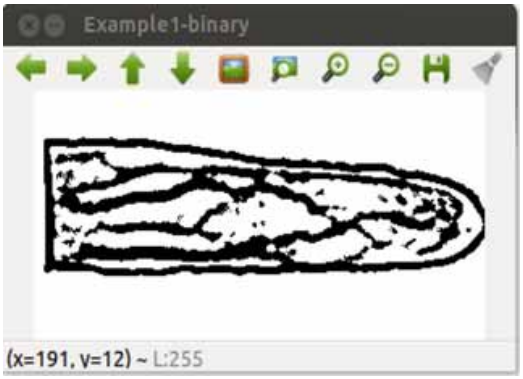


Figura 3.8: Imágen binarizada.

Filtro de mediana

Una vez que se obtuvo la imágen binarizada es necesario aplicarle un filtro de mediana para suavizar o eliminar el ruido existente en ella. La Figura 3.9 muestra la imágen despues de aplicar dicho filtro.

Para realizar este filtro se utilizo la función cvSmooth() de OpenCV, a continuación se describe su sintaxis.

void cvSmooth(CvArr* src, CvArr* dst, int smoothtype, int size1, int size2)

- **src** - Imágen fuente.
- **dst** - Imágen destino.
- **smoothtype** - Tipo de filtro.
  - CV_BLUR_NO_SCALE
  - CV_BLUR
  - CV_GAUSSIAN
  - CV_MEDIAN
  - CV_BILATERAL
- **size1** - Dimensión de anchura de la submatriz.
- **size2** - Dimensión de altura de la submatriz. Ignorado por los métodos CV_MEDIAN y CV_BILATERAL. En caso de no ser ignorado y ser igual a 0 se iguala a size1.

El valor del parámetro "tipo de filtro" que utilizamos fue CV_MEDIAN y el tamaño de la submatriz que se utilizo para calcular la mediana fue de 5X5, permitiendo con ello eliminar la mayor cantidad de ruido, sin llegar a distorsionar la imágen.

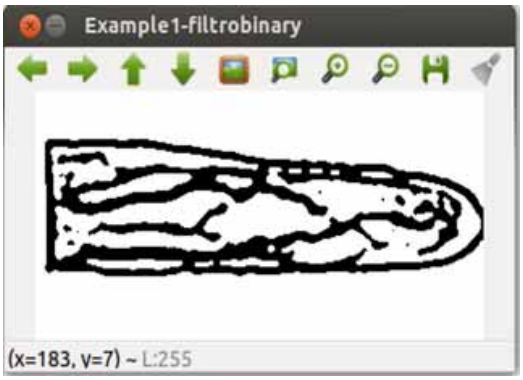


Figura 3.9: Filtro de mediana a imágen binaria.

Adelgazamiento

El último paso para poder obtener el patrón de las venas bien definido es aplicar un algoritmo de adelgazamiento, el cual consite en remover pixeles, dejando solo una linea bien definida de solo un pixel de ancho.

El algoritmo que se utilizo fue desarrollado por T. Y. Zhang y C. Y. Suen y es llamado "Algoritmo paralelo rápido para adelgazamiento"[8]. El cual se divide en dos etapas. Para llevar a cabo estas dos etapas es necesario definir a la imágen en una matriz de tamaño 3 X 3, donde cada pixel tiene un valor de 0 ó 1. El algoritmo procesa punto a punto y determina el valor que toma cada pixel, tomando en cuenta a sus 8 vecinos como se muestra en la Figura 3.10.

P9 (i-1, j-1)	P2 (i-1, j)	P3 (i-1, j+1)
P8 (i, j-1)	P1 (i, j)	P4 (i, j+1)
P7 (i+1, j-1)	P6 (i+1, j)	P5 (i+1, j+1)

Figura 3.10: Definición de la matriz utilizada por el algoritmo.

En la primer etapa el punto P1 es eliminado del patrón si se satisfacen las siguientes condiciones.

1. 2 <= B(P1) <= 6
2. A(P1) = 1
3. P2*P4*P6 = 0
4. P4*P6*P8 = 0

Donde:

A(P1) es el número de veces que aparece el patrón "01" durante un recorrido a los 8 vecinos del pixel que se procesa actualmente. El recorrido debe seguir el orden P2, P3, P4, ..., P8, P9.

B(P1) es el número de vecinos que tienen un valor diferente de 0, es decir B(P1) = P2 + P3 + P4 + ...+ P8 + P9

Si alguna de estas condiciones no se satisface, el punto P1 se conserva en el patrón.

En la segunda etapa, las condiciones 3 y 4 se cambian por las siguientes:

- P2*P4*P8 = 0
- P2*P6*P8 = 0

dejando las otras dos condiciones iguales. La implementación de este algoritmo se puede apreciar en el archivo adelgazamiento.c dentro del Apéndice B. La Figura 3.11 muestra el patrón de las venas, una vez que se aplico el proceso de adelgazamiento.

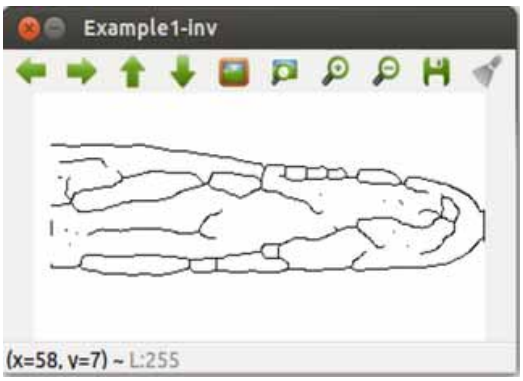


Figura 3.11: Adelgazamieto de las venas.

3.2.3. Extracción de características

Una vez obtenido el esqueleto de las venas, se aplica un algoritmo de extracción de minucias con el propósito de generar la plantilla, que será almacenada en la base de datos durante la fase de registro de los usuarios, y que posteriormente se utiliza para realizar la comparación con las características extraídas durante la fase de verificación.

El algoritmo que se utilizo para implementar este proceso se llama "Cross Number (CN)"[9]. El cual esta definido como el número de transiciones del valor 0 al valor 1 y viceversa, durante el recorrido de los 8 vecinos del pixel que se este procesando, en una matriz de tamaño 3 X 3 como se muestra en la Figura 3.12.

Los pixeles de nuestro interes son aquellos conocidos como bifurcaciones, el algoritmo CN permite hallar estos puntos bajo la condición de que el numero de transiciones entre los valores 0 y 1 sea igual o mayor que 6. Una vez que se encuentran estos puntos, se genera el vector característico para el patrón de la venas, que será almacenado en la base de datos, almacenando en él las coordenadas de los puntos y el número de minucias que se encontraron.

La implemetación del algoritmo CN se puede observar en el archivo registraTemplate.c dentro del Apéndice B. La Figura 3.13 muestra el resultado que arroja el algoritmo de extracción de minucias.

P8	P1	P2
P7	P0	P3
P6	P5	P4

Figura 3.12: Matriz utilizada por el algoritmo CN.

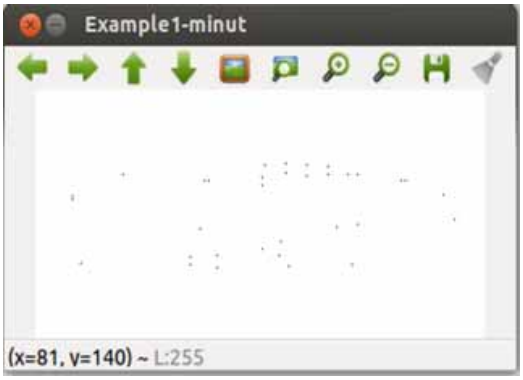


Figura 3.13: Minucias extraídas del patrón de las venas.

3.2.4. Verificación

Por último, es en el módulo de verificación donde se determina la autentificación de las personas (para fines de pruebas, en este proyecto tres individuos), esto se lleva a cabo utilizando el algoritmo de distancia de Hausdorff, el cual calcula la disimilitud entre dos imágenes.

Dados dos conjuntos de puntos A = $\{a_1, ..., a_m\}$ y B = $\{b_1, ..., b_n\}$ la distancia de Hausdorff es definida como:

$$H(A,B) = \min(h(A,B), h(B,A))$$

donde:

$$h(A,B) = \min \| a - b \|$$

La función h(A,B) es conocida como distancia directa de Hausdorff de A a B. Esta función mide la distancia de un punto en A a su vecino mas cercano en B. Así la distancia de Hausdorff H(A,B) mide el grado de desigualdad entre dos conjuntos de puntos. El valor mas pequeño de H(A,B) indica la mejor similitud entre los dos conjuntos.

La implementación de este algoritmo se puede observar en el archivo distancias.c dentro del Apéndice B.

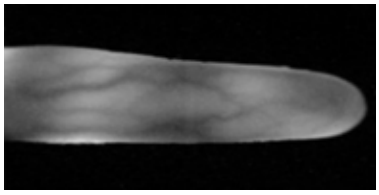
Capítulo 4

Pruebas y Resultados

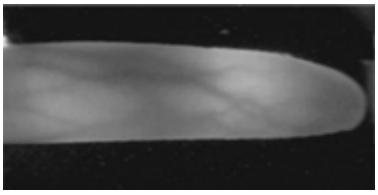
Para realizar las pruebas del sistema biométrico es necesario tener instaladas las bibliotecas de OpenCV. El proceso para instalar estas bibliotecas se describe a detalle en el Apéndice A. Además es necesario compilar el código fuente con ayuda del archivo makefile mostrado en el Apéndice B, el cuál debe de estar ubicado en el directorio donde se encuentre el código fuente. Para realizar la compilación del código fuente es necesario ubicarse en el directorio que almacena los archivos fuente y ejecutar el siguiente comando:

```
$ make
```

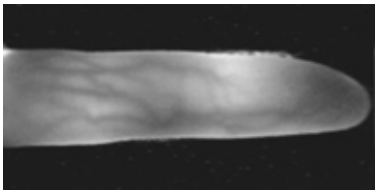
Las pruebas se realizaron con tres imágenes de dedos diferentes (Figura 4.1), con los nombres vein1.jpg, vein2.jpg y vein3.jpg respectivamente.



(a) vein1.jpg



(b) vein2.jpg



(c) vein3.jpg

Figura 4.1: Imágenes utilizadas para las pruebas

Para poder ejecutar el sistema biométrico es necesario teclear el siguiente comando en una terminal:

```
$ ./biometria [source]
```

Donde [source] corresponde a la imágen que se quiera procesar.

En la Figura 4.2 se puede apreciar la ejecución del sistema biométrico, pasandole como argumento la imagen vein1.jpg. El sistema nos despliega el menú, con las opciones: Registrar plantilla y Proceso de verificación.

La primer etapa de pruebas fue el registro de plantillas de cada una de las imágenes, de esta manera la opción que seleccionamos del menú es la número uno. El sistema nos pide que asignemos el nombre con el que deseamos guardar la plantilla que se genera.

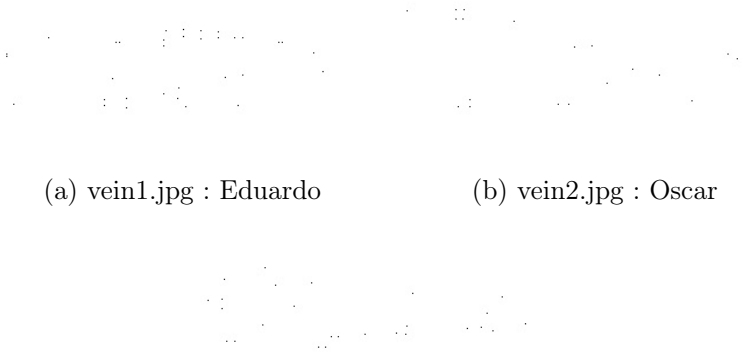
Los nombres que se asignaron, para las plantillas generadas al procesar cada una de las imágenes, fueron los siguientes :

- vein1.jpg : Eduardo
- vein2.jpg : Oscar
- vein3.jpg : Angel

Figura 4.2: Ejecución del sistema biométrico.

En la Figura 4.3 se pueden apreciar las minucias que el sistema biométrico extrajo en cada una de las imágenes procesadas.

Las plantillas que el sistema genero, para cada una de las imágenes, se guardaron dentro de un directorio nombrado “Plantillas”. En la Figura 4.4 se pueden apreciar los archivos que contienen la información de cada una de estas plantillas, los cuales estan en formato de texto plano.



(a) vein1.jpg : Eduardo (b) vein2.jpg : Oscar

(c) vein3.jpg : Angel

Figura 4.3: Minucias extraídas.

.

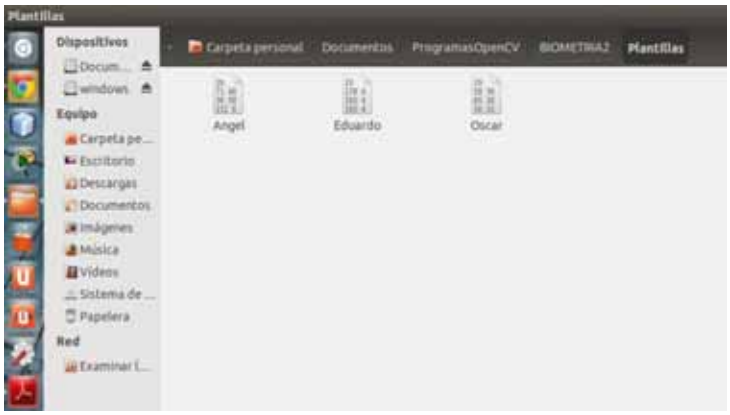


Figura 4.4: Archivos de plantillas generadas.

Una vez que se habian registrado las plantillas de las imágenes, la siguiente etapa de pruebas del sistema biométrico consitio en la fase de verificación. En la cual se debia determinar, la identidad de la persona según su patrón vascular de las venas del dedo.

La primer prueba que se realizo fue con la imagen nombrada “vein3.jpg”, la cual si recordamos, en la etapa de registro se le asigno el nombre de Angel a la plantilla generada en base a esta imágen. En la Figura 4.5 se puede observar la ejecución del sistema biométrico pasandole como argumento dicha imágen.

Figura 4.5: Proceso de verificación para la imágen vein3.jpg

En primer lugar el sistema biométrico nos arroja la distancia de disimilitud, que existe al comparar la imágen que se pasa como argumento respecto a las plantillas que se tienen almacenadas en la base de datos. Dando los siguientes resultados :

- 20 respecto a la plantilla correspondiente a Oscar.
- 0 respecto a la plantilla correspondiente a Angel.
- 15 respecto a la plantilla correspondiente a Eduardo.

Y tomando en cuenta estos valores, el sistema determina que la identidad de la persona, a quien corresponde el dedo que se esta procesando es “Angel”, dado que existe una disimilitud de valor 0. En otras palabras, el patrón vascular del dedo que se esta procesando es exactamente igual a la plantilla que se tiene almacenada en la base de datos con dicho nombre.

La segunda imágen con la cual se realizo el proceso de verificación fue con la imágen vein2.jpg. En la Figura 4.6 se pueden observar los resultados que arroja el sistema biométrico al procesar dicha imágen.
Dados los siguientes valores de disimilitud:

- 0 respecto a la plantilla correspondiente a Oscar.
- 17 respecto a la plantilla correspondiente a Angel.
- 18 respecto a la plantilla correspondiente a Eduardo.

El sistema determina que el patrón vascular, correspondiente a la imágen del dedo que se esta procesando, pertenece a “Oscar”.

Por último, al realizar el proceso de verificación para la imágen vein1.jpg (Figura 4.7). Los valores de disimilitud fueron los siguientes :

Figura 4.6: Proceso de verificación para la imágen vein2.jpg

- 20 respecto a la plantilla correspondiente a Oscar.
- 17 respecto a la plantilla correspondiente a Angel.
- 0 respecto a la plantilla correspondiente a Eduardo.

Figura 4.7: Proceso de verificación para la imágen vein1.jpg

Determinando que el patrón vascular del dedo que se esta procesando, pertenece a “Eduardo”.

Los resultados obtenidos con el sistema biométrico fueron satisfactorios, al haber identificado de manera correcta los patrones vasculares de las tres imágenes con las que se realizaron las pruebas.

Capítulo 5

Conclusiones

Con el desarrollo del sistema biométrico se obtuvieron importantes resultados durante la etapa de pruebas, dejandonos ver que el método de autentificación personal basado en las venas del dedo tiene un alto grado de confiabilidad, pudiendo ser uno de los métodos mas confiables en cuestión de seguridad de la actualidad.

La implementación del sistema fue un gran reto, debido a que se tuvo que estudiar y comprender el funcionamiento de las bibliotecas OpenCV, además de entender el funcionamiento de los algoritmos utilizados, para poder implementarlos correctamente.

Los objetivos a los que se pretendia llegar se cumplieron satisfactoriamente, obteniendo un sistema biométrico capaz de realizar una autentificación personal confiable basada en las venas del dedo, que erradica las desventajas que se tienen con los métodos tradicionales, como el uso de contraseñas o tarjetas inteligentes.

El presente trabajo puede ser el comienzo de un potente sistema de autentificación, ya que podría adaptarse para que la identificación personal se base, no solo en el patrón de venas de un dedo si no, en el patrón de las venas de los cinco dedos de la mano, incrementando con esto el grado de robustez y confiabilidad.

Apéndice A

Instalación de la biblioteca OpenCV

A.1. Actualiza los repositorios

Abre el terminal y ejecuta los siguientes comandos:

```
$ sudo apt-get update
```

```
$ sudo apt-get upgrade
```

A.2. Instala las dependencias

El siguiente paso es instalar las dependencias de OpenCV. Para ello, ejecuta el siguiente comando en tu terminal:

```
$ sudo apt-get install build-essential libgtk2.0-dev libjpeg-dev libtiff4-
    dev libjasper-dev libopenexr-dev cmake python-dev python-numpy python-
    tk libtbb-dev libeigen2-dev yasm libfaac-dev libopencore-amrnb-dev
    libopencore-amrwb-dev libtheora-dev libvorbis-dev libxvidcore-dev
    libx264-dev libqt4-dev libqt4-opengl-dev sphinx-common texlive-latex-
    extra libv4l-dev libdc1394-22-dev libavcodec-dev libavformat-dev
    libswscale-dev
```

A.3. Descarga y descomprime OpenCV

Entra a la página web oficial de descargas de OpenCV, descargas OpenCV 2.4.5 for Linux/Mac (esta fue la versión que se utilizo para desarrollar este proyecto), lo descomprimes y entra al directorio creado desde la terminal.

```
$ tar -xvf OpenCV-2.4.5.tar.bz2
```

```
$ cd OpenCV-2.4.5
```

A.4. Compila e instala OpenCV

Asegurate de estar dentro del directorio de OpenCV y ejecuta los siguientes comandos:

```
$ mkdir build
```

```
$ cd build
```

```
$ cmake -D WITH_TBB=ON -D BUILD_NEW_PYTHON_SUPPORT=ON -D WITH_V4L=ON -D
    INSTALL_C_EXAMPLES=ON -D INSTALL_PYTHON_EXAMPLES=ON -D BUILD_EXAMPLES=
    ON -D WITH_QT=ON -D WITH_OPENGL=ON ..
```

```
$ make
```

```
$ sudo make install
```

A.5. Prepara OpenCV

Ahora edita el archivo de configuración de OpenCV con el comando:

```
$ sudo gedit /etc/ld.so.conf.d/opencv.conf
```

Seguramente el archivo esté vacío, escribe /usr/local/lib en él, guárdalo y sal del editor.

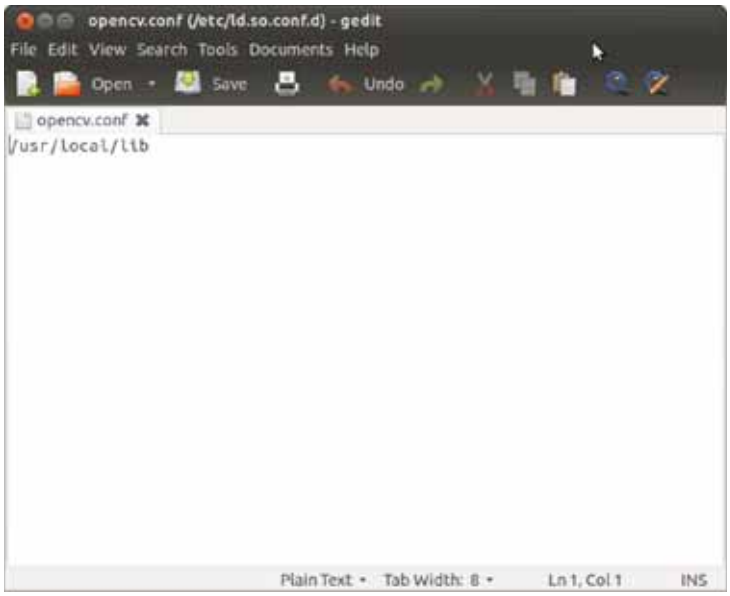


Figura A.1: Archivo de configuración opencv.conf

A continuación ejecuta el siguiente comando en la terminal:

```
$ sudo ldconfig
```

Ahora añade el PATH editando el arranque de Bash. Para ello ejecuta:

```
$ sudo gedit /etc/bash.bashrc
```

Al final del archivo, añade:

.

```
PKG_CONFIG_PATH=$PKG_CONFIG_PATH:/usr/local/lib/pkgconfig
export PKG_CONFIG_PATH
```

.

Guarda el archivo y sal del editor, por último, reinicia la computadora.

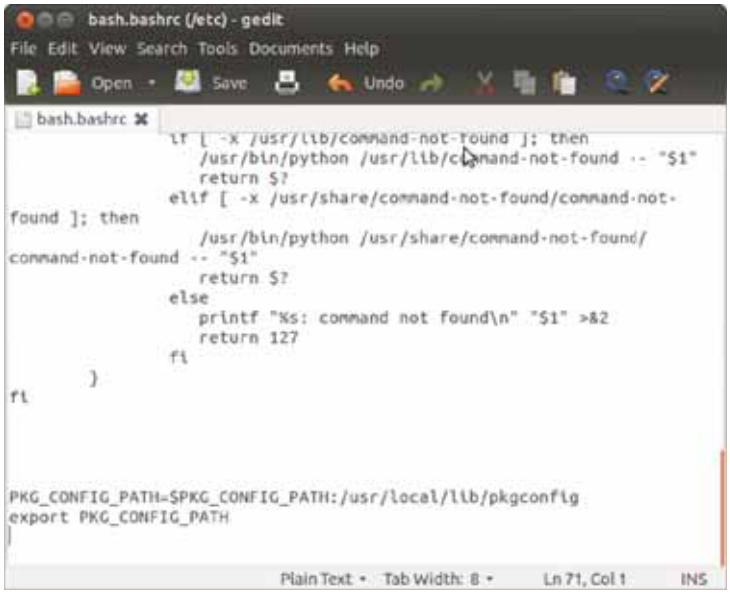


Figura A.2: Archivo de configuración bash.bashrc

A.6. Prueba un ejemplo

Para probar que la instalación fue exitosa, ejecuta los siguientes comandos:

```
$ cd ~/OpenCV-2.4.5/samples/c
```

```
$ chmod +x build_all.sh
```

```
$ ./build_all.sh
```

```
$ ./facedetec -cascade="/usr/local/share/OpenCV/haarcascades/
    haarcascade_frontalface_alt.xml" -scale=1.5 lena.jpg
```

Si todo esta correcto se verá a la famosa Lena con una circunferencia azul rodeando su cara.

Apéndice B

Código fuente del sistema biométrico

Archivo biometria.c

```
#include <stdio.h>
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/imgproc/imgproc_c.h"
#include "registraTemplate.h"
#include "adelgazamiento.h"
#include "verificacion.h"

int main(int argc, char** argv)
{
    int op;
    printf("1. Registrar plantilla\n");
    printf("2. Proceso de verificacion\n");
    printf("Seleccione una opcion : ");
    scanf("%d", &op);

    int i, j, aux;
    int x=0;
    int y=0;
    IplImage* img = cvLoadImage(argv[1], CV_LOAD_IMAGE_COLOR); //Carga
        una imagen en la variable img

    cvNamedWindow("Example1", CV_WINDOW_AUTOSIZE); //Crea una ventana
    cvNamedWindow("Example1-gray", CV_WINDOW_AUTOSIZE);
    cvNamedWindow("Example1-median", CV_WINDOW_AUTOSIZE);
    cvNamedWindow("Example1-edges", CV_WINDOW_AUTOSIZE);
    cvNamedWindow("Example1-dilation", CV_WINDOW_AUTOSIZE);
    cvNamedWindow("Example1-resizemedian", CV_WINDOW_AUTOSIZE);
    cvNamedWindow("Example1-resizemd", CV_WINDOW_AUTOSIZE);
    cvNamedWindow("Example1-gaussian", CV_WINDOW_AUTOSIZE);
    cvNamedWindow("Example1-binary", CV_WINDOW_AUTOSIZE);
    cvNamedWindow("Example1-filtrobinary", CV_WINDOW_AUTOSIZE);
    cvNamedWindow("Example1-inv", CV_WINDOW_AUTOSIZE);
    cvNamedWindow("Example1-minut", CV_WINDOW_AUTOSIZE);


```

```
36      //Conversion de la imagen original a escala de grises.
37      IplImage* gray = cvCreateImage(cvGetSize(img), IPL_DEPTH_8U, 1);
38      cvCvtColor(img, gray, CV_BGR2GRAY);
39
40      //Aplicacion de filtro de mediana
41      IplImage* median = cvCreateImage(cvGetSize(img), IPL_DEPTH_8U, 1);
42      cvSmooth(gray, median, CV_MEDIAN, 3, 3, 0, 0);
43
44
45      //Deteccion de bordes
46      IplImage* edge = cvCreateImage(cvGetSize(gray), IPL_DEPTH_8U, 1);
47      cvCanny(gray, edge, 50, 200, 3);
48
49      //Morphological dilation
50      IplImage* md = cvCreateImage(cvGetSize(edge), IPL_DEPTH_8U, 1);
51      cvDilate(edge, md, NULL, 1);
52
53      cvFloodFill(md, cvPoint(100,100), cvScalarAll(255), cvScalarAll(0),
            cvScalarAll(0), NULL, 4, NULL );
54
55      CvScalar s;
56
57      for(i=5; i < md->height; i++)
58      {
59         for(j=0; j < md->width - 5; j++)
60         {
61            s = cvGet2D(md, i, j); //Obtiene el valor del pixel i, j
62            aux = s.val[0];
63            if(aux == 255)
64            {
65         if(j > x)
66                {
67                   y = i;
68                   x = j;
69                }
70            }
71         }
72      }
73
74      printf("x = %d\n", x);
75      printf("y = %d\n", y);
76
77      printf("height = %d\n", i);
78      printf("width = %d\n", j);
79
80      int dy = (md->height/2) - y;
81      int dx = md->width - x;
82
83      printf("dy = %d\n", dy);
84
85      CvMat* M = cvCreateMat(2, 3, CV_32FC1);
86      cvmSet(M, 0, 0, 1);
87      cvmSet(M, 0, 1, 0);
```

```
88      cvmSet(M, 1, 0, 0);
89      cvmSet(M, 1, 1, 1);
90      cvmSet(M, 0, 2, dx);
91      cvmSet(M, 1, 2, dy);
92
93
94      //Sustituimos por la imagen en escala de grises.
95      //Alineacion de la imagen.
96      cvWarpAffine(md, md, M, CV_INTER_LINEAR+CV_WARP_FILL_OUTLIERS,
            cvScalarAll(0));
97      cvWarpAffine(median, median, M, CV_INTER_LINEAR+
            CV_WARP_FILL_OUTLIERS, cvScalarAll(0)); //Traslacion
98
99      //Resize
100     IplImage* resmedian = cvCreateImage(cvSize(320, 160), median->depth,
            median->nChannels);
101     cvResize(median, resmedian, CV_INTER_LINEAR);
102
103     IplImage* resmd = cvCreateImage(cvSize(320, 160), md->depth, md->
            nChannels);
104     cvResize(md, resmd, CV_INTER_LINEAR);
105
106     //Extraccion del dedo con fondo blanco
107
108     CvScalar a;
109     a.val[0] =255;
110
111     for(i=0; i < resmd->height; i++)
112     {
113        for(j=0; j < resmd->width; j++)
114        {
115           s = cvGet2D(resmd, i, j); //Obtiene el valor del pixel i, j
116           aux = s.val[0];
117           if(aux == 0)
118           {
119        cvSet2D(resmedian, i, j, a);
120           }
121        }
122     }
123
124     //FILTRO GAUSSIAN
125     IplImage* gaussian = cvCreateImage(cvGetSize(resmedian),
            IPL_DEPTH_8U, 1);
126     cvSmooth(resmedian, gaussian, CV_GAUSSIAN, 7, 7, 0, 0);
127
128     //BINARIZACION
129     IplImage* binary = cvCreateImage(cvGetSize(resmedian), IPL_DEPTH_8U,
            1);
130     cvAdaptiveThreshold(gaussian, binary, 255, CV_ADAPTIVE_THRESH_MEAN_C
            , CV_THRESH_BINARY, 11, 1);
131
132     //FILTRO DE MEDIANA BINARIA
133     IplImage* filtrobinary = cvCreateImage(cvGetSize(binary),
```

```
        IPL_DEPTH_8U, 1);
134     cvSmooth(binary, filtrobinary, CV_MEDIAN, 5, 5, 0, 0);
135
136     //************************ THINNING
            **********************************//
137     IplImage* inv = cvCreateImage(cvGetSize(filtrobinary), IPL_DEPTH_8U,
            1);
138
139     CvScalar uno, z;
140     uno.val[0] = 1;
141     CvScalar cero;
142     cero.val[0] = 0;
143
144     for(i = 0; i < filtrobinary->height; i++){
145        for(j = 0; j < filtrobinary->width; j++){
146           z = cvGet2D(filtrobinary, i, j);
147           aux = z.val[0];
148           if( aux == 0){
149              cvSet2D(inv, i, j, uno);
150           }
151           else{
152              cvSet2D(inv, i, j, cero);
153           }
154        }
155     }
156
157     CvScalar o;
158     o.val[0] = 255;
159
160     IplImage* ceros = cvCreateImage(cvGetSize(inv), IPL_DEPTH_8U, 1);
161     cvSetZero(ceros); //Llenamos la matriz con ceros
162
163     IplImage* diferencia = cvCreateImage(cvGetSize(filtrobinary),
           IPL_DEPTH_8U, 1);
164
165     do
166     {
167        thinning(inv, 0);
168        thinning(inv, 1);
169        thinning(inv, 2);
170        cvAbsDiff(inv, ceros, diferencia);
171        cvCopy(inv, ceros, NULL);
172     }while(cvCountNonZero(diferencia) > 0);
173
174     cvConvertScale(inv, inv, 255, 0); //Multiplicacion de la matriz por
           el valor 255
175
176     int k;
177
178     for(i = 0; i < inv->height; i++){
179        for(j = 0; j < inv->width; j++){
180           for(k=0; k<12; k++){
181              cvSet2D(inv, i, k, o);
```

```
182             }
183             z = cvGet2D(inv, i, j);
184             aux = z.val[0];
185             if( aux == 0){
186                 cvSet2D(inv, i, j, o);
187             }
188             else{
189                 cvSet2D(inv, i, j, cero);
190             }
191         }
192     }
193
194     //********************* Extraccion de minucias
            **********************//
195
196     IplImage* thin = cvCreateImage(cvGetSize(inv), IPL_DEPTH_8U, 1);
197
198
199     for(i = 0; i < inv->height; i++){
200         for(j = 0; j < inv->width; j++){
201             z = cvGet2D(inv, i, j);
202             aux = z.val[0];
203             if( aux == 0){
204                 cvSet2D(thin, i, j, uno);
205             }
206             else{
207                 cvSet2D(thin, i, j, cero);
208             }
209         }
210     }
211
212
213
214     if(op == 1)
215         extractMinutiae(thin);
216     else
217         match(thin);
218
219
220         cvConvertScale(thin, thin, 255, 0); //Multiplicacion de la matriz
                por el valor 255
221
222     //*************************************************************//
223     cvSaveImage("Patron.jpg", filtrobinary, 0);
224
225     cvShowImage("Example1", img);     // Muestra la imagen en la ventana
226     cvShowImage("Example1-gray", gray);
227     cvShowImage("Example1-median", median);
228     cvShowImage("Example1-edges", edge);
229     cvShowImage("Example1-dilation", md);
230     cvShowImage("Example1-resizemedian", resmedian);
231     cvShowImage("Example1-resizemd", resmd);
232     cvShowImage("Example1-gaussian", gaussian);
```

```
233     cvShowImage("Example1-binary", binary);
234     cvShowImage("Example1-filtrobinary", filtrobinary);
235     cvShowImage("Example1-inv", inv);
236     cvShowImage("Example1-minut", thin);
237 
238     cvWaitKey(0);    // Espera una tecla
239 
240     cvReleaseImage(&img);
241     cvReleaseImage(&gray);
242     cvReleaseImage(&median);
243     cvReleaseImage(&edge);
244     cvReleaseImage(&md);
245     cvReleaseImage(&resmedian);
246     cvReleaseImage(&resmd);
247     cvReleaseImage(&gaussian);
248     cvReleaseImage(&binary);
249     cvReleaseImage(&filtrobinary);
250     cvReleaseImage(&inv);
251     cvReleaseImage(&thin);
252 
253 
254     cvDestroyWindow("Example1");
255     cvDestroyWindow("Example1-gray");
256     cvDestroyWindow("Example1-median");
257     cvDestroyWindow("Example1-edges");
258     cvDestroyWindow("Example1-dilation");
259     cvDestroyWindow("Example1-resizemedian");
260     cvDestroyWindow("Example1-resizemd");
261     cvDestroyWindow("Example1-gaussian");
262     cvDestroyWindow("Example1-binary");
263     cvDestroyWindow("Example1-filtrobinary");
264     cvDestroyWindow("Example1-inv");
265     cvDestroyWindow("Example1-minut");
266 
267     return 0;
268 }
```

Archivo adelgazamiento.c

```
1  #include<stdio.h>
2  #include<string.h>
3  #include "opencv2/highgui/highgui.hpp"
4  #include "opencv2/imgproc/imgproc.hpp"
5  #include "opencv2/imgproc/imgproc_c.h"
6 
7  void thinning(IplImage* binary, int iter)
8  {
9 
10     int i, j, p2, p3, p4, p5, p6, p7, p8, p9, A, B, m1, m2, aux;
11     CvScalar a, b, pa2, pa3, pa4, pa5, pa6, pa7, pa8, pa9;
12     a.val[0] = 1;
13 
14     IplImage* marker = cvCreateImage(cvGetSize(binary), IPL_DEPTH_8U, 1)
           ;
```

```
15      cvSetZero(marker); //Llenamos la matriz con ceros
16
17
18      for(i = 1; i < binary->height-1; i++)
19      {
20         for(j = 1; j < binary->width-1; j++)
21         {
22            pa2 = cvGet2D(binary, i-1, j);
23            pa3 = cvGet2D(binary, i-1, j+1);
24            pa4 = cvGet2D(binary, i, j+1);
25            pa5 = cvGet2D(binary, i+1, j+1);
26            pa6 = cvGet2D(binary, i+1, j);
27            pa7 = cvGet2D(binary, i+1, j-1);
28            pa8 = cvGet2D(binary, i, j-1);
29            pa9 = cvGet2D(binary, i-1, j-1);
30
31            p2 = pa2.val[0];
32            p3 = pa3.val[0];
33            p4 = pa4.val[0];
34            p5 = pa5.val[0];
35            p6 = pa6.val[0];
36            p7 = pa7.val[0];
37            p8 = pa8.val[0];
38            p9 = pa9.val[0];
39
40            A = (p2 == 0 && p3 == 1) + (p3 == 0 && p4 == 1) +
41                     (p4 == 0 && p5 == 1) + (p5 == 0 && p6 == 1) +
42                     (p6 == 0 && p7 == 1) + (p7 == 0 && p8 == 1) +
43                     (p8 == 0 && p9 == 1) + (p9 == 0 && p2 == 1);
44
45            B = p2 + p3 + p4 + p5 + p6 + p7 + p8 + p9;
46
47            m1 = iter == 0 ? (p2 * p4 * p6) : (p2 * p4 * p8);
48            m2 = iter == 0 ? (p4 * p6 * p8) : (p2 * p6 * p8);
49
50      if(A == 1 && (B >= 2 && B <= 6) && m1 == 0 && m2 == 0)
51            {
52               cvSet2D(marker, i, j, a);
53            }
54         }
55      }
56
57      CvScalar uno;
58      uno.val[0] = 1;
59      CvScalar cero;
60      cero.val[0] = 0;
61
62      for(i = 0; i < marker->height; i++){
63         for(j = 0; j < marker->width; j++){
64            b = cvGet2D(marker, i, j);
65            aux = b.val[0];
66            if( aux == 0){
67               cvSet2D(marker, i, j, uno);
```

```
68              }
69              else{
70                  cvSet2D(marker, i, j, cero);
71              }
72          }
73      }
74
75      cvAnd(binary, marker, binary, NULL);
76  }
```

Archivo registraTemplate.c

```
1   #include<stdio.h>
2   #include<string.h>
3   #include "opencv2/highgui/highgui.hpp"
4   #include "opencv2/imgproc/imgproc.hpp"
5   #include "opencv2/imgproc/imgproc_c.h"
6
7   void extractMinutiae(IplImage* thin)
8   {
9       char* nombre;
10      char* ruta = "./Plantillas/";
11      char* nombrecompleto;
12      FILE* f;
13
14
15      printf("Introduzca el nombre para la plantilla : ");
16      scanf("%s", *(&nombre));
17      nombrecompleto = malloc(strlen(ruta) + strlen(nombre));
18      sprintf(nombrecompleto, "%s%s", ruta, nombre);
19      printf("nombre completo : %s\n", nombrecompleto);
20
21      if((f = fopen(nombrecompleto, "wb+")) == NULL){
22          printf("Error al crear el archivo");
23      }
24
25      int i, j, k=0, l=0, cn, p0, p1, p2, p3, p4, p5, p6, p7, p8, sp1, sp2
            , sp3, sp4, sp5, sp6, sp7, sp8, count=0;
26      CvScalar pa0, pa1, pa2, pa3, pa4, pa5, pa6, pa7, pa8, uno;
27      uno.val[0] = 1;
28
29      IplImage* m = cvCreateImage(cvGetSize(thin), IPL_DEPTH_8U, 1);
30      cvSetZero(m);
31
32      for(i = 1; i < thin->height-1; i=i+1)
33      {
34          for(j = 1; j < thin->width-1; j=j+1)
35          {
36              pa0 = cvGet2D(thin, i, j);
37              p0 = pa0.val[0];
38
39              if(p0 == 1)
40              {
41                  pa1 = cvGet2D(thin, i-1, j-1);
```

```
42        pa2 = cvGet2D(thin, i-1, j);
43              pa3 = cvGet2D(thin, i-1, j+1);
44              pa4 = cvGet2D(thin, i, j+1);
45              pa5 = cvGet2D(thin, i+1, j+1);
46              pa6 = cvGet2D(thin, i+1, j);
47              pa7 = cvGet2D(thin, i+1, j-1);
48              pa8 = cvGet2D(thin, i, j-1);
49
50              p1 = pa1.val[0];
51              p2 = pa2.val[0];
52              p3 = pa3.val[0];
53              p4 = pa4.val[0];
54              p5 = pa5.val[0];
55              p6 = pa6.val[0];
56              p7 = pa7.val[0];
57              p8 = pa8.val[0];
58
59        sp1 = abs(p2-p1);
60        sp2 = abs(p3-p2);
61        sp3 = abs(p4-p3);
62        sp4 = abs(p5-p4);
63        sp5 = abs(p6-p5);
64        sp6 = abs(p7-p6);
65        sp7 = abs(p8-p7);
66        sp8 = abs(p1-p8);
67
68              cn = sp1 + sp2 + sp3 + sp4 + sp5 + sp6 + sp7 + sp8;
69
70              if(cn >= 6)
71              {
72           count++;
73           cvSet2D(m, i, j, uno);
74              }
75           }
76        }
77     }
78
79     printf("contador %d\n", count);
80
81     //Arreglo para guardar las coordenadas de las minucias
82     int patron[count][2];
83
84     for(i = 0; i < m->height; i++)
85     {
86        for(j = 0; j < m->width; j++)
87        {
88     pa0 = cvGet2D(m, i, j);
89           p0 = pa0.val[0];
90           if( p0 == 1)
91           {
92              patron[k][l] = j;
93              l++;
94              patron[k][l] = i;
```

```
95              k++;
96              l--;
97          }
98       }
99    }
100
101   fprintf(f, "%d\n", count);
102
103   for(i=0; i<count; i++){
104      for(j=0; j<2; j++){
105         fprintf(f, "%d ", patron[i][j]);
106         if(j==1)
107            fprintf(f, "%s", "\n");
108      }
109   }
110
111   fclose(f);
112
113   cvAnd(thin, m, thin, NULL);
114 }
```

Archivo verificacion.c

```
1  #include<stdio.h>
2  #include<string.h>
3  #include "opencv2/highgui/highgui.hpp"
4  #include "opencv2/imgproc/imgproc.hpp"
5  #include "opencv2/imgproc/imgproc_c.h"
6  #include "directorio.h"
7
8  void match(IplImage* thin)
9  {
10
11    int i, j, k=0, l=0, cn, p0, p1, p2, p3, p4, p5, p6, p7, p8, sp1, sp2
          , sp3, sp4, sp5, sp6, sp7, sp8, count=0;
12    CvScalar pa0, pa1, pa2, pa3, pa4, pa5, pa6, pa7, pa8, uno;
13    uno.val[0] = 1;
14
15    IplImage* m = cvCreateImage(cvGetSize(thin), IPL_DEPTH_8U, 1);
16    cvSetZero(m);
17
18    for(i = 1; i < thin->height-1; i=i+1)
19    {
20       for(j = 1; j < thin->width-1; j=j+1)
21       {
22          pa0 = cvGet2D(thin, i, j);
23          p0 = pa0.val[0];
24
25          if(p0 == 1)
26          {
27             pa1 = cvGet2D(thin, i-1, j-1);
28       pa2 = cvGet2D(thin, i-1, j);
29             pa3 = cvGet2D(thin, i-1, j+1);
30             pa4 = cvGet2D(thin, i, j+1);
```

```
31              pa5 = cvGet2D(thin, i+1, j+1);
32              pa6 = cvGet2D(thin, i+1, j);
33              pa7 = cvGet2D(thin, i+1, j-1);
34              pa8 = cvGet2D(thin, i, j-1);
35
36              p1 = pa1.val[0];
37              p2 = pa2.val[0];
38              p3 = pa3.val[0];
39              p4 = pa4.val[0];
40              p5 = pa5.val[0];
41              p6 = pa6.val[0];
42              p7 = pa7.val[0];
43              p8 = pa8.val[0];
44
45        sp1 = abs(p2-p1);
46        sp2 = abs(p3-p2);
47        sp3 = abs(p4-p3);
48        sp4 = abs(p5-p4);
49        sp5 = abs(p6-p5);
50        sp6 = abs(p7-p6);
51        sp7 = abs(p8-p7);
52        sp8 = abs(p1-p8);
53
54              cn = sp1 + sp2 + sp3 + sp4 + sp5 + sp6 + sp7 + sp8;
55
56              if(cn >= 6)
57              {
58           count++;
59           cvSet2D(m, i, j, uno);
60              }
61           }
62        }
63     }
64
65     printf("contador %d\n", count);
66
67     //Arreglo para guardar las coordenadas de las minucias
68     int patron[count][2];
69
70     for(i = 0; i < m->height; i++)
71     {
72        for(j = 0; j < m->width; j++)
73        {
74     pa0 = cvGet2D(m, i, j);
75           p0 = pa0.val[0];
76           if( p0 == 1)
77           {
78              patron[k][l] = j;
79              l++;
80              patron[k][l] = i;
81              k++;
82              l--;
83           }
```

```
84          }
85      }
86
87      abrirDirectorio(patron, count);
88
89      cvAnd(thin, m, thin, NULL);
90  }
```

Archivo directorio.c

```
1   #include <stdlib.h>
2   #include <stdio.h>
3   #include <string.h>
4   #include <sys/types.h>
5   #include <dirent.h>
6   #include "archivos.h"
7
8
9   void abrirDirectorio(int patron[][2], int elementos)
10  {
11
12      DIR* dir; //Puntero para abrir el directorio
13      struct dirent* ent; //habra informacion del archivo que se esta
            sacando
14      char* ruta = "./Plantillas";
15
16      int count = 0;
17
18      dir = opendir(ruta);
19      if(dir == NULL)
20          printf("Error al abrir el directorio\n");
21
22      while((ent = readdir(dir)) != NULL)
23      {
24          if((strcmp(ent->d_name, ".") != 0) && (strcmp(ent->d_name, "..")
                != 0))
25          {
26              count++;//Guarda el numero de plantillas que existen dentro
                    del directorio.
27          }
28      }
29
30      char* plantillas[count]; //Arreglo para guardar los nombres de las
            plantillas.
31      int distancias[count]; //Arreglo para guardar las distancias del
            patron respecto a las plantillas.
32      count = 0;
33      closedir(dir);
34      dir = opendir(ruta);
35
36      while((ent = readdir(dir)) != NULL)
37      {
38          if((strcmp(ent->d_name, ".") != 0) && (strcmp(ent->d_name, "..")
                != 0))
```

```
39          {
40              distancias[count] = leerArchivo(patron, elementos, ruta, ent);
41              plantillas[count] = ent->d_name;
42              count++;
43          }
44      }
45
46      int k;
47      int menor = distancias[0];
48      char* nombre;
49
50      for(k=0; k<count; k++){
51          if(distancias[k] < menor)
52          {
53              menor = k;
54          }
55      }
56
57      for(k=0; k<count; k++){
58          printf("plantilla[%d] = %s\n", k, plantillas[k]);
59          printf("distancias[%d] = %d\n", k, distancias[k]);
60          if(menor == k)
61          {
62              nombre = plantillas[k];
63          }
64      }
65
66      printf("numero de plantillas = %d\n", count);
67      printf("El patron pertenece a : %s\n", nombre);
68      closedir(dir);
69  }
```

Archivo archivos.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <sys/types.h>
5  #include <dirent.h>
6  #include "distancias.h"
7
8  int leerArchivo(int patron[][2], int elementos, char* ruta, struct
       dirent* archivo)
9  {
10     int tmp, minucias, i, j, regresaVal;
11     char* nombrecompleto;
12
13     tmp = strlen(ruta); //determinamos el tamano de la cadena "ruta"
14     nombrecompleto = malloc(tmp + strlen(archivo->d_name) + 2);
15     if(ruta[tmp-1] == '/')
16         sprintf(nombrecompleto, "%s%s", ruta, archivo->d_name);
17     else
18         sprintf(nombrecompleto, "%s/%s", ruta, archivo->d_name);
19
```

```
20      printf("intentando abrir archivo : %s\n", nombrecompleto);
21
22      FILE* f;
23      if((f = fopen(nombrecompleto, "r")) == NULL)
24      {
25          printf("Error al abrir el archivo\n");
26      }
27      else{
28          printf("Se abrio el archivo : %s\n", archivo->d_name);
29          fscanf(f, "%d", &minucias);
30      }
31
32      int valores[minucias][2];
33
34      for(i=0; i<minucias; i++){
35          for(j=0; j<2; j++){
36              fscanf(f, "%d", &valores[i][j]);
37          }
38      }
39
40      //Valor de regreso.
41      regresaVal = calcularDistancia(patron, elementos, valores, minucias)
            ;
42
43      fclose(f);
44
45      return regresaVal;
46  }
```

Archivo distancias.c

```
1   #include <stdio.h>
2   #include <math.h>
3
4   int calcularDistancia(int matriz[][2], int elem1, int matriz2[][2], int
        elem2)
5   {
6
7       int mat[elem1][elem2];
8       int i, j;
9       int yA = 0;
10      int yB = 0;
11
12      printf("numero de elem1 : %d\n", elem1);
13      printf("numero de elem2 : %d\n", elem2);
14
15      for(i=0; i<elem1; i++)
16      {
17          for(j=0; j<elem2; j++)
18          {
19              mat[i][j] = sqrt( pow((matriz[i][yA] - matriz2[j][yB]),2) +
                    pow((matriz[i][yA+1] - matriz2[j][yB+1]),2) );
20      yA = 0;
21              yB = 0;
```

```
22          }
23      }
24
25      //Obtener los minimos de A - B
26      int minimos[elem1];
27      int menor;
28
29      for(i=0; i<elem1; i++)
30      {
31          menor = mat[i][0];
32          for(j=0; j<elem2; j++)
33          {
34              if(mat[i][j] < menor)
35              {
36                  menor = mat[i][j];
37      }
38              if(j == elem2-1)
39              {
40                  minimos[i] = menor;
41              }
42          }
43      }
44
45      //Calcular promedio de minimos
46      int suma=0;
47      int promedio;
48
49      for(i=0; i<elem1; i++)
50          suma = suma + minimos[i];
51
52      promedio = suma / elem1;
53
54      /*---------------------------------------*/
55      for(i=0; i<elem1; i++)
56      {
57          for(j=0; j<elem2; j++)
58          {
59              printf("elemento[%d][%d] = %d\n", i, j, mat[i][j]);
60          }
61      }
62
63      printf("***************\n");
64      for(i=0; i<elem1 ; i++)
65          printf("elemento[%d] = %d\n", i, minimos[i]);
66
67      printf("promedio = %d\n", promedio);
68
69      return promedio;
70  }
```

Archivo makefile

```
1
2   all: biometria
```

```
3
4  biometria: adelgazamiento.o registraTemplate.o distancias.o
5       archivos.o directorio.o verificacion.o biometria.o
6      gcc -lm `pkg-config opencv --cflags --libs` adelgazamiento.o
7      registraTemplate.o distancias.o archivos.o directorio.o
8      verificacion.o biometria.o -o biometria
9
10 adelgazamiento.o: adelgazamiento.c adelgazamiento.h
11     gcc -c adelgazamiento.c -o adelgazamiento.o
12
13 registraTemplate.o: registraTemplate.c registraTemplate.h
14     gcc -c registraTemplate.c -o registraTemplate.o
15
16 verificacion.o: verificacion.c verificacion.h
17     gcc -c verificacion.c -o verificacion.o
18
19 directorio.o: directorio.c directorio.h
20     gcc -c directorio.c -o directorio.o
21
22 archivos.o: archivos.c archivos.h
23     gcc -c archivos.c -o archivos.o
24
25 distancias.o: distancias.c distancias.h
26     gcc -c distancias.c -o distancias.o
27
28 biometria.o: biometria.c
29     gcc -c biometria.c -o biometria.o
30
31 clean:
32     rm *.o biometria
```

Bibliografía

[1] C. M. Travieso González, M. del Pozo Baños, J. R. Ticay Rivas, "Sistemas Biométricos," pp. 5-21, 2011.

[2] T. Yanagawa, S. Aoki, and T. Ohyama, "Human Finger Vein Images are Diverse and Its Patterns are Useful for Personal Identification," MHF Preprint Series, Faculty of Mathematics, Kyushu University, Fukuoka – JAPAN, April 2007.

[3] S. L. Yang, K. Sakiyama, and I. M. Verbauwhede, "A compact and efficient fingerprint verification system for secure embedded devices," in Conference Record of the Thirty-Seventh Asilomar Conference on Signals, Systems and Computers 2003, pp. 2058-2062.

[4] J. E. Purkinje, "Purkinje's observations on fingerprints and other skin features", Journal of Criminal law and criminology, vol. 31, article 12, 1940.

[5] F. Galton, "Finger Prints", Macmillan and CO, London, 1892.

[6] S. Lim, K. Lee, O. Byeon and T. Kim. "Efficient iris recognition through improvement of feature vector and classifier", Electronics and Telecommunications Research Institute Journal, vol. 23, no. 2, pp. 61-70, 2001.

[7] D. Mulyono and H. S. Jinn, "A study of finger vein biometric for personal identification," in International Symposium on Biometrics and Security Technologies, 2008, pp. 1-8.

[8] T. Y. Zhang and C. Y. Suen, "A fast parallel algorithm for thinning digital patterns," Communications of ACM, vol. 27, pp.236-239, 1984.

[9] X. Sun and Z.M. Ai, "Automatic feature extraction and recognition of fingerprint images", Proceedings of ICSP', 1996.

[10] T. D. Santiago Santiago, A. de J. Hernández Trejo, "Sistema de identificación de personas a través del iris mediante análisis de texturas por filtrado de wavelets,"proyecto terminal, División de Ciencias Básicas e Ingeniería, Universidad Autónoma Metropolitana Azcapotzalco, México, 2010.

[11] M. Beltrán Acosta, "Sistema computarizado auxiliar en el diagnóstico de enfermedades mediante el método de la iridología,"proyecto terminal, División de

Ciencias Básicas e Ingeniería, Universidad Autónoma Metropolitana Azcapotzalco, México, 2012.

[12] E. T. Gracia Silva, C. de la L. Tapia Corona, "Análisis de gesticulación facial mediante puntos de referencia," proyecto terminal, División de Ciencias Básicas e Ingeniería, Universidad Autónoma Metropolitana Azcapotzalco, México, 2013.

[13] OpenCV (visto el 19 de noviembre de 2013).
Disponible en : http://docs.opencv.org/